AWS IoT Core

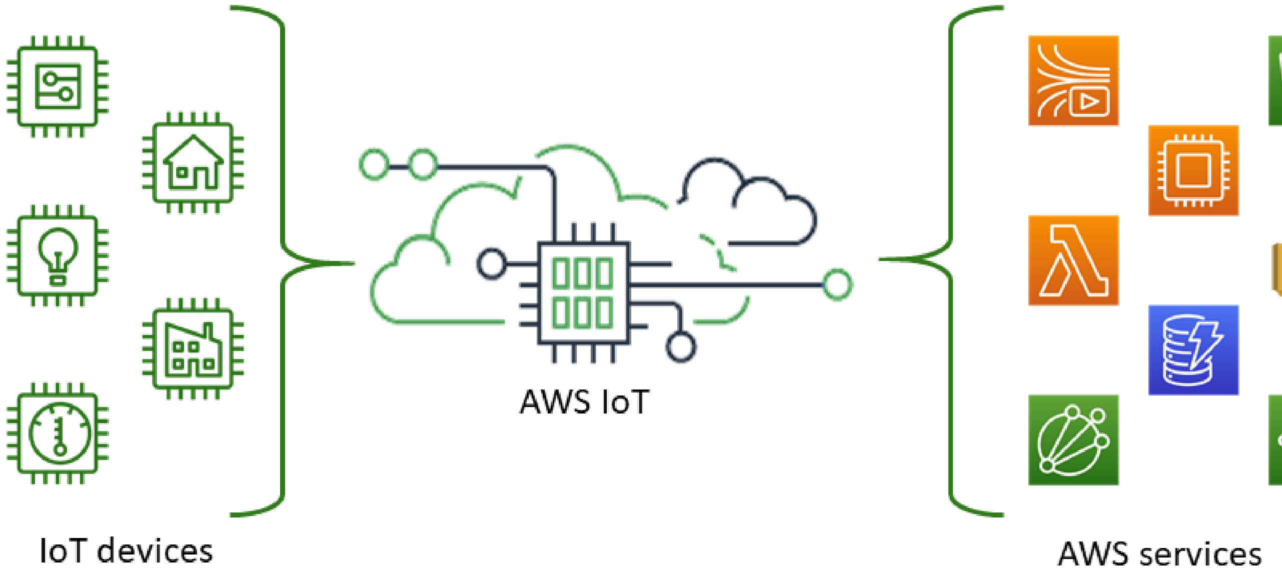
Developer Guide

AWS IoT Core: Developer Guide

Table of Contents

What is AWS IoT?

AWS IoT provides the cloud services that connect your IoT devices to other devices and AWS cloud services. AWS IoT provides device software that can help you integrate your IoT devices into AWS IoT-based solutions. If your devices can connect to AWS IoT, AWS IoT can connect them to the cloud services that AWS provides.



AWS IoT lets you select the most appropriate and up-to-date technologies for your solution. To help you manage and support your IoT devices in the field, AWS IoT Core supports these protocols:

- MQTT (Message Queuing and Telemetry Transport) (p. 78)
- MQTT over WSS (Websockets Secure) (p. 78)
- HTTPS (Hypertext Transfer Protocol - Secure) (p. 82).
- LoRaWAN (Long Range Wide Area Network) (p. 104)

The AWS IoT Core message broker supports devices and clients that use MQTT and MQTT over WSS protocols to publish and subscribe to messages. It also supports devices and clients that use the HTTPS protocol to publish messages.

AWS IoT Core for LoRaWAN helps you connect and manage wireless LoRaWAN (low-power long-range Wide Area Network) devices. AWS IoT Core for LoRaWAN replaces the need for you to develop and operate a LoRaWAN Network Server (LNS).

If you don't require AWS IoT features such as device communications, rules (p. 352), or jobs (p. 537), see AWS Messaging for information about other AWS messaging services that might better fit your requirements.

Whether you're new to IoT or you have years of experience, make sure to review How AWS IoT works (p. 4). This topic helps you understand AWS IoT concepts and terms to help you get started with AWS IoT more effectively.

How to get started with AWS IoT

- **Look** inside AWS IoT and its components in How AWS IoT works (p. 4).
- **Learn** more about AWS IoT (p. 12) from our collection of training materials and videos. This topic also includes a list of services that AWS IoT can connect to, social media links, and links to communication protocol specifications.
- **Connect** your first device to AWS IoT in Getting started with AWS IoT Core (p. 17).
- **Develop** your IoT solutions by Connecting to AWS IoT Core (p. 65) and exploring the AWS IoT Tutorials (p. 117).
- **Test and validate** your IoT devices for secure and reliable communication by using the Device Advisor (p. 898).
- **Manage** your solution by using AWS IoT Core management services such as Fleet indexing service (p. 684), Jobs (p. 537), and AWS IoT Device Defender (p. 714).
- **Analyze** the data from your devices by using the AWS IoT data services (p. 8).

How your devices and apps access AWS IoT

AWS IoT provides the following interfaces for AWS IoT Tutorials (p. 117):

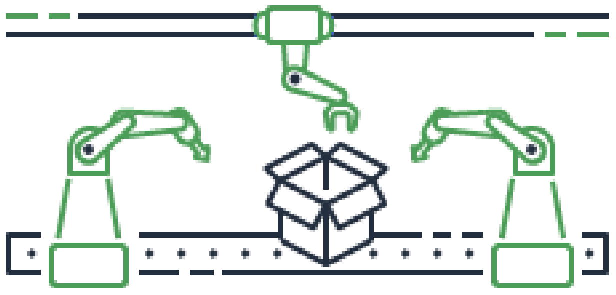
- **AWS IoT Device SDKs**—Build applications on your devices that send messages to and receive messages from AWS IoT. For more information, see *AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client* (p. 990).
- **AWS IoT Core for LoRaWAN**—Connect and manage your long range WAN (LoRaWAN) devices and gateways by using AWS IoT Core for LoRaWAN (p. 104).
- **AWS Command Line Interface (AWS CLI)**—Run commands for AWS IoT on Windows, macOS, and Linux. These commands allow you to create and manage thing objects, certificates, rules, jobs, and policies. To get started, see the AWS Command Line Interface User Guide. For more information about the commands for AWS IoT, see iot in the *AWS CLI Command Reference*.
- **AWS IoT API**—Build your IoT applications using HTTP or HTTPS requests. These API actions allow you to programmatically create and manage thing objects, certificates, rules, and policies. For more information about the API actions for AWS IoT, see Actions in the *AWS IoT API Reference*.
- **AWS SDKs**—Build your IoT applications using language-specific APIs. These SDKs wrap the HTTP/HTTPS API and allow you to program in any of the supported languages. For more information, see AWS SDKs and Tools.

You can also access AWS IoT through the AWS IoT console, which provides a graphical user interface (GUI) through which you can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of your IoT solutions.

What AWS IoT can do

This topic describes some of the solutions that you might need that AWS IoT supports.

IoT in Industry



These are some examples of AWS IoT solutions for industrial use cases that apply IoT technologies to improve the performance and productivity of industrial processes.

Solutions for industrial use cases

- **Use AWS IoT to build predictive quality models in industrial operations**

  See how AWS IoT can collect and analyze data from industrial operations to build predictive quality models. Learn more
- **Use AWS IoT to support predictive maintenance in industrial operations**

  See how AWS IoT can help plan preventive maintenance to reduce unplanned downtime. Learn more

IoT in Home automation

These are some examples of AWS IoT solutions for home automation use cases that apply IoT technologies to build scalable IoT applications that automate household activities using connected home devices.

Solutions for home automation

- **Use AWS IoT in your connected home**

  See how AWS IoT can provide integrated home automation solutions.
- **Use AWS IoT to provide home security and monitoring**

  See how AWS IoT can apply machine learning and edge computing to your home automation solution.

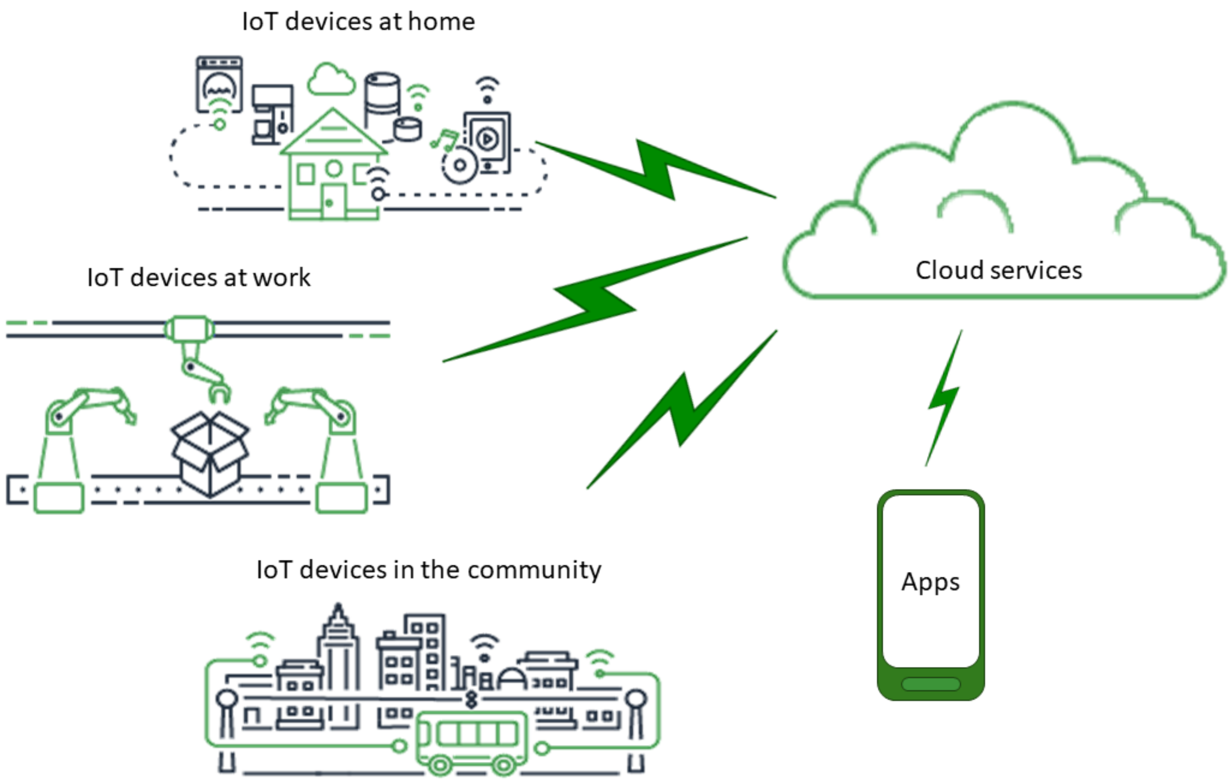
For a list of solutions for industrial, consumer, and commercial use cases, see the AWS IoT Solution Repository.

How AWS IoT works

AWS IoT provides cloud services and device support that you can use to implement IoT solutions. AWS provides many cloud services to support IoT-based applications. So to help you understand where to start, this section provides a diagram and definition of essential concepts to introduce you to the IoT universe.

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.



Apps

Apps give end users access to IoT devices and the features provided by the cloud services to which those devices are connected.

Cloud services

Cloud services are distributed, large-scale data storage and processing services that are connected to the internet. Examples include:

- IoT connection and management services.

  AWS IoT is an example of an IoT connection and management service.

- Compute services, such as Amazon Elastic Compute Cloud and AWS Lambda.
- Database services, such as Amazon DynamoDB

Communications

Devices communicate with cloud services by using various technologies and protocols. Examples include:

- Wi-Fi/Broadband internet
- Broadband cellular data
- Narrow-band cellular data
- Long-range Wide Area Network (LoRaWAN)
- Proprietary RF communications

Devices

A device is a type of hardware that manages interfaces and communications. Devices are usually located in close proximity to the real-world interfaces they monitor and control. Devices can include computing and storage resources, such as microcontrollers, CPU, memory. Examples include:

- Raspberry Pi
- Arduino
- Voice-interface assistants
- LoRaWAN and devices
- Amazon Sidewalk devices
- Custom IoT devices

Interfaces

An interface is a component that connects a device to the physical world.

- User interfaces

  Components that allow devices and users to communicate with each other.
  - Input interfaces

    Enable a user to communicate with a device

    Examples: keypad, button
  - Output interfaces

    Enable a device to communicate with a user

    Examples: Alpha-numeric display, graphical display, indicator light, alarm bell
- Sensors

  Input components that measure or sense something in the outside world in a way that a device understands. Examples include:
  - Temperature sensor (converts temperature to an analog voltage)
  - Humidity sensor (converts relative humidity to an analog voltage)
  - Analog to digital convertor (converts an analog voltage to a numeric value)
  - Ultrasonic distance measuring unit (converts a distance to a numeric value)

- Optical sensor (converts a light level to a numeric value)
- Camera (converts image data to digital data)

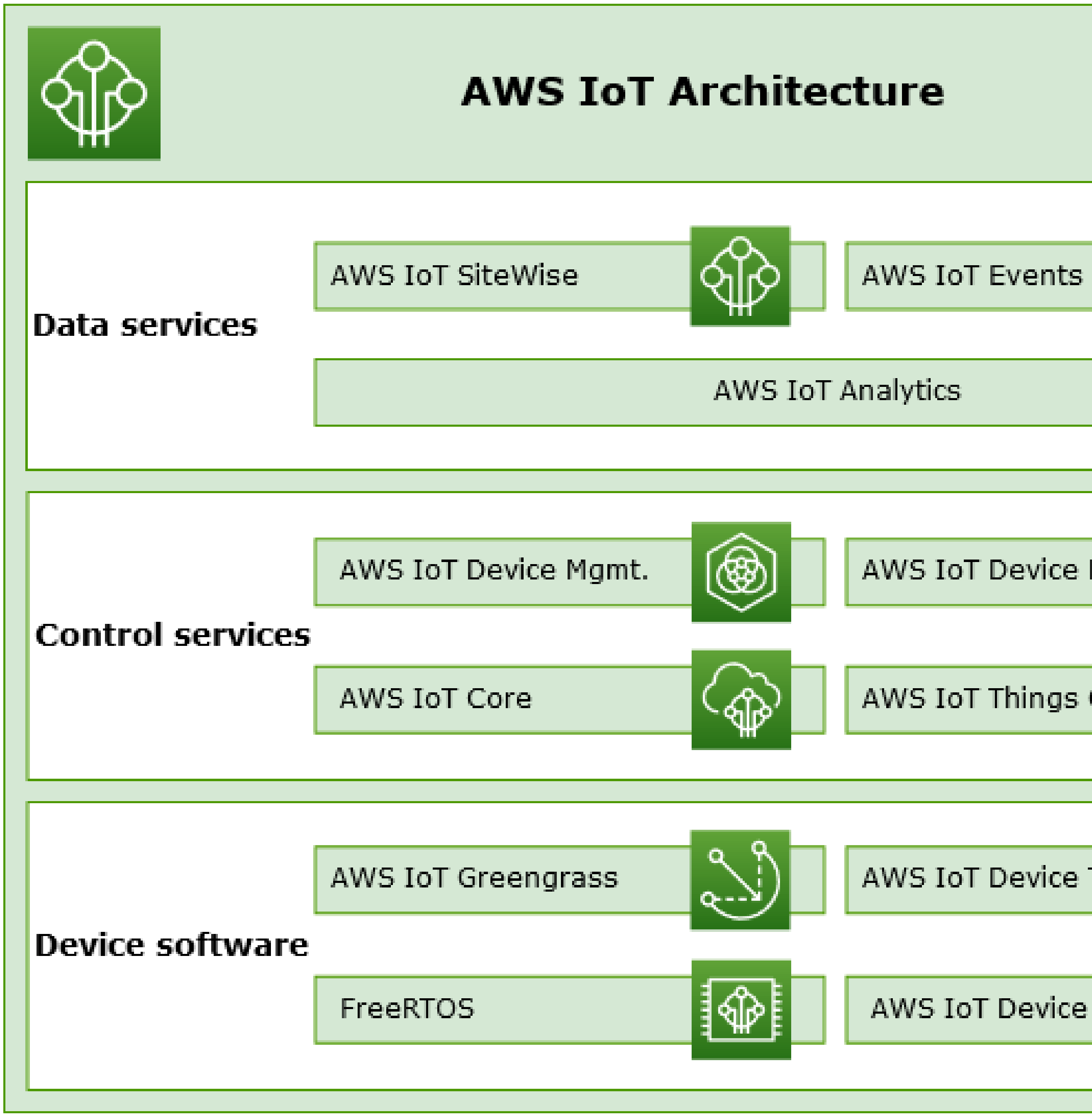
- Actuators

  Output components that the device can use to control something in the outside world. Examples include:

  - Stepper motors (convert electric signals to movement)
  - Relays (control high electric voltages and currents)

AWS IoT services overview

In the IoT universe, AWS IoT provides the services that support the devices that interact with the world and the data that passes between them and AWS IoT. AWS IoT is made up of the services that are shown in this illustration to support your IoT solution.



AWS IoT device software

AWS IoT provides this software to support your IoT devices.

AWS IoT Greengrass

AWS IoT Greengrass extends AWS to edge devices so they can act locally on the data they generate and use the cloud for management, analytics, and durable storage. With AWS IoT Greengrass, connected devices can run AWS Lambda functions, Docker containers, or both, execute predictions based on machine learning models, keep device data in sync, and communicate with other devices securely – even when they are not connected to the Internet.

AWS IoT Device Tester

AWS IoT Device Tester for FreeRTOS and AWS IoT Greengrass is a test automation tool for microcontrollers. AWS IoT Device Tester, test your device to determine if it will run FreeRTOS or AWS IoT Greengrass and interoperate with AWS IoT services.

AWS IoT Device SDKs

The AWS IoT Device and Mobile SDKs (p. 990) help you efficiently connect your devices to AWS IoT. The AWS IoT Device and Mobile SDKs include open-source libraries, developer guides with samples, and porting guides so that you can build innovative IoT products or solutions on your choice of hardware platforms.

FreeRTOS

FreeRTOS is an open source, real-time operating system for microcontrollers that lets you include small, low-power edge devices in your IoT solution. FreeRTOS includes a kernel and a growing set of software libraries that support many applications. FreeRTOS systems can securely connect your small, low-power devices to AWS IoT and support more powerful edge devices running AWS IoT Greengrass.

AWS IoT control services

Connect to the following AWS IoT services to manage the devices in your IoT solution.

AWS IoT Core

AWS IoT Core is a managed cloud service that enables connected devices to securely interact with cloud applications and other devices. AWS IoT Core can support many devices and messages, and it can process and route those messages to AWS endpoints and other devices. With AWS IoT Core, your applications can interact with all of your devices even when they aren't connected.

AWS IoT Device Management

AWS IoT Device Management services help you track, monitor, and manage the plethora of connected devices that make up your devices fleets. AWS IoT Device Management services help you ensure that your IoT devices work properly and securely after they have been deployed. They also provide secure tunneling to access your devices, monitor their health, detect and remotely troubleshoot problems, as well as services to manage device software and firmware updates.

AWS IoT Device Defender

AWS IoT Device Defender helps you secure your fleet of IoT devices. AWS IoT Device Defender continuously audits your IoT configurations to make sure that they aren't deviating from security best practices. AWS IoT Device Defender sends an alert when it detects any gaps in your IoT configuration that might create a security risk, such as identity certificates being shared across multiple devices or a device with a revoked identity certificate trying to connect to AWS IoT Core.

AWS IoT Things Graph

AWS IoT Things Graph is a service that lets you visually connect different devices and web services to build IoT applications. AWS IoT Things Graph provides a visual drag-and-drop interface for connecting and coordinating interactions between devices and web services, so that you can build IoT applications efficiently.

AWS IoT data services

Analyze the data from the devices in your IoT solution and take appropriate action by using the following AWS IoT services.

AWS IoT Analytics

AWS IoT Analytics lets you efficiently run and operationalize sophisticated analytics on massive volumes unstructured IoT data. AWS IoT Analytics automates each difficult step that is required to analyze data from IoT devices. AWS IoT Analytics filters, transforms, and enriches IoT data before storing it in a time-series data store for analysis. You can analyze your data by running one-time or scheduled queries using the built-in SQL query engine or machine learning.

AWS IoT SiteWise

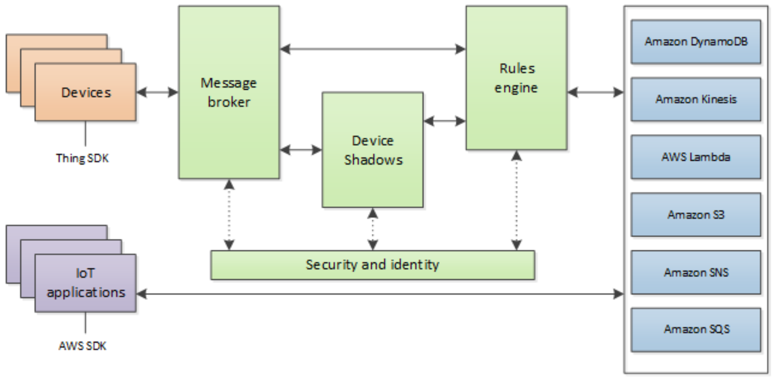
AWS IoT SiteWise collects, stores, organizes, and monitors data passed from industrial equipment by MQTT messages or APIs at scale by providing software that runs on a gateway in your facilities. The gateway securely connects to your on-premises data servers and automates the process of collecting and organizing the data and sending it to the AWS Cloud.

AWS IoT Events

AWS IoT Events detects and responds to events from IoT sensors and applications. Events are patterns of data that identify more complicated circumstances than expected, such as motion detectors using movement signals to activate lights and security cameras. AWS IoT Events continuously monitors data from multiple IoT sensors and applications, and integrates with other services, such as AWS IoT Core, IoT SiteWise, DynamoDB, and others to enable early detection and unique insights.

AWS IoT Core services

AWS IoT Core provides the services that connect your IoT devices to the AWS Cloud so that other cloud services and applications can interact with your internet-connected devices.



The next section describes each of the AWS IoT Core services shown in the illustration.

AWS IoT Core messaging services

The AWS IoT Core connectivity services provide secure communication with the IoT devices and manage the messages that pass between them and AWS IoT.

Device gateway

Enables devices to securely and efficiently communicate with AWS IoT. Device communication is secured by secure protocols that use of X.509 certificates.

Message broker

Provides a secure mechanism for devices and AWS IoT applications to publish and receive messages from each other. You can use either the MQTT protocol directly or MQTT over WebSocket to publish and subscribe. For more information about the protocols that AWS IoT supports, see the section called "Device communication protocols" (p. 76). Devices and clients can also use the HTTP REST interface to publish data to the message broker.

The message broker distributes device data to devices that have subscribed to it and to other AWS IoT Core services, such as the Device Shadow service and the rules engine.

AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN makes is possible to set up a private LoRaWAN network by connecting your LoRaWAN devices and gateways to AWS without the need to develop and operate a LoRaWAN Network Server (LNS). Messages received from LoRaWAN devices are sent to the rules engine where they can be formatted and sent to other AWS services.

Rules engine

The Rules engine connects data from the message broker to other AWS services for storage and additional processing. For example, you can insert, update, or query a DynamoDB table or invoke a Lambda function based on an expression that you defined in the Rules engine. You can use an SQL-based language to select data from message payloads, and then process and send the data to other services, such as Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, and AWS Lambda. You can also create rules that republish messages to the message broker and on to other subscribers. For more information, see Rules for AWS IoT (p. 352).

AWS IoT Core control services

The AWS IoT Core control services provide device security, management, and registration features.

Custom Authentication service

You can define custom authorizers that allow you to manage your own authentication and authorization strategy using a custom authentication service and a Lambda function. Custom authorizers allow AWS IoT to authenticate your devices and authorize operations using bearer token authentication and authorization strategies.

Custom authorizers can implement various authentication strategies; for example, JSON Web Token verification or OAuth provider callout. They must return policy documents that are used by the device gateway to authorize MQTT operations. For more information, see Custom authentication (p. 222).

Device Provisioning service

Allows you to provision devices using a template that describes the resources required for your device: a *thing object*, a certificate, and one or more policies. A thing object is an entry in the registry that contains attributes that describe a device. Devices use certificates to authenticate with AWS IoT. Policies determine which operations a device can perform in AWS IoT.

The templates contain variables that are replaced by values in a dictionary (map). You can use the same template to provision multiple devices just by passing in different values for the template variables in the dictionary. For more information, see Device provisioning (p. 658).

Group registry

Groups allow you to manage several devices at once by categorizing them into groups. Groups can also contain groups—you can build a hierarchy of groups. Any action that you perform on a parent group will apply to its child groups. The same action also applies to all the devices in the parent group and all devices in the child groups. Permissions granted to a group will apply to all devices in the group and in all of its child groups. For more information, see Managing devices with AWS IoT (p. 172).

Jobs service

Allows you to define a set of remote operations that are sent to and run on one or more devices connected to AWS IoT. For example, you can define a job that instructs a set of devices to download and install application or firmware updates, reboot, rotate certificates, or perform remote troubleshooting operations.

To create a job, you specify a description of the remote operations to be performed and a list of targets that should perform them. The targets can be individual devices, groups or both. For more information, see Jobs (p. 537).

Registry

Organizes the resources associated with each device in the AWS Cloud. You register your devices and associate up to three custom attributes with each one. You can also associate certificates and MQTT client IDs with each device to improve your ability to manage and troubleshoot them. For more information, see Managing devices with AWS IoT (p. 172).

Security and Identity service

Provides shared responsibility for security in the AWS Cloud. Your devices must keep their credentials safe to securely send data to the message broker. The message broker and rules engine use AWS security features to send data securely to devices or other AWS services. For more information, see Authentication (p. 200).

AWS IoT Core data services

The AWS IoT Core data services help your IoT solutions provide a reliable application experience even with devices that are not always connected.

Device shadow

A JSON document used to store and retrieve current state information for a device.

Device Shadow service

The Device Shadow service maintains a device's state so that applications can communicate with a device whether the device is online or not. When a device is offline, the Device Shadow service manages its data for connected applications. When the device reconnects, it synchronizes its state with that of its shadow in the Device Shadow service. Your devices can also publish their current state to a shadow for use by applications or other devices that might not be connected all the time. For more information, see AWS IoT Device Shadow service (p. 491).

AWS IoT Core support service

Alexa Voice Service (AVS) Integration for AWS IoT

Brings Alexa Voice to any connected device. AVS for AWS IoT reduces the cost and complexity of integrating Alexa. This feature uses AWS IoT to offload intensive computational and memory audio tasks from the device to the cloud. Because of the resulting reduction in the engineering bill of materials (EBOM) cost, device makers can cost-effectively bring Alexa to resource-constrained IoT devices, and enable consumers to talk directly to Alexa in parts of their home, office, or hotel rooms for an ambient experience.

AVS for AWS IoT enables Alexa built-in functionality on MCUs, such as the ARM Cortex M class with less than 1 MB embedded RAM. To do so, AVS offloads memory and compute tasks to a virtual Alexa Built-in device in the cloud. This reduces EBOM cost by up to 50 percent. For more information, see Alexa Voice Service (AVS) Integration for AWS IoT (p. 980).

Amazon Sidewalk Integration for AWS IoT Core

Amazon Sidewalk is a shared network that improves connectivity options to help devices work together better. Amazon Sidewalk supports a wide range of customer devices such as those that locate pets or valuables, those that provide smart home security and lighting control, and those that provide remote diagnostics for appliances and tools. Amazon Sidewalk Integration for AWS IoT Core makes it possible for device manufacturers to add their Sidewalk device fleet to the AWS Cloud.

For more information, see Amazon Sidewalk Integration for AWS IoT Core (p. 988)

Learn more about AWS IoT

This topic helps you get familiar with the world of AWS IoT. You can get general information about how IoT solutions are applied in various use cases, training resources, links to social media for AWS IoT and all other AWS services, and a list of services and communication protocols that AWS IoT uses.

Training resources for AWS IoT

We provide these training courses to help you learn about AWS IoT and how to apply them to your solution design.

- **Introduction to AWS IoT**

  A video overview of AWS IoT and its core services.
- **Deep Dive into AWS IoT Authentication and Authorization**

  An advanced course that explores the concepts of AWS IoT authentication and authorization. You will learn how to authenticate and authorize clients to access the AWS IoT control plane and data plane APIs.
- **Internet of Things Foundation Series**

  A learning path of IoT eLearning modules on different IoT technologies and features.

AWS IoT resources and guides

These are in-depth technical resources on specific aspects of AWS IoT.

- **IoT Lens – AWS Well-Architected Framework**

  A PDF document that describes the best practices for architecting your IoT applications on AWS.
- **Designing MQTT Topics for AWS IoT Core**

  A PDF document that describes the best practices for designing MQTT topics in AWS IoT Core and leveraging AWS IoT Core features with MQTT.
- **AWS IoT Resources**

  IoT-specific resources, such as Technical Guides, Reference Architectures, eBooks, and curated blog posts, presented in a searchable index.
- **IoT Atlas**

  Overviews on how to solve common IoT design problems. The *IoT Atlas* provides in-depth looks into the design challenges that you are likely to encounter while developing your IoT solution.
- **AWS Whitepapers & Guides**

  Our current collection of whitepapers and guides on AWS IoT and other AWS technologies.

AWS IoT in social media

These social media channels provide information about AWS IoT and AWS-related topics.

- The Internet of Things on AWS – Official Blog
- AWS IoT videos in the Amazon Web Services channel on YouTube

These social media accounts cover all AWS services, including AWS IoT

- The Amazon Web Services channel on YouTube
- Amazon Web Services on Twitter
- Amazon Web Services on Facebook
- Amazon Web Services on Instagram
- Amazon Web Services on LinkedIn

AWS services used by the AWS IoT Core rules engine

The AWS IoT Core rules engine can connect to these AWS services.

- **Amazon DynamoDB**

  Amazon DynamoDB is a scalable, NoSQL database service that provides fast and predictable database performance.
- **Amazon Kinesis**

  Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information. Amazon Kinesis can ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics, and other applications.
- **AWS Lambda**

  AWS Lambda lets you run code without provisioning or managing servers. You can set up your code to automatically trigger from AWS IoT data and events or call it directly from a web or mobile app.
- **Amazon Simple Storage Service**

  Amazon Simple Storage Service (Amazon S3) can store and retrieve any amount of data at any time, from anywhere on the web. AWS IoT rules can send data to Amazon S3 for storage.
- **Amazon Simple Notification Service**

  Amazon Simple Notification Service (Amazon SNS) is a web service that enables applications, end users, and devices to send and receive notifications from the cloud.
- **Amazon Simple Queue Service**

  Amazon Simple Queue Service (Amazon SQS) is a message queuing service that decouples and scales microservices, distributed systems, and serverless applications.
- **Amazon Elasticsearch Service**

  Amazon Elasticsearch Service (Amazon ES) is a managed service that makes it easy to deploy, operate, and scale Elasticsearch, a popular open-source search and analytics engine.
- **Amazon Machine Learning**

  Amazon Machine Learning can create machine learning (ML) models by finding patterns in your IoT data. The service uses these models to process new data and generate predictions for your application.

- **Amazon CloudWatch**

  Amazon CloudWatch provides a reliable, scalable, and flexible monitoring solution to help set up, manage, and scale your own monitoring systems and infrastructure.

Communication protocols supported by AWS IoT Core

These topics provide more information about the communication protocols used by AWS IoT. For more information about the protocols used by AWS IoT and connecting devices and services to AWS IoT, see Connecting to AWS IoT Core (p. 65).

- **MQTT (Message Queuing Telemetry Transport)**

  The home page of the MQTT.org site where you can find the MQTT protocol specifications. For more information about how AWS IoT supports MQTT, see MQTT (p. 78).
- **HTTPS (Hypertext Transfer Protocol - Secure)**

  Devices and apps can access AWS IoT services by using HTTPS.
- **LoRaWAN (Long Range Wide Area Network)**

  LoRaWAN devices and gateways can connect to AWS IoT Core by using AWS IoT Core for LoRaWAN.
- **TLS (Transport Layer Security) v1.2**

  The specification of the TLS v1.2 (RFC 5246). AWS IoT uses TLS v1.2 to establish secure connections between devices and AWS IoT.

What's new in the AWS IoT console

We're in the process of updating the user interface of the AWS IoT console to a new experience. We're updating the user interface in stages, so some pages in the console will have a new experience, some might have both the original and the new experience, and some might have only the original experience.

This table displays the current state of individual areas of the AWS IoT console user interface.

AWS IoT console user interface status

Console page	Original experience	New experience	Comments
Monitor	Available	Available	
Activity	Available	Not available yet	
Onboard - Get started	Available	Not available yet	
Onboard - Fleet provisioning templates	Available	Not available yet	
Manage - Things	Available	Not available yet	
Manage - Types	Available	Not available yet	
Manage - Thing groups	Available	Not available yet	
Manage - Billing groups	Available	Not available yet	

Console page	Original experience	New experience	Comments
Manage - Jobs	Available	Not available yet	
Manage - Tunnels	Available	Not available yet	
Fleet Hub - Get started	Available	Not available yet	Not available in all AWS Regions
Fleet Hub - Applications	Available	Not available yet	Not available in all AWS Regions
Greengrass - Getting started	Available	Not available yet	
Greengrass - Core devices	Available	Not available yet	
Greengrass - Components	Available	Not available yet	
Greengrass - Deployments	Available	Not available yet	
Greengrass - Classic (V1)	Available	Not available yet	
Wireless connectivity - Intro	Not available	Available	Not available in all AWS Regions
Wireless connectivity - Gateways	Not available	Available	Not available in all AWS Regions
Wireless connectivity - Devices	Not available	Available	Not available in all AWS Regions
Wireless connectivity - Profiles	Not available	Available	Not available in all AWS Regions
Wireless connectivity - Destinations	Not available	Available	Not available in all AWS Regions
Secure - Certificates	Available	Not available yet	
Secure - Policies	Available	Not available yet	
Secure - CAs	Available	Not available yet	
Secure - Role Aliases	Available	Not available yet	
Secure - Authorizers	Available	Not available yet	
Defend - Intro	Available	Not available yet	
Defend - Audit	Available	Not available yet	
Defend - Detect	Available	Not available yet	
Defend - Mitigation actions	Available	Not available yet	

Console page	Original experience	New experience	Comments
Defend - Settings	Available	Not available yet	Not available in all AWS Regions
Act - Rules	Available	Not available yet	
Act - Destinations	Available	Not available yet	
Test - Device Advisor	Available	Not available yet	Not available in all AWS Regions
Test - MQTT test client	Available	Not available yet	
Software	Available	Not available yet	
Settings	Available	Not available yet	
Learn	Available	Not available yet	

Legend

Status values

- **Available**

  This user interface experience can be used.
- **Not available**

  This user interface experience can't be used.
- **Not available yet**

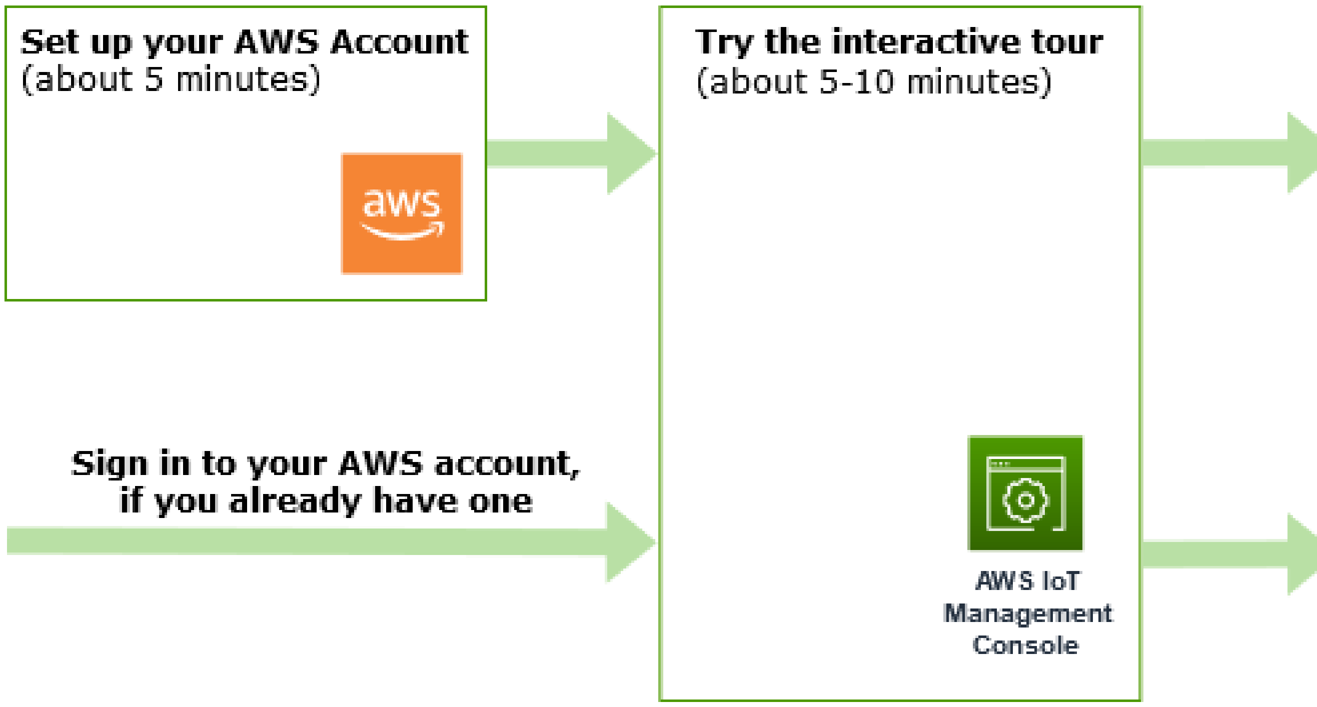
  The new user interface experience is being worked on, but it's not ready, yet.
- **In progress**

  The new user interface experience is in the process of being updated. Some pages might still have the original user experience, however.

Getting started with AWS IoT Core

AWS IoT Core services connect IoT devices to AWS IoT services and other AWS services. AWS IoT Core includes the device gateway and the message broker, which connect and process messages between your IoT devices and the cloud.

Here's how you can get started with AWS IoT Core and AWS IoT.



This section presents a tour of the AWS IoT Core to introduce its key services and provides several examples of how to connect a device to AWS IoT Core and pass messages between them. Passing messages between devices and the cloud is fundamental to every IoT solution and is how your devices can interact with other AWS services.

- **Set up your AWS account (p. 18)**

  Before you can use AWS IoT services, you must set up an AWS account. If you already have an AWS account and an IAM user for yourself, you can use them and skip this step.
- **Try the interactive demo (p. 20)**

  This demo is best if you want to see what a basic AWS IoT solution can do without connecting a device or downloading any software. The interactive demo presents a simulated solution built on AWS IoT Core services that illustrates how they interact.
- **Try the quick connect tutorial (p. 24)**

  This tutorial is best if you want to quickly get started with AWS IoT and see how it works in a limited scenario. In this tutorial, you'll need a device and you'll install some AWS IoT software on it. If you don't have an IoT device, you can use your Windows, Linux, or macOS personal computer as a device for this tutorial. If you want to try AWS IoT, but you don't have a device, try the next option.
- **Explore AWS IoT Core services with a hands-on tutorial (p. 32)**

This tutorial is best for developers who want to get started with AWS IoT so they can continue to explore other AWS IoT Core features such as the rules engine and shadows. This tutorial follows a process similar to the quick connect tutorial, but provides more details on each step to enable a smoother transition to the more advanced tutorials.

Note
If you want to try more than one of these getting started tutorials or repeat the same tutorial, you should delete the thing object that you created from an earlier tutorial before you start another one. If you don't delete the thing object from an earlier tutorial, you will need to use a different thing name for subsequent tutorials. This is because the thing name must be unique in your account and AWS Region.

For more information about AWS IoT Core, see What Is AWS IoT Core (p. 1)?

Topics

Set up your AWS account

Before you use AWS IoT Core for the first time, complete the following tasks:

- Sign up for an AWS account (p. 18)
- Create a user and grant permissions (p. 19)
- Open the AWS IoT console (p. 19)

If you already have an AWS account and an IAM user for yourself, you can use them and skip ahead to the section called "Open the AWS IoT console" (p. 19).

Sign up for an AWS account

When you sign up for AWS, your account is automatically signed up for all services in AWS. If you have an AWS account already, skip this procedure. If you don't have an AWS account, use the following procedure to create one.

You can expect to spend about 5 minutes setting up your AWS account.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   Note
   Save your AWS account number, because you need it for the next task.

Create a user and grant permissions

This procedure describes how to create an IAM user for yourself and add that user to a group that has administrative permissions from an attached managed policy. IAM is the AWS service that manages users of and access to AWS resources. You must do this so that you can create the AWS IoT resources in your account and grant them permission to do what they need to do.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   Note
   We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    Note
    You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access management and Example policies.

Open the AWS IoT console

Most of the console-oriented topics in this section start from the AWS IoT console. If you aren't already signed in to your AWS account, sign in, then open the AWS IoT console and continue to the next section to continue getting started with AWS IoT.

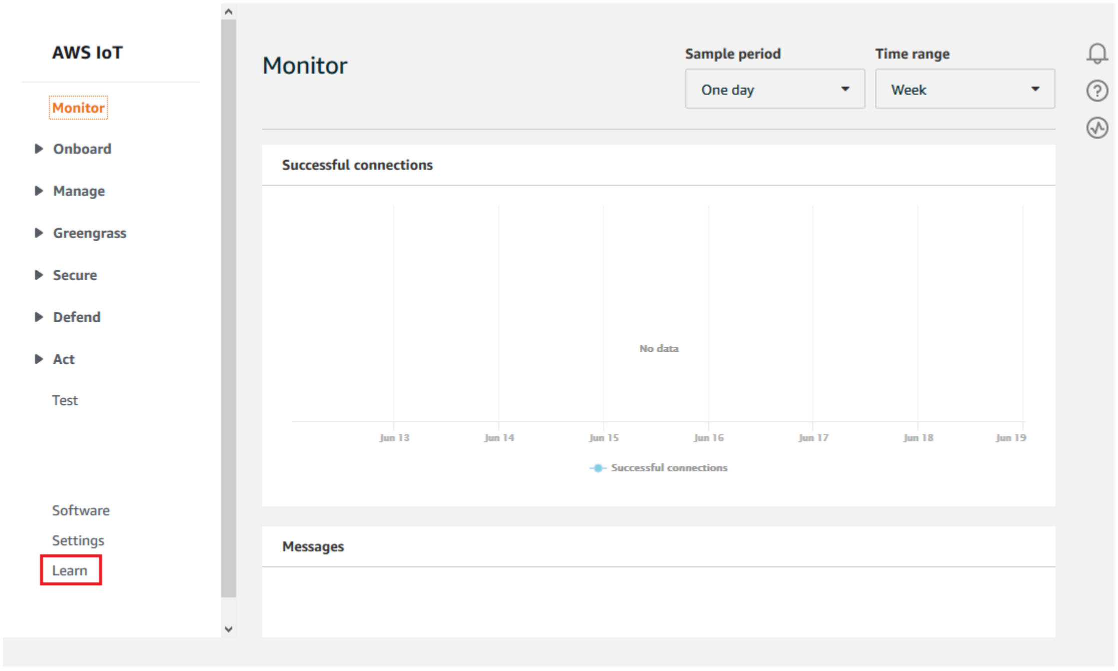
Try the AWS IoT Core interactive demo

The interactive demo shows the components of a simple IoT solution built on AWS IoT and how they interact with AWS IoT Core services. It does not create any AWS IoT resources nor does it require that you download any software or do any coding.

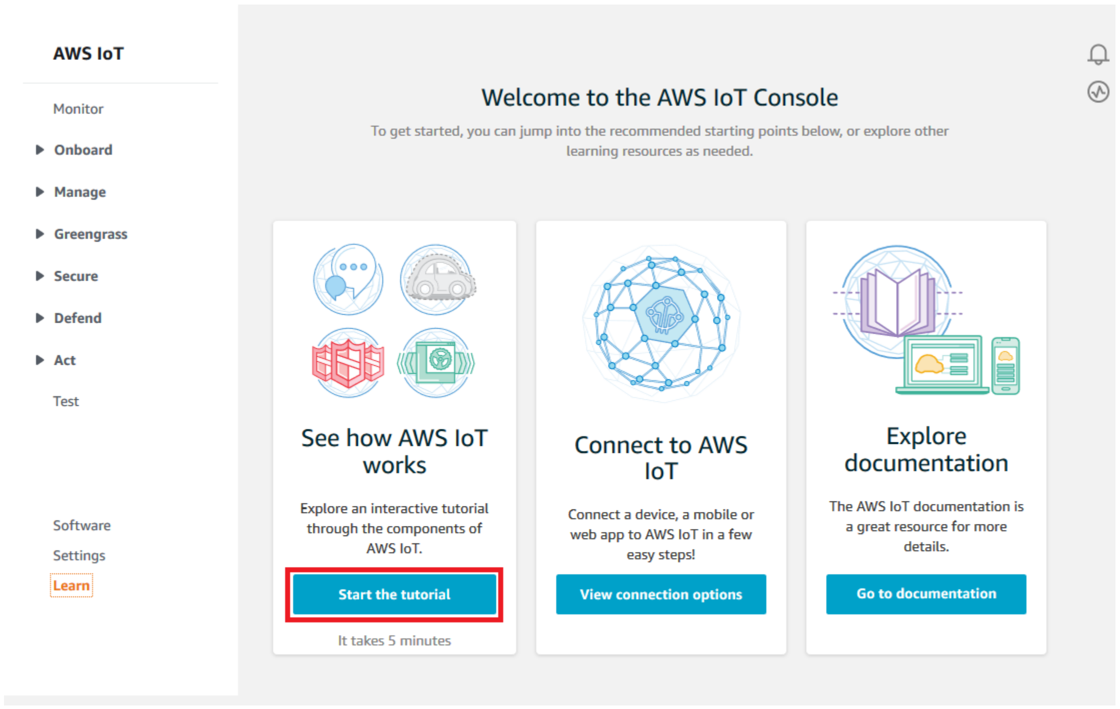
You can expect to spend about 10 minutes with this demo. The tile in the console says that it takes 5 minutes, and you can probably do it in 5 minutes, but 10 minutes will give you time to explore each of the different steps more thoroughly.

To run the AWS IoT Core interactive demo

1. Open your AWS IoT console and from the left menu, choose **Learn**.



2. On the **See how AWS IoT works** tile, choose **Start the tutorial**.



Welcome to the AWS IoT Console

To get started, you can jump into the recommended starting points below, or explore other learning resources as needed.

See how AWS IoT works

Explore an interactive tutorial through the components of AWS IoT.

It takes 5 minutes

Connect to AWS IoT

Connect a device, a mobile or web app to AWS IoT in a few easy steps!

Explore documentation

The AWS IoT documentation is a great resource for more details.

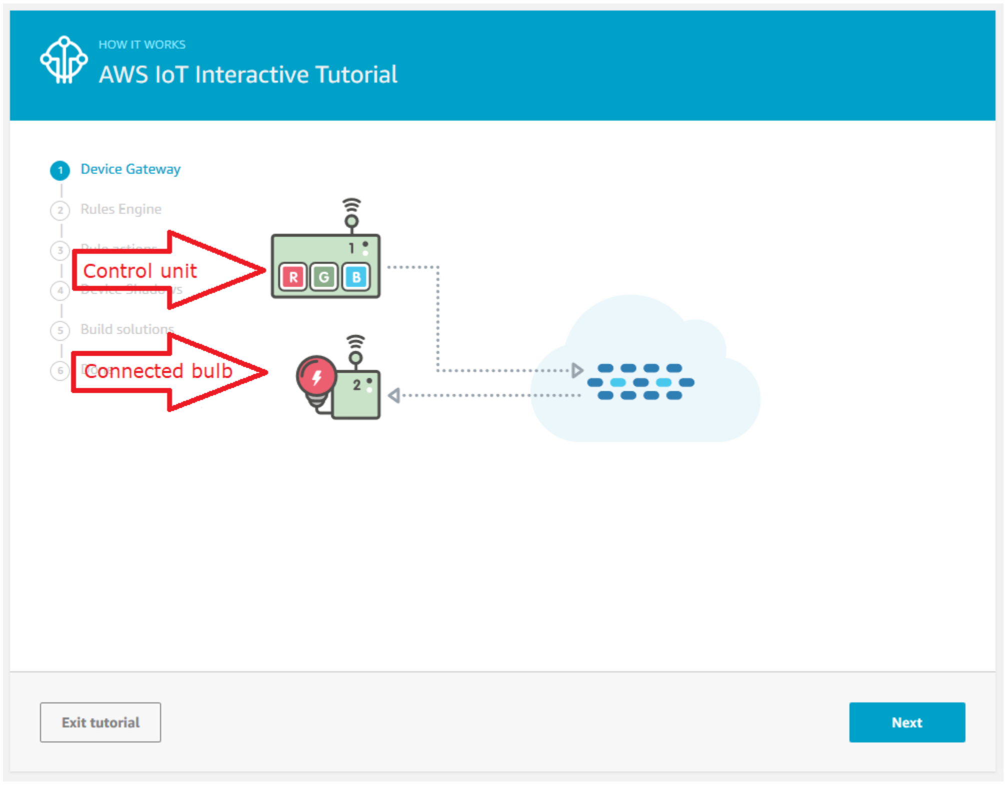
3. On the **AWS IoT Interactive Tutorial** page, review the tutorial overview to get an idea of what you'll learn.

   Choose **Start tutorial** when you're ready to continue.

4. **Interactive demo step 1**

   Read the instructions in the right panel, which describe the IoT system components.

   Choose the different buttons on the control unit and watch the effect on the connected light bulb.



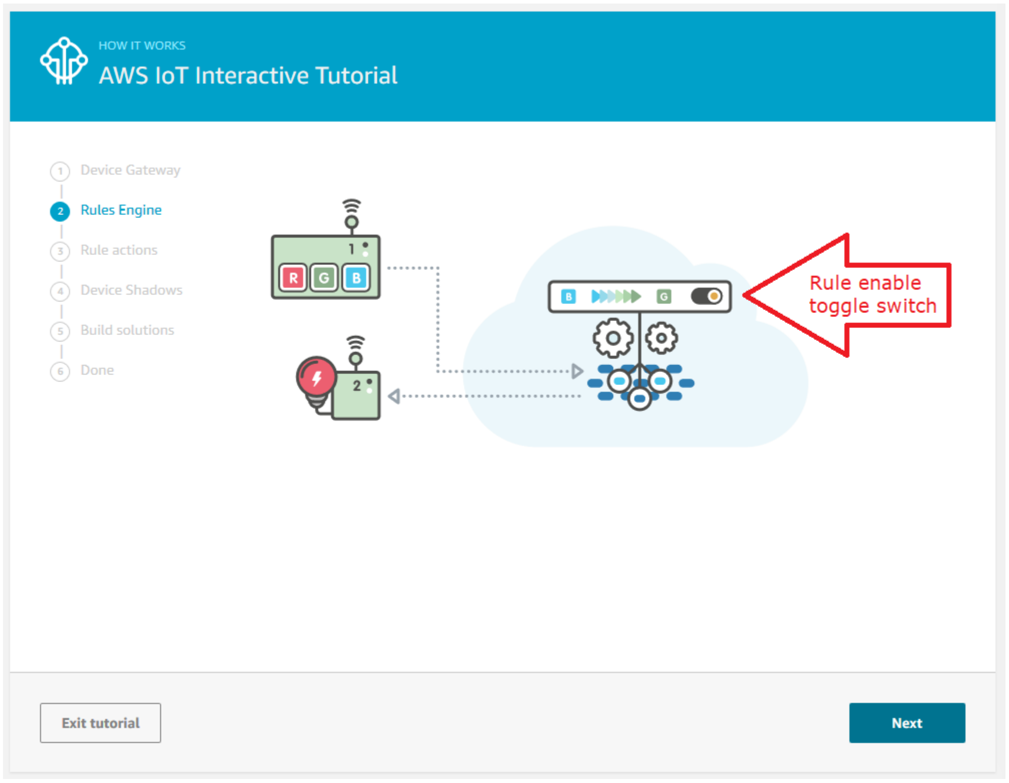
Choose **Next** to continue.

5. **Interactive demo step 2**

   Read the instructions in the right panel, which describe the new components.

   Choose the different buttons on the control unit and watch the effect on the connected light bulb.

   Move the **Rule enable toggle switch** to disable the rule and choose different buttons on the control unit to see how the bulb behaves differently.



Choose **Next** to continue.

6. **Interactive demo step 3**

   This step adds another rule. Read the instructions in the right panel to understand the new rule, and then try the interactions described in the right panel.

   Choose **Next** to continue.

7. **Interactive demo step 4**

   This step adds a device shadow to AWS IoT and a power toggle to the connected bulb.

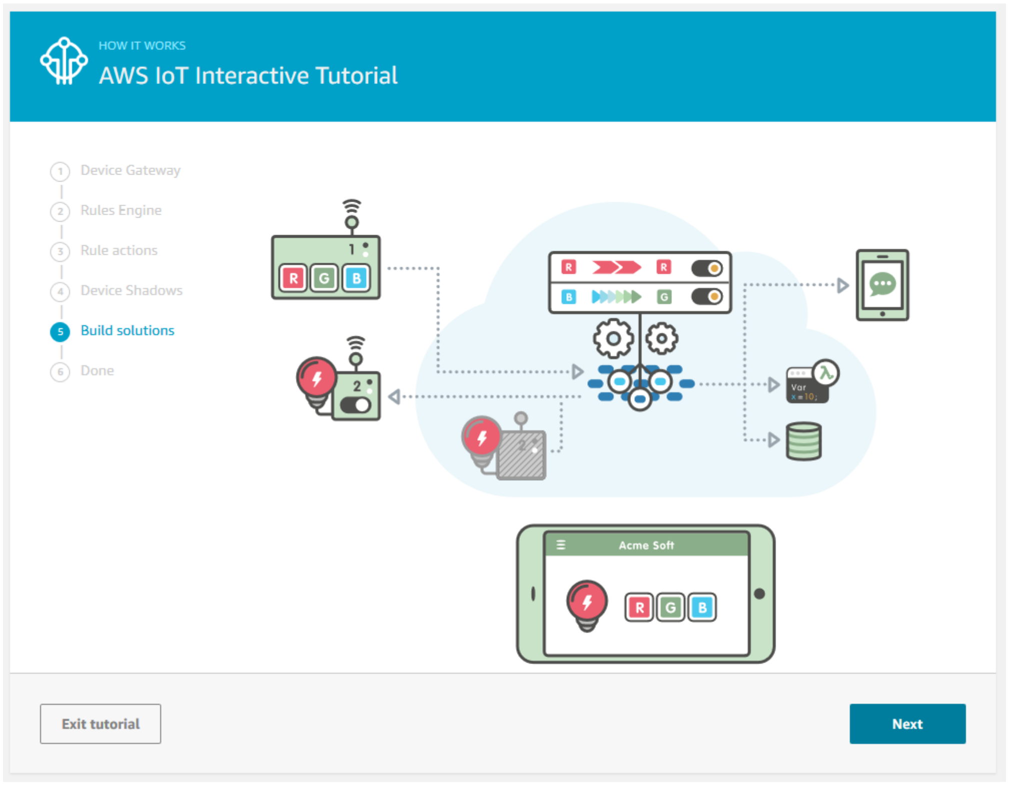
   Read the instructions in the right panel to understand how the new components will affect the system operation and then try the interactions described in the right panel.

   Choose **Next** to continue.

8. **Interactive demo step 5**

   This step adds a mobile device with an app at the bottom of the screen, which interacts with the IoT solution.

   Read the instructions in the right panel to understand how the new components work and then try the interactions described in the right panel.



Choose **Next** to continue.

9. **Interactive demo step 6**

   This is the final step of the demo where you can review and explore how this IoT solution works.

   Choose **Get started** to continue.

10. On the **Register a thing** page, choose **Cancel** to exit the demo.

Try the AWS IoT quick connect

In this tutorial, you'll create your first thing object, connect a device to it, and watch it send MQTT messages.

You can expect to spend 15-20 minutes on this tutorial.

This tutorial is best for people who want to quickly get started with AWS IoT to see how it works in a limited scenario. If you're looking for an example that will get you started so that you can explore more features and services, try Explore AWS IoT Core services in hands-on tutorial (p. 32).

In this tutorial, you'll download and run software on a device that connects to a *thing object* in AWS IoT Core as part of a very small IoT solution. The device can be an IoT device, such as a Raspberry Pi, or it can also be a computer that is running Linux, OS and OSX, or Windows. If you're looking to connect a Long Range WAN (LoRaWAN) device to AWS IoT, refer to the tutorial Connecting devices and gateways to AWS IoT Core for LoRaWAN (p. 104).

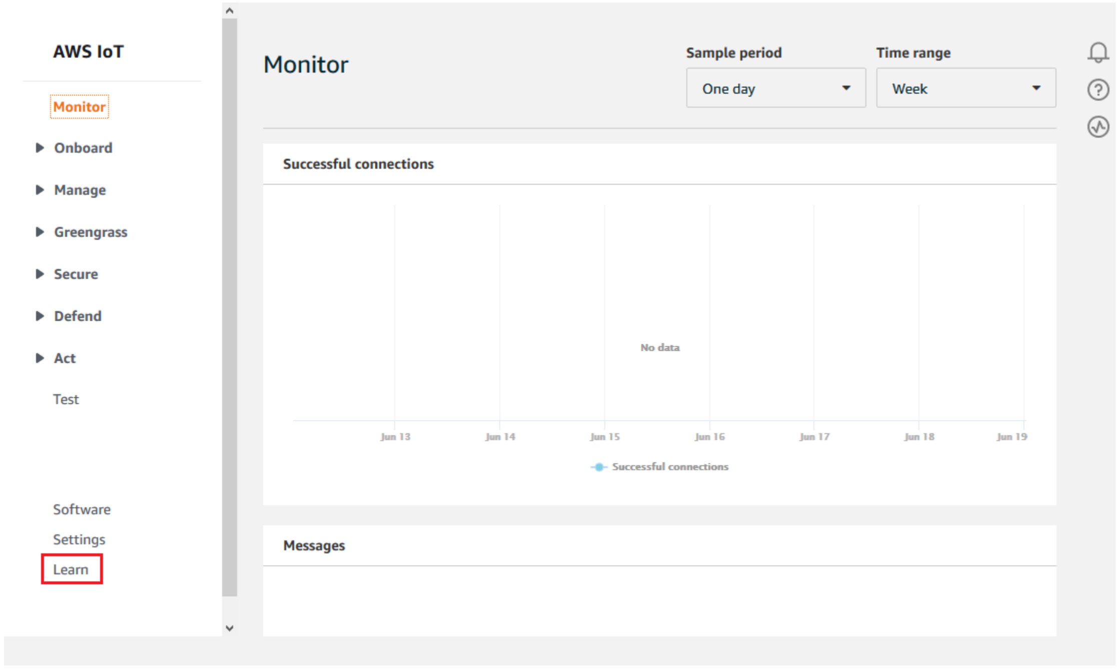
If your device supports a browser that can run the AWS IoT console, we recommend you complete this tutorial on that device.

Note
If your device doesn't have a compatible browser, follow this tutorial on a computer. When the procedure asks you to download the file, download it to your computer, and then transfer the downloaded file to your device using by Secure Copy (SCP) or a similar process.

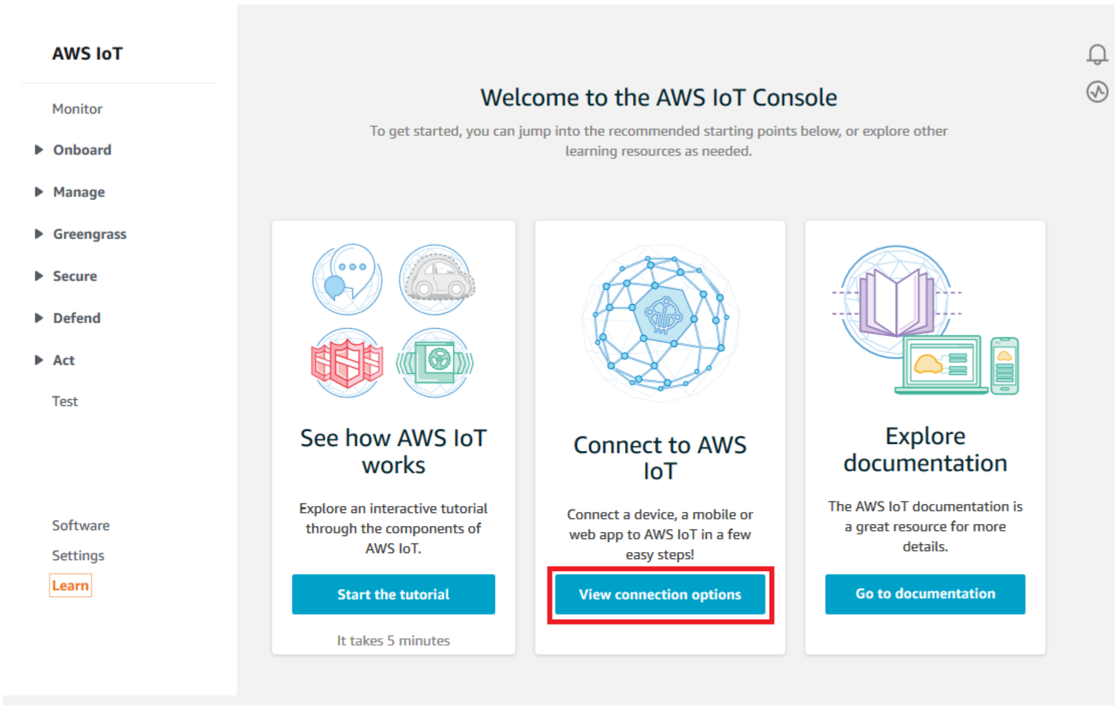
Step 1. Start the tutorial

If possible, complete this procedure on your device; otherwise, be ready to transfer a file to your device later in this procedure.

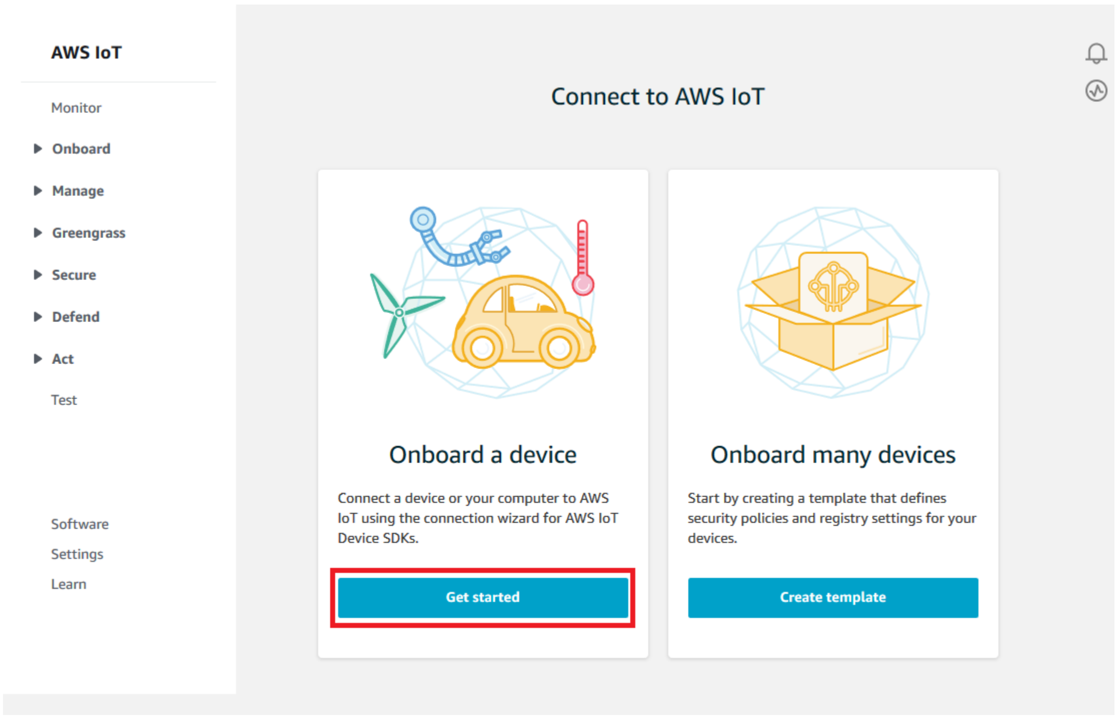
1. Open your AWS IoT console and from the left menu, choose **Learn**.



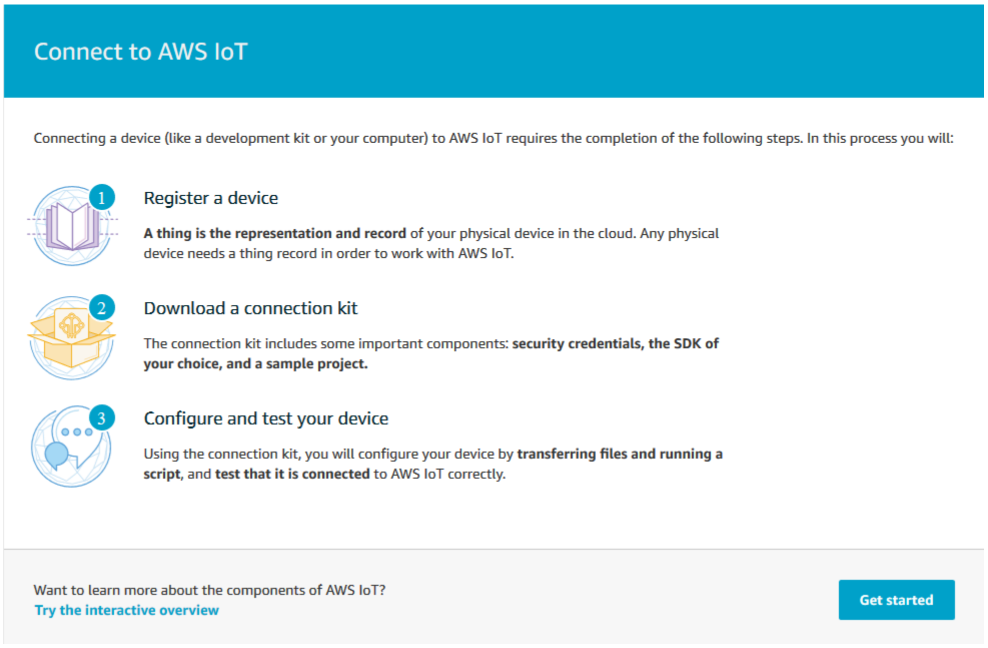
2. On the **Connect to AWS IoT** tile, choose **View connection options**.


AWS IoT
Monitor
Onboard
Manage
Greengrass
Secure
Defend
Act
Test
Software
Settings
Learn
Welcome to the AWS IoT Console
To get started, you can jump into the recommended starting points below, or explore other learning resources as needed.
See how AWS IoT works
Explore an interactive tutorial through the components of AWS IoT.
Start the tutorial
It takes 5 minutes
Connect to AWS IoT
Connect a device, a mobile or web app to AWS IoT in a few easy steps!
View connection options
Explore documentation
The AWS IoT documentation is a great resource for more details.
Go to documentation

3. In the **Onboard a device** tile, choose **Get started**.


AWS IoT
Monitor
Onboard
Manage
Greengrass
Secure
Defend
Act
Test
Software
Settings
Learn
Connect to AWS IoT
Onboard a device
Connect a device or your computer to AWS IoT using the connection wizard for AWS IoT Device SDKs.
Get started
Onboard many devices
Start by creating a template that defines security policies and registry settings for your devices.
Create template

4. Review the steps that describe what you'll do in this tutorial. When you're ready to continue, choose **Get started**.


Connect to AWS IoT

Connecting a device (like a development kit or your computer) to AWS IoT requires the completion of the following steps. In this process you will:

Register a device

A thing is the representation and record of your physical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT.

Download a connection kit

The connection kit includes some important components: **security credentials, the SDK of your choice, and a sample project.**

Configure and test your device

Using the connection kit, you will configure your device by **transferring files and running a script**, and **test that it is connected** to AWS IoT correctly.

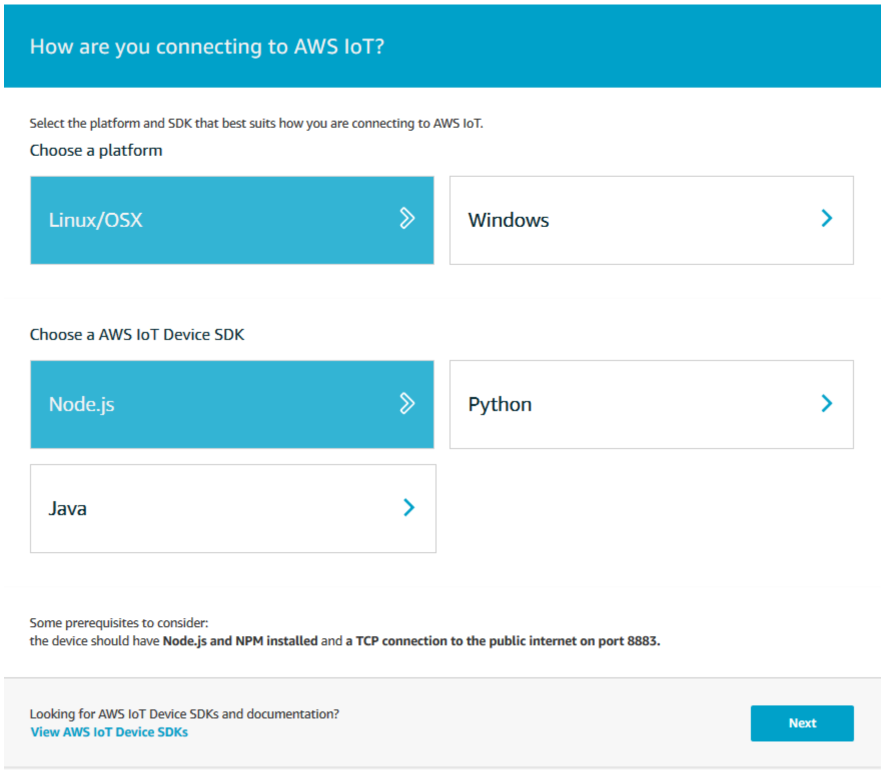
Want to learn more about the components of AWS IoT?
Try the interactive overview

Step 2. Create a thing object

1. On the **How are you connecting to AWS IoT?** page, choose the platform and the language of the AWS IoT Device SDK that you want to use. This example uses the Linux/OSX platform and the Node.js SDK.

   Note
   Be sure to check the list of prerequisite software required by your chosen SDK at the bottom of the console page.
   You must have the required software installed on your target computer before you continue to the next step.

   After you choose the platform and device SDK language, choose **Next**.


How are you connecting to AWS IoT?

Select the platform and SDK that best suits how you are connecting to AWS IoT.

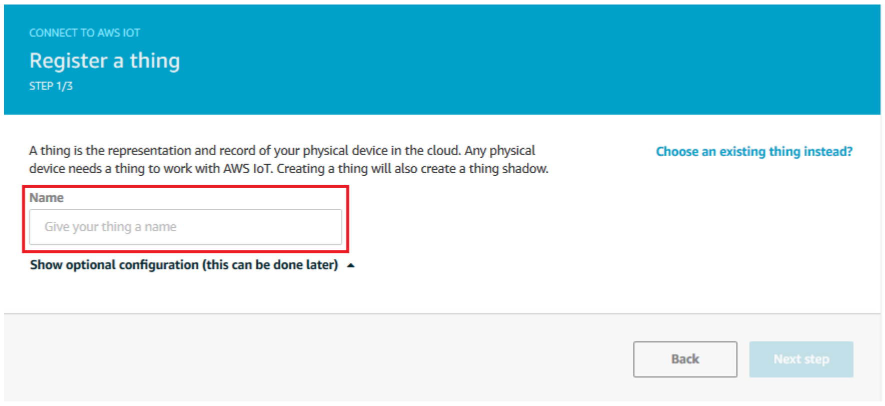
Choose a platform
Linux/OSX
Windows
Choose a AWS IoT Device SDK
Node.js
Python
Java

Some prerequisites to consider:
the device should have **Node.js and NPM installed** and a **TCP connection to the public internet on port 8883.**

Looking for AWS IoT Device SDKs and documentation?
View AWS IoT Device SDKs

2. In the **Name** field, enter the name for your thing object. The thing name used in this example is **`MyIotThing`**.

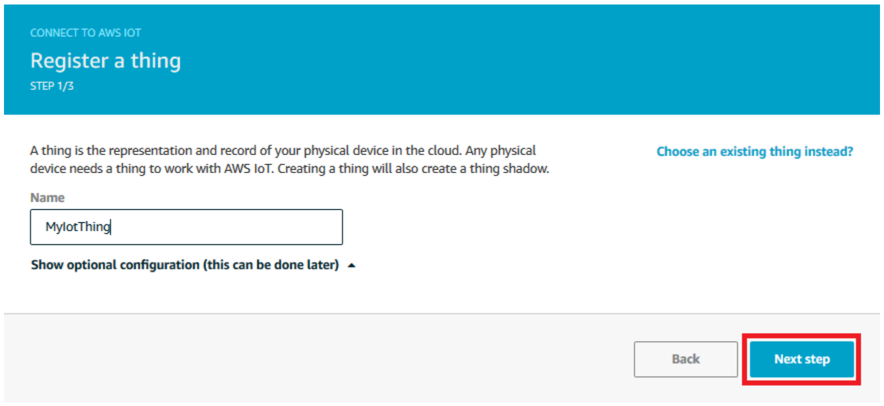
   Important
   Double-check your thing name before you continue.
   A thing name can't be changed after the thing object is created. If you want to change a thing name, you must create a new thing object with the correct thing name and then delete the one with the incorrect name.


CONNECT TO AWS IOT
Register a thing
STEP 1/3

A thing is the representation and record of your physical device in the cloud. Any physical device needs a thing to work with AWS IoT. Creating a thing will also create a thing shadow.

Choose an existing thing instead?
Name
Give your thing a name
Show optional configuration (this can be done later)
Back
Next step

3. After you give your thing object a name, choose **Next step**.



CONNECT TO AWS IOT

Register a thing

STEP 1/3

A thing is the representation and record of your physical device in the cloud. Any physical device needs a thing to work with AWS IoT. Creating a thing will also create a thing shadow.

Choose an existing thing instead?

Name: MyIotThing

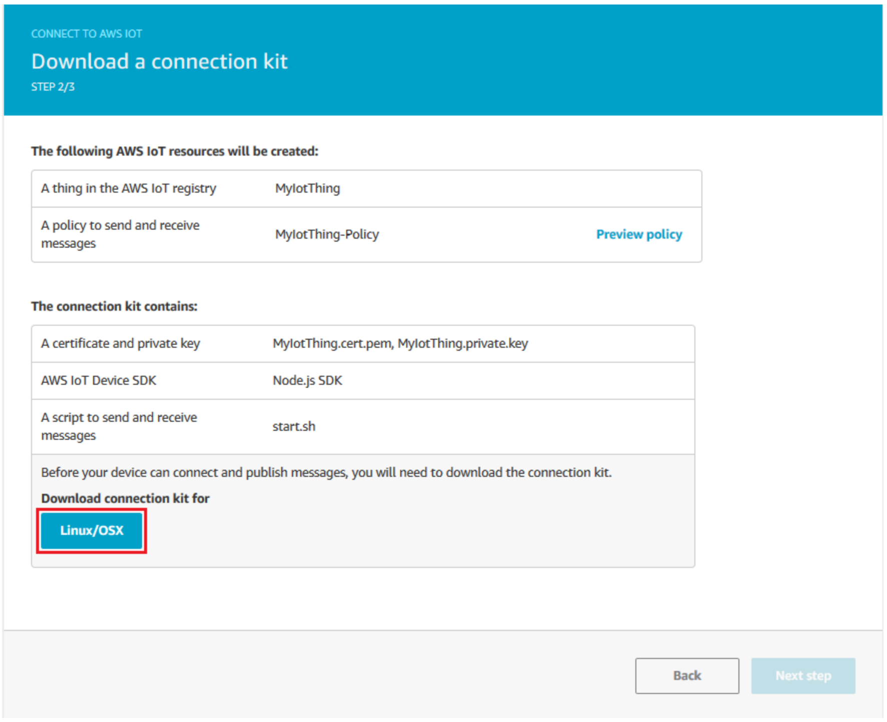
Show optional configuration (this can be done later)

Step 3. Download files to your device

This page appears after AWS IoT has created the connection kit, which includes the following files and resources that your device requires:

- The thing's certificate files used to authenticate the device
- A policy resource to authorize your thing object to interact with AWS IoT
- The script to download the AWS Device SDK and run the sample program on your device

1. When you're ready to continue, choose the **Download connection kit for** button to download the connection kit for the platform that you chose earlier.



CONNECT TO AWS IOT

Download a connection kit

STEP 2/3

The following AWS IoT resources will be created:

A thing in the AWS IoT registry	MyIotThing	
A policy to send and receive messages	MyIotThing-Policy	Preview policy

The connection kit contains:

A certificate and private key	MyIotThing.cert.pem, MyIotThing.private.key
AWS IoT Device SDK	Node.js SDK
A script to send and receive messages	start.sh

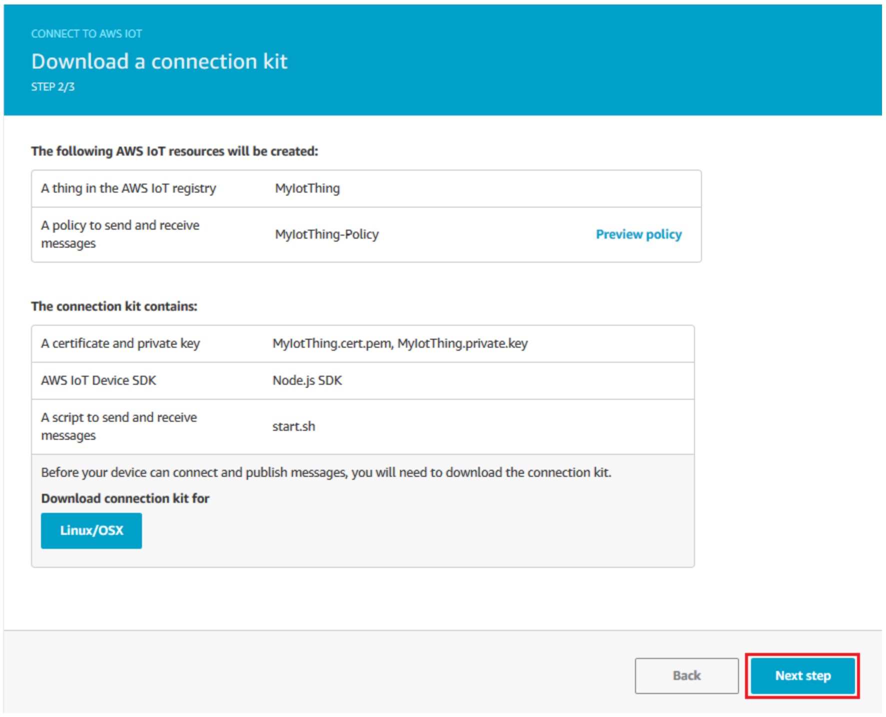
Before your device can connect and publish messages, you will need to download the connection kit.

Download connection kit for

2. If you're running this procedure on your device, save the connection kit file to a directory from which you can run command line commands.

If you're not running this procedure on your device, save the connection kit file to a local directory and then transfer the file to your device.

3. After you have the connection kit file on the device, continue the tutorial by choosing **Next step**.



CONNECT TO AWS IOT

Download a connection kit

STEP 2/3

The following AWS IoT resources will be created:

A thing in the AWS IoT registry	MyIotThing	
A policy to send and receive messages	MyIotThing-Policy	Preview policy

The connection kit contains:

A certificate and private key	MyIotThing.cert.pem, MyIotThing.private.key
AWS IoT Device SDK	Node.js SDK
A script to send and receive messages	start.sh

Before your device can connect and publish messages, you will need to download the connection kit.

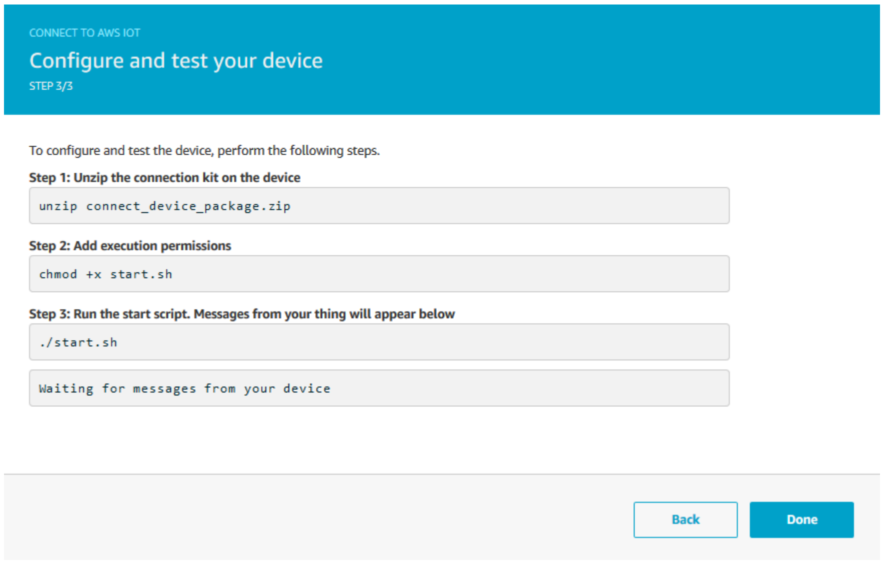
Download connection kit for

Step 4. Run the sample

This procedure is done in a terminal or command window on your device while following the directions displayed in the console. The commands shown in the console are those for the operating system you chose in the section called “Step 2. Create a thing object” (p. 27). Those shown here are for the Linux/OSX operating systems.

1. In a terminal or command window on your device, in the directory with the connection kit file, perform the steps shown in the AWS IoT console.

   If you're using a Windows PowerShell command window and the **unzip** command doesn't work, replace **unzip** with **expand-archive** and try the command line again.



To configure and test the device, perform the following steps.

Step 1: Unzip the connection kit on the device

```
unzip connect_device_package.zip
```

Step 2: Add execution permissions

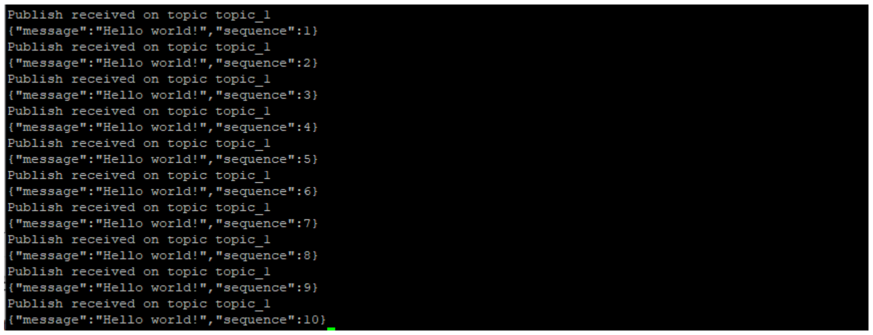
```
chmod +x start.sh
```

Step 3: Run the start script. Messages from your thing will appear below

```
./start.sh
```

```
Waiting for messages from your device
```

2. In the terminal or command window on your device, after you enter the command from **Step 3** in the console, you should see an output similar to this. This output is from the messages the program is sending to and then receiving back from AWS IoT Core. While the sample program is communicating with AWS IoT Core, you won't see any activity in the console. To see activity in the console while you run the sample program, see **Step 4** of this procedure.



```
Publish received on topic topic_1
{"message":"Hello world!","sequence":1}
Publish received on topic topic_1
{"message":"Hello world!","sequence":2}
Publish received on topic topic_1
{"message":"Hello world!","sequence":3}
Publish received on topic topic_1
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

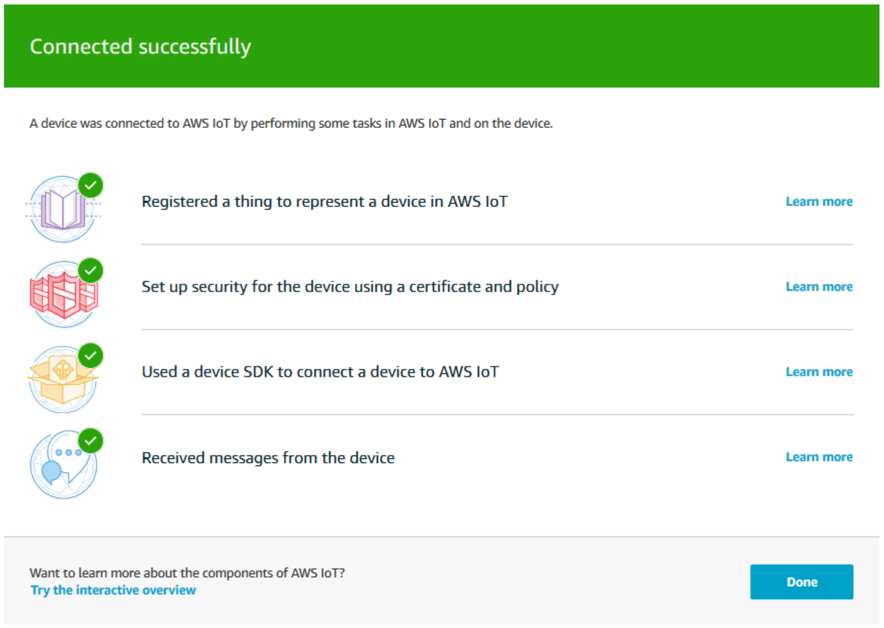
3. You can repeat the commands from **Step 3/3** in the console of this procedure), to run the sample program again.

4. (Optional) If you want to see the messages from your IoT client in the AWS IoT console, open the MQTT client on the **Test** page of the AWS IoT console. In the **MQTT client**, subscribe to **`sdk/test/SDK_programming_language`**. The topic name depends on the programming language of the SDK you chose in **Step 1/1**. The possible topic names are as follows.

   - For the AWS IoT Device SDK, the topic is **`sdk/test/javascript`**.
   - For the Python AWS IoT Device SDK, the topic is **`sdk/test/python`**.
   - For the Java AWS IoT Device SDK, the topic is **`sdk/test/java`**.

5. After you subscribe to the test topic, run this program on your device **./start.sh** as described in the previous step. For more information, see the section called “View MQTT messages with the AWS IoT MQTT client” (p. 60) for more information.

6. After you've finished running the sample app on your device, in the AWS IoT console, choose **Done** to finish the tutorial and see this summary.



Connected successfully

A device was connected to AWS IoT by performing some tasks in AWS IoT and on the device.

Registered a thing to represent a device in AWS IoT — Learn more

Set up security for the device using a certificate and policy — Learn more

Used a device SDK to connect a device to AWS IoT — Learn more

Received messages from the device — Learn more

Want to learn more about the components of AWS IoT?
Try the interactive overview

Step 5. Explore further

Here are some ideas to explore AWS IoT further after you complete the quick start.

- **View MQTT messages in the MQTT client**

  From the AWS IoT console, you can open the MQTT client on the **Test** page of the AWS IoT console. In the **MQTT client**, subscribe to **#**, and then, on your device, run the program **./start.sh** as described in the previous step. For more information, see the section called "View MQTT messages with the AWS IoT MQTT client" (p. 60).
- **the section called "Try the AWS IoT Core interactive demo" (p. 20)**

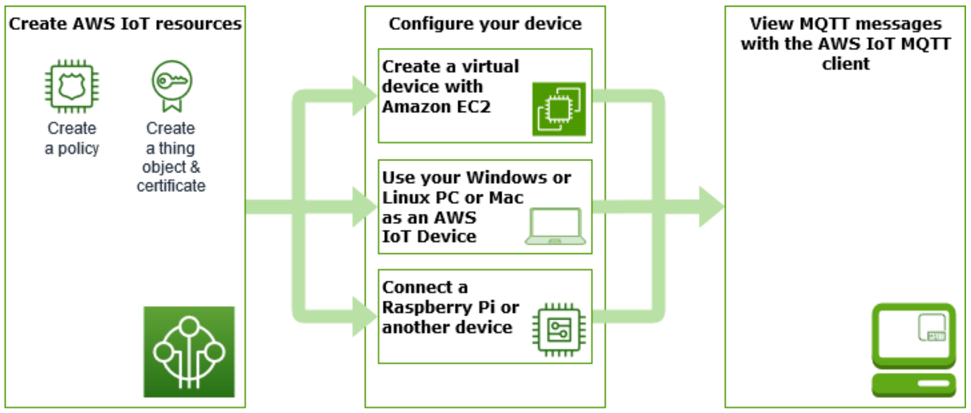
  To start the interactive tutorial, from the **Learn** page of the AWS IoT console, in the **See how AWS IoT works** tile, choose **Start the tutorial**.
- **Get ready to explore more tutorials (p. 32)**

  This quick start gives you just a sample of AWS IoT. If you want to explore AWS IoT further and learn about the features that make it a powerful IoT solution platform, start preparing your development platform by Explore AWS IoT Core services in hands-on tutorial (p. 32).

Explore AWS IoT Core services in hands-on tutorial

In this tutorial, you'll install the software and create the AWS IoT resources necessary to connect a device to AWS IoT so that it can send and receive MQTT messages with AWS IoT Core. You'll see the messages in the MQTT client in the AWS IoT console.

You can expect to spend 20-30 minutes on this tutorial. If you are using an IoT device or a Raspberry Pi, this tutorial might take longer if, for example, you need to install the operating system and configure the device.


Create AWS IoT resources
Create a policy
Create a thing object & certificate
Configure your device
Create a virtual device with Amazon EC2
Use your Windows or Linux PC or Mac as an AWS IoT Device
Connect a Raspberry Pi or another device
View MQTT messages with the AWS IoT MQTT client

This tutorial is best for developers who want to get started with AWS IoT so they can continue to explore more advanced features, such as the rules engine and shadows. This tutorial prepares you to continue learning about AWS IoT Core and how it interacts with other AWS services by explaining the steps in greater detail than the quick start tutorial. If you are looking for just a quick, *Hello World*, experience, try the Try the AWS IoT quick connect (p. 24).

After setting up your AWS account and AWS IoT console, you'll follow these steps to see how to connect a device and have it send messages to AWS IoT.

Next steps

- Choose which device option is the best for you (p. 33)
- the section called “Create AWS IoT resources” (p. 34) if you are not going to create a virtual device with Amazon EC2.
- the section called “Configure your device” (p. 38)
- the section called “View MQTT messages with the AWS IoT MQTT client” (p. 60)

For more information about AWS IoT Core, see What Is AWS IoT Core (p. 1)?

Which device option is the best for you?

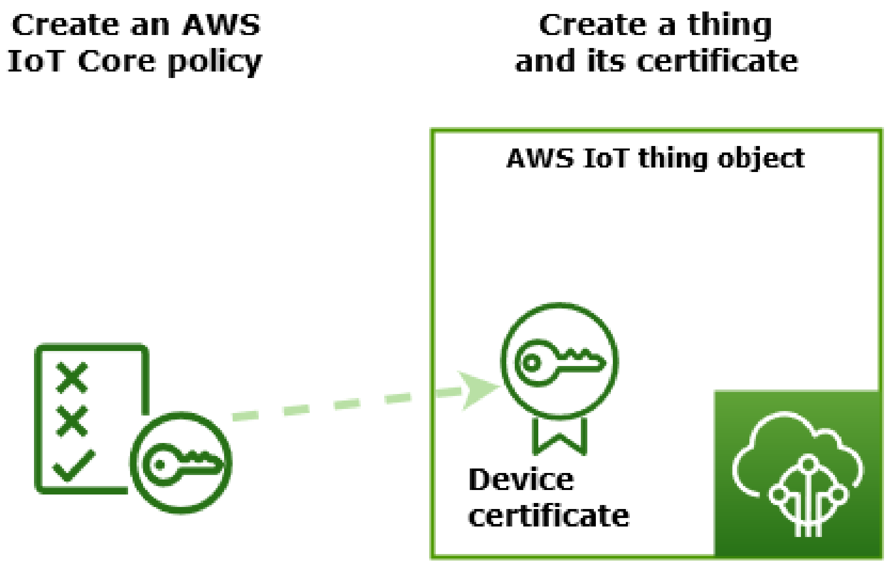
If you're not sure which option to pick, use the following list of each option's advantages and disadvantages to help you decide which one is best for you.

Option	This might be a good option if:	This might not be a good option if:
the section called “Create a virtual device with Amazon EC2” (p. 38)	• You don't have your own device to test. • You don't want to install any software on your own system. • You want to test on a Linux OS.	• You're not comfortable using command-line commands. • You don't want to incur any additional AWS charges. • You don't want to test on a Linux OS.
the section called “Use your Windows or Linux PC or Mac as an AWS IoT device” (p. 46)	• You don't want to incur any additional AWS charges. • You don't want to configure any additional devices.	• You don't want to install any software on your personal computer. • You want a more representative test platform.

Option	This might be a good option if:	This might not be a good option if:
the section called "Connect a Raspberry Pi or another device" (p. 51)	• You want to test AWS IoT with an actual device. • You already have a device to test with. • You have experience integrating hardware into systems.	• You don't want to buy or configure a device just to try it out. • You want to test AWS IoT as simply as possible, for now.

Create AWS IoT resources

In this tutorial, you'll create the AWS IoT resources that a device requires to connect to AWS IoT and exchange messages.


Create an AWS IoT Core policy
Create a thing and its certificate
AWS IoT thing object
Device certificate

1. Create an AWS IoT policy document, which will authorize your device to interact with AWS IoT services.
2. Create a thing object in AWS IoT and its X.509 device certificate, and then attach the policy document. The thing object is the virtual representation of your device in the AWS IoT registry. The certificate authenticates your device to AWS IoT Core, and the policy document authorizes your device to interact with AWS IoT.

    Note
    If you are planning to the section called "Create a virtual device with Amazon EC2" (p. 38), you can skip this page and continue to the section called "Configure your device" (p. 38). You will create these resources when you create your virtual thing.

This tutorial uses the AWS IoT console to create the AWS IoT resources. If your device supports a web browser, it might be easier to run this procedure on the device's web browser because you will be able to download the certificate files directly to your device. If you run this procedure on another computer, you will need to copy the certificate files to your device before they can be used by the sample app.

Create an AWS IoT policy

X.509 certificates are used to authenticate your device with AWS IoT Core. AWS IoT policies are attached to the certificate that authenticates the device to determine the AWS IoT operations, such as subscribing or publishing to MQTT topics that the device is permitted to perform. Your device presents its certificate when it connects and sends messages to AWS IoT Core.

In this procedure, you will create a policy that allows your device to perform the AWS IoT operations necessary to run the example program. You must create the AWS IoT policy first, so that you can attach it to the device certificate that you will create later.

To create an AWS IoT policy

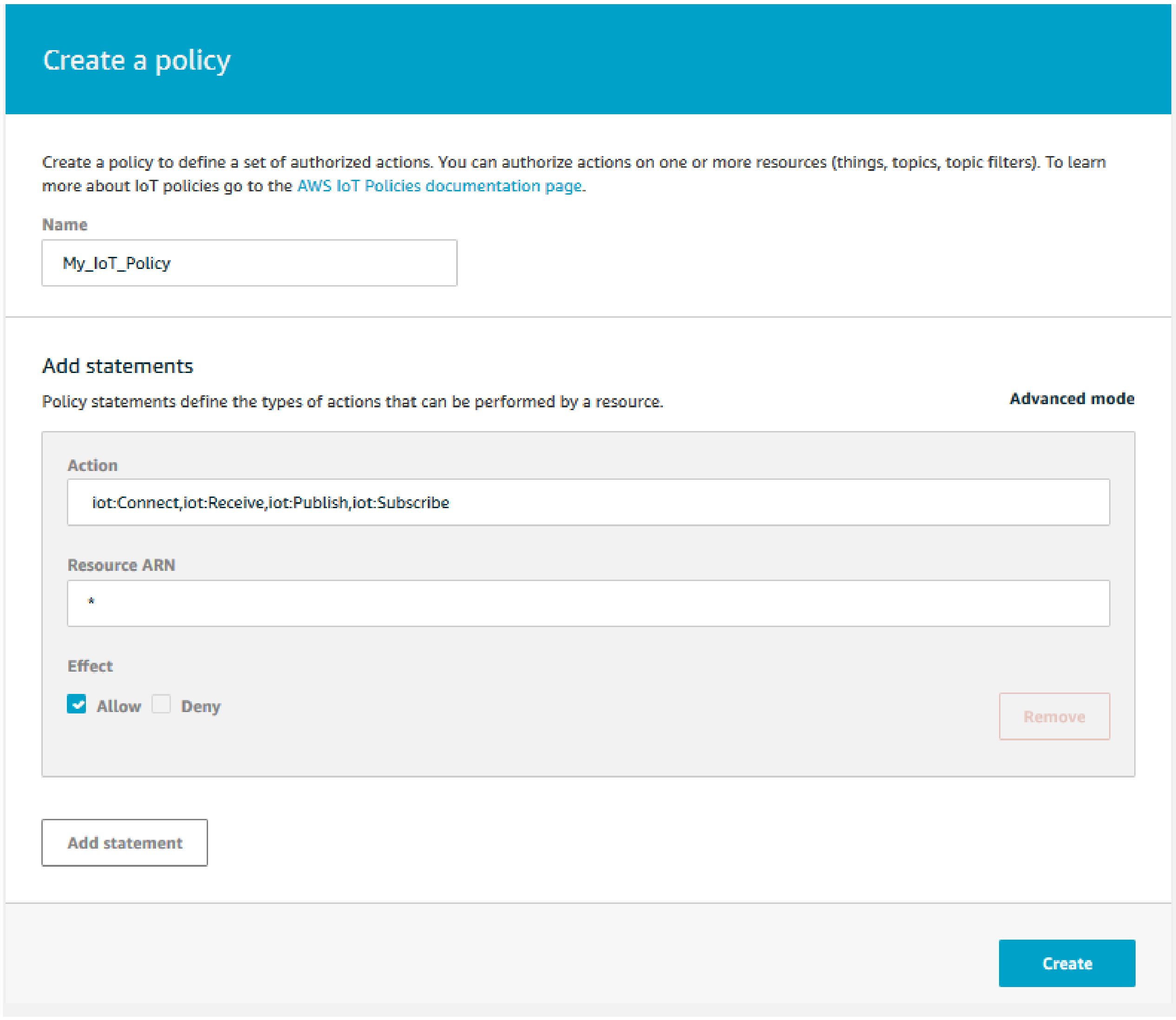
1. In the left menu, choose **Secure**, and then choose **Policies**. On the **You don't have a policy yet** page, choose **Create a policy**.

   If your account has existing policies, choose **Create**.
2. On the **Create a policy** page:
   1. In the **Name** field, enter a name for the policy (for example, **`My_Iot_Policy`**). Do not use personally identifiable information in your policy names.
   2. In the **Action** field, enter **`iot:Connect,iot:Receive,iot:Publish,iot:Subscribe`**. These are the actions that the device will need permission to perform when it runs the example program from the Device SDK.

      For more information about IoT policies, see AWS IoT Core policies (p. 235).
   3. In the **Resource ARN** field, enter **`*`**. This selects any client (device).

      Note
      In this quick start, the wildcard (*) character is used for simplicity. For higher security, you should restrict which clients (devices) can connect and publish messages by specifying a client ARN (Amazon resource name) instead of the wildcard character as the resource. Client ARNs follow this format:
      `arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`*
      However, you must first create the resource (client device, thing shadow, etc.) before you can assign its ARN to a policy.
   4. Choose the **Allow** check box.

      These values allows all clients that have this policy attached to their certificate to perform the actions listed in the **Action** field.


Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the AWS IoT Policies documentation page.

Name: My_IoT_Policy

Add statements

Policy statements define the types of actions that can be performed by a resource.

Action: iot:Connect,iot:Receive,iot:Publish,iot:Subscribe

Resource ARN: *

Effect: [✓ Allow] Deny

3. After you have entered the information for your policy, choose **Create**.

For more information, see IAM policies (p. 282).

Create a thing object

Devices connected to AWS IoT are represented by *thing objects* in the AWS IoT registry. A *thing object* represents a specific device or logical entity. It can be a physical device or sensor (for example, a light bulb or a light switch on the wall). It can also be a logical entity, like an instance of an application or physical entity that does not connect to AWS IoT, but is related to other devices that do (for example, a car that has engine sensors or a control panel).

To create a thing in the AWS IoT console

1. In the AWS IoT console, on the **Welcome to the AWS IoT console** page, in the left menu, choose **Manage**.
2. On **You don't have any things yet**, choose **Register a thing**.

If your account already has some things, choose **Create**.

3. On **Creating AWS IoT things**, choose **Create a single thing**.
4. On the **Add your device to the thing registry** page, in the **Name** field, enter a name for your thing, such as **`MyIotThing`**.

   When naming things, choose the name carefully, because you can't change a thing name after you create it.

   To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

   Note
   Do not use personally identifiable information in your thing name. The thing name can appear in unencrypted communications and reports.

   Leave the rest of the fields on this page empty, for now.

   Choose **Next**.
5. On **Add a certificate for your thing**, choose **Create certificate**.
6. On the **Certificate created!** page:

   1. Download each of the certificate and key files and save them for later. You will need to install these files on your device.

      When you save your certificate files, give them the names in the following table. These are the file names used in later examples.

      Certificate file names

File	File path
Private key	`private.pem.key`
Public key	*(not used in these examples)*
Device certificate	`device.pem.crt`
Root CA certificate	`Amazon-root-CA-1.pem`

   2. Download the root CA file for these files by choosing the **A root CA for AWS IoT Download** link and downloading the **RSA 2048 bit key: Amazon Root CA 1** certificate file.

      Important
      You must save the certificate files before you leave this page. After you leave this page in the console, you will no longer have access to the certificate files.
      If you forgot to download the certificate files that you created in this step, you must exit this console screen, go to the list of things in the console, delete the thing object you created, and then restart this procedure from the beginning.
   3. Choose **Activate**, to enable the certificate for connections to AWS IoT.
   4. Choose **Attach a policy**, to attach the policy you created in the previous section to this certificate.
   5. On **Add a policy for your thing**, select the policy you created in the previous section. In that section, the policy was named, **`My_Iot_Policy`**.
   6. Choose **Register Thing**.

After you complete this procedure, you should see the new thing object in your list of things.

Configure your device

This section describes how to configure your device to connect to AWS IoT. If you'd like to get started with AWS IoT but don't have a device yet, you can create a virtual device by using Amazon EC2 or you can use your Windows PC or Mac as an IoT device.

Select the best device option for you to try AWS IoT. Of course, you can try all of them, but try only one at a time. If you're not sure which device option is best for you, read about how to choose which device option is the best (p. 33), and then return to this page.

Device options

- Create a virtual device with Amazon EC2 (p. 38)
- Use your Windows or Linux PC or Mac as an AWS IoT device (p. 46)
- Connect a Raspberry Pi or another device (p. 51)

Create a virtual device with Amazon EC2

In this tutorial, you'll create an Amazon EC2 instance to serve as your virtual device in the cloud.

To complete this tutorial, you need an AWS account. If you don't have one, complete the steps described in Set up your AWS account (p. 18) before you continue.

In this tutorial, you'll:

- Set up an Amazon EC2 instance (p. 38)
- Install Git, Node.js and configure the AWS CLI (p. 39)
- Create AWS IoT resources for your virtual device (p. 41)
- Install the AWS IoT Device SDK for JavaScript (p. 44)
- Run the sample application (p. 44)
- View messages from the sample app in the AWS IoT console (p. 45)

Set up an Amazon EC2 instance

The following steps show you how to create an Amazon EC2 instance that will act as your virtual device in place of a physical device.

To launch an instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the console dashboard, choose **Launch Instance**.
3. The **Step 1: Choose an Amazon Machine Image (AMI)** page displays a list of basic configurations, called *Amazon Machine Images (AMIs)*, which serve as templates for your instance. Select an HVM version of Amazon Linux 2, such as *Amazon Linux 2 AMI (HVM), SSD Volume Type*. Notice that this AMI is marked "Free tier eligible."
4. On the **Choose an Instance Type** page, you can select the hardware configuration of your instance. Select the `t2.micro` type, which is selected by default. Notice that this instance type is eligible for the free tier.
5. Choose **Review and Launch** to let the wizard complete the other configuration settings for you.
6. On the **Review Instance Launch** page, choose **Launch**.
7. When prompted for a key pair, select **Create a new key pair**, enter a name for the key pair, and then choose **Download Key Pair**. *This is your only chance to save the private key file, so be sure to download it.* Save the private key file in a safe place. You'll need to provide the name of your key

pair when you launch an instance and the corresponding private key each time you connect to the instance.

> **Warning**
> Don't select the **Proceed without a key pair** option. If you launch your instance without a key pair, then you can't connect to it.

When you are ready, choose **Launch Instances**.

8. A confirmation page lets you know that your instance is launching. Choose **View Instances** to close the confirmation page and return to the console.
9. On the **Instances** screen, you can view the status of the launch. It takes a short time for an instance to launch. When you launch an instance, its initial state is `pending`. After the instance starts, its state changes to `running` and it receives a public DNS name. (If the **Public DNS (IPv4)** column is hidden, choose **Show/Hide Columns** (the gear-shaped icon) in the top right corner of the page and then select **Public DNS (IPv4)**.)
10. It can take a few minutes for the instance to be ready so that you can connect to it. Check that your instance has passed its status checks; you can view this information in the **Status Checks** column.

    After your new instance has passed its status checks, continue to the next procedure and connect to it.

To connect to your instance

You can connect to an instance using the browser-based client by selecting the instance from the Amazon EC2 console and choosing to connect using Amazon EC2 Instance Connect. Instance Connect handles the permissions and provides a successful connection.

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the left menu, choose **Instances**.
3. Select the instance and choose **Connect**.
4. Choose **Amazon EC2 Instance Connect (browser-based SSH connection)**, **Connect**.

You should now have an **Amazon EC2 Instance Connect** window that is logged into your new Amazon EC2 instance.

Install Git, Node.js and configure the AWS CLI

In this section, you'll install Git and Node.js, on your Linux instance.

To install Git

1. In your **Amazon EC2 Instance Connect** window, update your instance by using the following command.

```
sudo yum update -y
```

2. In your **Amazon EC2 Instance Connect** window, install Git by using the following command.

```
sudo yum install git -y
```

To install Node.js

1. In your **Amazon EC2 Instance Connect** window, install node version manager (nvm) by using the following command.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```

We will use nvm to install Node.js because nvm can install multiple versions of Node.js and allow you to switch between them.

2. In your **Amazon EC2 Instance Connect** window, activate nvm by using this command.

```
. ~/.nvm/nvm.sh
```

3. In your **Amazon EC2 Instance Connect** window, use nvm to install the latest version of Node.js by using this command.

```
nvm install node
```

Installing Node.js also installs the Node Package Manager (npm) so you can install additional modules as needed.

4. In your **Amazon EC2 Instance Connect** window, test that Node.js is installed and running correctly by using this command.

```
node -v
```

This tutorial requires Node v10.0 or later.

To configure AWS CLI

Your Amazon EC2 instance comes preloaded with the AWS CLI. However, you must complete your AWS CLI profile. For more information on how to configure your CLI, see Configuring the AWS CLI.

1. The following example shows sample values. Replace them with your own values. You can find these values in your AWS console in your account info under **My Security Credentials**.

   In your **Amazon EC2 Instance Connect** window, enter this command:

```
aws configure
```

Then enter the values from your account at the prompts displayed.

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

2. You can test your AWS CLI configuration with this command:

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

If your AWS CLI is configured correctly, the command should return an endpoint address from your AWS account.

Create AWS IoT resources for your virtual device

This section describes how to use the AWS CLI to create the thing object and its certificate files directly on the virtual device. This is done directly on the device to avoid the potential complication that might arise from copying them to the device from another computer.

To create an AWS IoT thing object in your Linux instance

Devices connected to AWS IoT are represented by *thing objects* in the AWS IoT registry. A *thing object* represents a specific device or logical entity. In this case, your *thing object* will represent your virtual device, this Amazon EC2 instance.

1. In your **Amazon EC2 Instance Connect** window, run the following command to create your thing object.

```
aws iot create-thing --thing-name "MyIotThing"
```

2. The JSON response should look like this:

```
{
    "thingArn": "arn:aws:iot:your-region:your-aws-account:thing/MyIotThing",
    "thingName": "MyIotThing",
    "thingId": "6cf922a8-d8ea-4136-f3401EXAMPLE"
}
```

To create and attach AWS IoT keys and certificates in your Linux instance

The **create-keys-and-certificate** command creates client certificates signed by the Amazon Root certificate authority. This certificate is used to authenticate the identity of your virtual device.

1. In your **Amazon EC2 Instance Connect** window, create a directory to store your certificate and key files.

```
mkdir ~/certs
```

2. In your **Amazon EC2 Instance Connect** window, download a copy of the Amazon certificate authority (CA) certificate by using this command.

```
curl -o ~/certs/Amazon-root-CA-1.pem \
    https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

3. In your **Amazon EC2 Instance Connect** window, run the following command to create your private key, public key, and X.509 certificate files. This command also registers and activates the certificate with AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "~/certs/device.pem.crt" \
    --public-key-outfile "~/certs/public.pem.key" \
    --private-key-outfile "~/certs/private.pem.key"
```

The response looks like the following. Save the `certificateArn` so that you can use it in subsequent commands. You'll need it to attach your certificate to your thing and to attach the policy to the certificate in a later steps.

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC
 KEY-----\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h
\nMMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/
gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWEXAMPLE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\nhJLXkX3rHU2xbxJSq7D
+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB\n-----END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omitted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

4. In your **Amazon EC2 Instance Connect** window, attach your thing object to the certificate you just created by using the following command and the `certificateArn` in the response from the previous command.

```
aws iot attach-thing-principal \
    --thing-name "MyIotThing" \
    --principal "certificateArn"
```

   If successful, this command does not display any output.

To create and attach a policy

1. In your **Amazon EC2 Instance Connect** window, create the policy file by copying and pasting this policy document to a file named **`~/policy.json`**.

   If you don't have a favorite Linux editor, you can open **nano**, by using this command.

```
nano ~/policy.json
```

   And pasting the policy document for `policy.json` into it. Enter ctrl-x to exit the **nano** editor and save the file.

   Contents of the policy document for `policy.json`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

2. In your **Amazon EC2 Instance Connect** window, create your policy by using the following command.

   ```
   aws iot create-policy \
       --policy-name "MyIotThingPolicy" \
       --policy-document "file://~/policy.json"
   ```

   Output:

   ```
   {
       "policyName": "MyIotThingPolicy",
       "policyArn": "arn:aws:iot:your-region:your-aws-account:policy/MyIotThingPolicy",
       "policyDocument": "{
           \"Version\": \"2012-10-17\",
           \"Statement\": [
               {
                   \"Effect\": \"Allow\",
                   \"Action\": [
                       \"iot:Publish\",
                       \"iot:Receive\",
                       \"iot:Subscribe\",
                       \"iot:Connect\"
                   ],
                   \"Resource\": [
                       \"*\"
                   ]
               }
           ]
       }",
       "policyVersionId": "1"
   }
   ```

3. In your **Amazon EC2 Instance Connect** window, attach the policy to your virtual device's certificate by using the following command.

   ```
   aws iot attach-policy \
       --policy-name "MyIotThingPolicy" \
       --target "certificateArn"
   ```

   If successful, this command does not display any output.

At this point, you have created for your virtual device:

- A thing object to represent your virtual device in AWS IoT.
- A certificate to authenticate your virtual device.

- A policy document to authorize your virtual device to Connect to AWS IoT, and to Publish, Receive, and Subscribe to messages.

Install the AWS IoT Device SDK for JavaScript

In this section, you'll install the AWS IoT Device SDK for JavaScript, which contains the code that applications can use to communicate with AWS IoT and the sample programs.

To install the AWS IoT Device SDK for JavaScript on your Linux instance

1. In your **Amazon EC2 Instance Connect** window, clone the AWS IoT Device SDK for JavaScript repository into the `aws-iot-device-sdk-js-v2` directory of your home directory by using this command.

   ```
   cd ~
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```
2. Navigate to the `aws-iot-device-sdk-js-v2` directory that you created in the preceding step.

   ```
   cd aws-iot-device-sdk-js-v2
   ```
3. Use npm to install the SDK.

   ```
   npm install
   ```

Run the sample application

The commands in the next sections assume that your key and certificate files are stored on your virtual device as shown in this table.

Certificate file names

File	File path
Private key	`~/certs/private.pem.key`
Device certificate	`~/certs/device.pem.crt`
Root CA certificate	`~/certs/Amazon-root-CA-1.pem`

In this section, you'll install and run the `pub-sub.js` sample app found in the `aws-iot-device-sdk-js-v2/samples/node` directory of the AWS IoT Device SDK for JavaScript. This app shows how a device, your Amazon EC2 instance, uses the MQTT library to publish and subscribe to MQTT messages. The `pub-sub.js` sample app subscribes to a topic, `topic_1`, publishes 10 messages to that topic, and displays the messages as they're received from the message broker.

To install and run the sample app

1. In your **Amazon EC2 Instance Connect** window, navigate to the `aws-iot-device-sdk-js-v2/samples/node/pub_sub` directory that the SDK created and install the sample app by using these commands.

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. In your **Amazon EC2 Instance Connect** window, get *your-iot-endpoint* from AWS IoT by using this command.

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

3. In your **Amazon EC2 Instance Connect** window, insert *your-iot-endpoint* as indicated and run this command.

   ```
   node dist/index.js --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

The sample app:

1. Connects to the AWS IoT service for your account.
2. Subscribes to the message topic, **topic_1**, and displays the messages it receives on that topic.
3. Publishes 10 messages to the topic, **topic_1**.
4. Displays output similar to the following:

```
Publish received on topic topic_1
{"message":"Hello world!","sequence":1}
Publish received on topic topic_1
{"message":"Hello world!","sequence":2}
Publish received on topic topic_1
{"message":"Hello world!","sequence":3}
Publish received on topic topic_1
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

If you're having trouble running the sample app, review the section called “Troubleshooting problems with the sample app” (p. 59).

You can also add the `--verbosity debug` parameter to the command line so the sample app displays detailed messages about what it’s doing. That information might provide you the help you need to correct the problem.

View messages from the sample app in the AWS IoT console

You can see the sample app's messages as they pass through the message broker by using the **MQTT client** in the **AWS IoT console**.

To view the MQTT messages published by the sample app

1. Review View MQTT messages with the AWS IoT MQTT client (p. 60). This helps you learn how to use the **MQTT client** in the **AWS IoT console** to view MQTT messages as they pass through the message broker.
2. Open the **MQTT client** in the **AWS IoT console**.

3. Subscribe to the topic, **topic_1**.
4. In your **Amazon EC2 Instance Connect** window, run the sample app again and watch the messages in the **MQTT client** in the **AWS IoT console**.

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
node dist/index.js --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

Use your Windows or Linux PC or Mac as an AWS IoT device

In this tutorial, you'll configure a personal computer for use with AWS IoT. These instructions support Windows and Linux PCs and Macs. To accomplish this, you need to install some software on your computer. If you don't want to install software on your computer, you might try Create a virtual device with Amazon EC2 (p. 38), which installs all software on a virtual machine.

In this tutorial, you'll:

- Set up your personal computer (p. 46)
- Install Git, Python, and the AWS IoT Device SDK for Python (p. 46)
- Set up the policy and run the sample application (p. 49)
- View messages from the sample app in the AWS IoT console (p. 51)

Set up your personal computer

To complete this tutorial, you need a Windows or Linux PC or a Mac with a connection to the internet.

Before you continue to the next step, make sure you can open a command line window on your computer. Use **cmd.exe** on a Windows PC. On a Linux PC or a Mac, use **Terminal**.

Install Git, Python, and the AWS IoT Device SDK for Python

In this section, you'll install Python, and the AWS IoT Device SDK for Python on your computer.

Install the latest version of Git and Python

To download and install Git and Python on your computer

1. Check to see if you have Git installed on your computer. Enter this command in the command line.

   ```
   git --version
   ```

   If the command displays the Git version, Git is installed and you can continue to the next step.

   If the command displays an error, open https://git-scm.com/download and install Git for your computer.
2. Check to see if you have already installed Python. Enter this command in the command line.

   ```
   python -V
   ```

   If the command displays the Python version, Python is already installed. This tutorial requires Python v3.5 or later.
3. If Python is installed, you can skip the rest of the steps in this section. If not, continue.
4. Open https://www.python.org/downloads/ and download the installer for your computer.
5. If the download didn't automatically start to install, run the downloaded program to install Python.

6. Verify the installation of Python.

   ```
   python -V
   ```

   Confirm that the command displays the Python version. If the Python version isn't displayed, try downloading and installing Python again.

Install the AWS IoT Device SDK for Python

To install the AWS IoT Device SDK for Python on your computer

1. Install v2 of the AWS IoT Device SDK for Python.

   ```
   python3 -m pip install awsiotsdk
   ```

2. Clone the AWS IoT Device SDK for Python repository into the aws-iot-device-sdk-python-v2 directory of your home directory. This procedure refers to the base directory for the files you're installing as *home*.

   The actual location of the *home* directory depends on your operating system.

   Linux/macOS

   In macOS and Linux, the *home* directory is ~.

   ```
   cd ~ git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   Windows

   In Windows, you can find the *home* directory path by running this command in the `cmd` window.

   ```
   echo %USERPROFILE%
   cd %USERPROFILE%
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   Note
   If you're using Windows PowerShell as opposed to **cmd.exe**, then use the following command.

   ```

   echo $home
   ```

Prepare to run the sample applications

To prepare your system to run the sample application

- Create the `certs` directory. Into the `certs` directory, copy the private key, device certificate, and root CA certificate files you saved when you created and registered the thing object in the section called "Create AWS IoT resources" (p. 34). The file names of each file in the destination directory should match those in the table.

  The commands in the next section assume that your key and certificate files are stored on your device as shown in this table.

Linux/macOS

Run this command to create the `certs` subdirectory that you'll use when you run the sample applications.

```
mkdir ~/certs
```

Into the new subdirectory, copy the files to the destination file paths shown in the following table.

Certificate file names

File	File path
Private key	`~/certs/private.pem.key`
Device certificate	`~/certs/device.pem.crt`
Root CA certificate	`~/certs/Amazon-root-CA-1.pem`

Run this command to list the files in the `certs` directory and compare them to those listed in the table.

```
ls -l ~/certs
```

Windows

Run this command to create the `certs` subdirectory that you'll use when you run the sample applications.

```
mkdir %USERPROFILE%\certs
```

Into the new subdirectory, copy the files to the destination file paths shown in the following table.

Certificate file names

File	File path
Private key	`%USERPROFILE%\certs\private.pem.key`
Device certificate	`%USERPROFILE%\certs\device.pem.crt`
Root CA certificate	`%USERPROFILE%\certs\Amazon-root-CA-1.pem`

Run this command to list the files in the `certs` directory and compare them to those listed in the table.

```
dir %USERPROFILE%\certs
```

Set up the policy and run the sample application

In this section, you'll set up your policy and run the `pubsub.py` sample script found in the `aws-iot-device-sdk-python-v2/samples` directory of the AWS IoT Device SDK for Python. This script shows how your device uses the MQTT library to publish and subscribe to MQTT messages.

The `pubsub.py` sample app subscribes to a topic, `test/topic`, publishes 10 messages to that topic, and displays the messages as they're received from the message broker.

To run the `pubsub.py` sample script, you need the following information:

Application parameter values

Parameter	Where to find the value
`your-iot-endpoint`	1. In the AWS IoT console, choose **Manage**, and then choose **Things**. 2. Choose the IoT thing you created for your device (**MyIotThing** was the name used earlier), and then choose **Interact**. 3. On the thing details page, your endpoint is displayed in the **HTTPS** section.

The *`your-iot-endpoint`* value has a format of: *`endpoint_id`*`-ats.iot.`*`region`*`.amazonaws.com`, for example, `a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`.

Before running the script, make sure your thing's policy provides permissions for the sample script to connect, subscribe, publish, and receive. The Policy JSON is displayed in the following example. In the `"Resource"` element, you should replace the region and account word with your AWS account and Region `"arn:aws:iot:`*`region`*`:`*`account`*`.:topic/test/topic"`

```
{
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iot:Publish",
            "iot:Receive"
        ],
        "Resource": [
            "arn:aws:iot:region:account:topic/test/topic"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Subscribe"
        ],
        "Resource": [
            "arn:aws:iot:region:account:topicfilter/test/topic"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Connect"
        ],
        "Resource": [
            "arn:aws:iot:region:account:client/test-*"
        ]
    }
]
}
```

Linux/macOS

To run the sample script on Linux/macOS

1. In your command line window, navigate to the `~/aws-iot-device-sdk-python-v2/samples/node/pub_sub` directory that the SDK created by using these commands.

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   ```

2. In your command line window, replace *your-iot-endpoint* as indicated and run this command.

   ```
   python pubsub.py --endpoint your-iot-endpoint --root-ca %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs\private.pem.key
   ```

Windows

To run the sample app on a Windows PC

1. In your command line window, navigate to the `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples` directory that the SDK created and install the sample app by using these commands.

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   ```

2. In your command line window, replace *your-iot-endpoint* as indicated and run this command.

   ```
   python pubsub.py --endpoint your-iot-endpoint --root-ca %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs\private.pem.key
   ```

The sample script:

1. Connects to the AWS IoT service for your account.
2. Subscribes to the message topic, **test/topic**, and displays the messages it receives on that topic.
3. Publishes 10 messages to the topic, **test/topic**.
4. Displays output similar to the following:

```
Publish received on topic test/topic
{"message":"Hello world!","sequence":1}
Publish received on topic test/topic
{"message":"Hello world!","sequence":2}
Publish received on topic test/topic
{"message":"Hello world!","sequence":3}
Publish received on topic test/topic
{"message":"Hello world!","sequence":4}
Publish received on topic test/topic
```

```
{"message":"Hello world!","sequence":5}
Publish received on topic test/topic
{"message":"Hello world!","sequence":6}
Publish received on topic test/topic
{"message":"Hello world!","sequence":7}
Publish received on topic test/topic
{"message":"Hello world!","sequence":8}
Publish received on topic test/topic
{"message":"Hello world!","sequence":9}
Publish received on topic test/topic
{"message":"Hello world!","sequence":10}
```

If you're having trouble running the sample app, review the section called "Troubleshooting problems with the sample app" (p. 59).

You can also add the `--verbosity debug` parameter to the command line so the sample app displays detailed messages about what it's doing. That information might help you correct the problem.

View messages from the sample app in the AWS IoT console

You can see the sample app's messages as they pass through the message broker by using the **MQTT client** in the **AWS IoT console**.

To view the MQTT messages published by the sample app

1. Review View MQTT messages with the AWS IoT MQTT client (p. 60). This helps you learn how to use the **MQTT client** in the **AWS IoT console** to view MQTT messages as they pass through the message broker.
2. Open the **MQTT client** in the **AWS IoT console**.
3. Subscribe to the topic, **test/topic**.
4. In your command line window, run the sample app again and watch the messages in the **MQTT client** in the **AWS IoT console**.

   Linux/macOS

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic test/topic --root-ca ~/certs/Amazon-root-CA-1.pem --cert
    ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

   Windows

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples\node\pub_sub
   python3 pubsub.py --topic test/topic --root-ca %USERPROFILE%\certs\Amazon-root-
   CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs
   \private.pem.key --endpoint your-iot-endpoint
   ```

Connect a Raspberry Pi or another device

In this section, we'll configure a Raspberry Pi for use with AWS IoT. If you have another device that you'd like to connect, the instructions for the Raspberry Pi include references that can help you adapt these instructions to your device.

This normally takes about 20 minutes, but it can take longer if you have many system software upgrades to install.

In this tutorial, you'll:

- Set up your device (p. 52)

- Install the required tools and libraries for the AWS IoT Device SDK (p. 52)
- Install AWS IoT Device SDK (p. 53)
- Install and run the sample app (p. 56)
- View messages from the sample app in the AWS IoT console (p. 58)

Important
Adapting these instructions to other devices and operating systems can be challenging. You'll need to understand your device well enough to be able to interpret these instructions and apply them to your device.
If you encounter difficulties while configuring your device for AWS IoT, we can't offer any assistance beyond the instructions in this section. However, you might try one of the other device options as an alternative, such as Create a virtual device with Amazon EC2 (p. 38) or Use your Windows or Linux PC or Mac as an AWS IoT device (p. 46).

Set up your device

The goal of this step is to collect what you'll need to configure your device such that it can start the operating system (OS), connect to the Internet, and allow you to interact with it at a command line interface.

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, complete the steps described in Set up your AWS account (p. 18) before you continue.
- A Raspberry Pi 3 Model B or more recent model. This might work on earlier versions of the Raspberry Pi, but they have not been tested.
- Raspberry Pi OS (32-bit) or later. We recommend using the latest version of the Raspberry Pi OS. Earlier versions of the OS might work, but they have not been tested.

  To run this example, you do not need to install the desktop with the graphical user interface (GUI); however, if you're new to the Raspberry Pi and your Raspberry Pi hardware supports it, using the desktop with the GUI might be easier.
- An Ethernet or Wi-Fi connection.
- Keyboard, mouse, monitor, cables, power supplies, and other hardware required by your device.

Important
Before you continue to the next step, your device must have its operating system installed, configured, and running. The device must be connected to the Internet and you will need to be able to access the device by using its command line interface. Command line access can be through a directly-connected keyboard, mouse, and monitor, or by using an SSH terminal remote interface.

If you are running an operating system on your Raspberry Pi that has a graphical user interface (GUI), open a terminal window on the device and perform the following instructions in that window. Otherwise, if you are connecting to your device by using a remote terminal, such as PuTTY, open a remote terminal to your device and use that.

Install the required tools and libraries for the AWS IoT Device SDK

Before you install the AWS IoT Device SDK and sample code, make sure your system is up-to-date and has the required tools and libraries to install the SDKs.

1. **Update the operating system and install required libraries**

   Before you install an AWS IoT Device SDK, run these commands in a terminal window on your device to update the operating system and install the required libraries.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install cmake
```

```
sudo apt-get install libssl-dev
```

2. **Install Git**

   If your device's operating system doesn't come with Git installed, you'll need to install it to install the AWS IoT Device SDK for JavaScript.

   a. Test to see if Git is already installed by running this command.

   ```
   git --version
   ```

   b. If the previous command returns the Git version, Git is already installed and you can skip to Step 3.

   c. If an error is displayed when you run the **git** command, install Git by running this command.

   ```
   sudo apt-get install git
   ```

   d. Test again to see if Git is installed by running this command.

   ```
   git --version
   ```

   e. If Git is installed, continue to the next section. If not, troubleshoot and correct the error before continuing. You need Git to install the AWS IoT Device SDK for JavaScript.

Install AWS IoT Device SDK

Install the AWS IoT Device SDK.

Python

In this section, you'll install Python, its development tools, and the AWS IoT Device SDK for Python on your device. These instructions are for a Raspberry Pi running the latest Raspberry Pi OS. If you have another device or are using another operating system, you might need to adapt these instructions for your device.

1. **Install Python and its development tools**

   The AWS IoT Device SDK for Python requires Python v3.5 or later to be installed on your Raspberry Pi.

   In a terminal window to your device, run these commands.

   1. Run this command to determine the version of Python installed on your device.

      ```
      python3 --version
      ```

      If Python is installed, it will display its version.

2. If the version displayed is `Python 3.5` or greater, you can skip to Step 2.
3. If the version displayed is less than `Python 3.5`, you can install the correct version by running this command.

```
sudo apt install python3
```

4. Run this command to confirm that the correct version of Python is now installed.

```
python3 --version
```

2. **Test for pip3**

   In a terminal window to your device, run these commands.

   1. Run this command to see if **pip3** is installed.

   ```
   pip3 --version
   ```

   2. If the command returns a version number, **pip3** is installed and you can skip to Step 3.
   3. If the previous command returns an error, run this command to install **pip3**.

   ```
   sudo apt install python3-pip
   ```

   4. Run this command to see if **pip3** is installed.

   ```
   pip3 --version
   ```

3. **Install the current AWS IoT Device SDK for Python**

   Install the AWS IoT Device SDK for Python and download the sample apps to your device.

   On your device, run these commands.

```
cd ~
python3 -m pip install awsiotsdk
```

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

JavaScript

In this section, you'll install Node.js, the npm package manager, and the AWS IoT Device SDK for JavaScript on your device. These instructions are for a Raspberry Pi running the Raspberry Pi OS. If you have another device or are using another operating system, you might need to adapt these instructions for your device.

1. **Install the latest version of Node.js**

   The AWS IoT Device SDK for JavaScript requires Node.js and the npm package manager to be installed on your Raspberry Pi.

   a. Download the latest version of the Node repository by entering this command.

   ```
   cd ~
   curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
   ```

   b. Install Node and npm.

```
sudo apt-get install -y nodejs
```

c. Verify the installation of Node.

```
node -v
```

Confirm that the command displays the Node version. This tutorial requires Node v10.0 or later. If the Node version isn't displayed, try downloading the Node repository again.

d. Verify the installation of npm.

```
npm -v
```

Confirm that the command displays the npm version. If the npm version isn't displayed, try installing Node and npm again.

e. Restart the device.

```
sudo shutdown -r 0
```

Continue after the device restarts.

2. **Install the AWS IoT Device SDK for JavaScript**

Install the AWS IoT Device SDK for JavaScript on your Raspberry Pi.

a. Install aws-crt, the common runtime library.

```
cd ~
npm install aws-crt
```

b. Install v2 of the AWS IoT Device SDK for JavaScript.

```
npm install aws-iot-device-sdk-v2
```

c. Clone the AWS IoT Device SDK for JavaScript repository into the `aws-iot-device-sdk-js-v2` directory of your *`home`* directory. On the Raspberry Pi, the *`home`* directory is `~/`, which is used as the *`home`* directory in the following commands. If your device uses a different path for the *`home`* directory, you must replace `~/` with the correct path for your device in the following commands.

These commands create the `~/aws-iot-device-sdk-js-v2` directory and copy the SDK code into it.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

d. Change to the `aws-iot-device-sdk-js-v2` directory that you created in the preceding step and install the SDK.

```
cd ~/aws-iot-device-sdk-js-v2
npm install
```

Install and run the sample app

In this section, you'll install and run the `pubsub` sample app found in the AWS IoT Device SDK. This app shows how your device uses the MQTT library to publish and subscribe to MQTT messages. The sample app subscribes to a topic, `topic_1`, publishes 10 messages to that topic, and displays the messages as they're received from the message broker.

Install the certificate files

The sample app requires the certificate files that authenticate the device to be installed on the device.

To install the device certificate files for the sample app

1. Create a `certs` subdirectory in your *home* directory by running these commands.

```
cd ~
mkdir certs
```

2. Into the `~/certs` directory, copy the private key, device certificate, and root CA certificate that you created earlier in the section called "Create AWS IoT resources" (p. 34).

   How you copy the certificate files to your device depends on the device and operating system and isn't described here. However, if your device supports a graphical user interface (GUI) and has a web browser, you can perform the procedure described in the section called "Create AWS IoT resources" (p. 34) from your device's web browser to download the resulting files directly to your device.

   The commands in the next section assume that your key and certificate files are stored on the device as shown in this table.

   Certificate file names

File	File path
Root CA certificate	`~/certs/Amazon-root-CA-1.pem`
Device certificate	`~/certs/device.pem.crt`
Private key	`~/certs/private.pem.key`

To run the sample app, you need the following information:

Application parameter values

Parameter	Where to find the value
your-iot-endpoint	In the AWS IoT console, choose **Manage**, and then choose **Things**. Choose the IoT thing you created for your device, **MyIotThing** was the name used earlier, and then choose **Interact**. On the thing details page, your endpoint is displayed in the **HTTPS** section.

The *your-iot-endpoint* value has a format of: *endpoint_id*`-ats.iot.`*region*`.amazonaws.com`, for example, `a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`.

Python

To install and run the sample app

1. Navigate to the sample app directory.

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. In the command line window, replace *your-iot-endpoint* as indicated and run this command.

```
python3 pubsub.py --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

3. Observe that the sample app:

   1. Connects to the AWS IoT service for your account.
   2. Subscribes to the message topic, **topic_1**, and displays the messages it receives on that topic.
   3. Publishes 10 messages to the topic, **topic_1**.
   4. Displays output similar to the following:

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to topic 'topic_1'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'topic_1': Hello World! [1]
Received message from topic 'topic_1': b'Hello World! [1]'
Publishing message to topic 'topic_1': Hello World! [2]
Received message from topic 'topic_1': b'Hello World! [2]'
Publishing message to topic 'topic_1': Hello World! [3]
Received message from topic 'topic_1': b'Hello World! [3]'
Publishing message to topic 'topic_1': Hello World! [4]
Received message from topic 'topic_1': b'Hello World! [4]'
Publishing message to topic 'topic_1': Hello World! [5]
Received message from topic 'topic_1': b'Hello World! [5]'
Publishing message to topic 'topic_1': Hello World! [6]
Received message from topic 'topic_1': b'Hello World! [6]'
Publishing message to topic 'topic_1': Hello World! [7]
Received message from topic 'topic_1': b'Hello World! [7]'
Publishing message to topic 'topic_1': Hello World! [8]
Received message from topic 'topic_1': b'Hello World! [8]'
Publishing message to topic 'topic_1': Hello World! [9]
Received message from topic 'topic_1': b'Hello World! [9]'
Publishing message to topic 'topic_1': Hello World! [10]
Received message from topic 'topic_1': b'Hello World! [10]'
10 message(s) received.
Disconnecting...
Disconnected!
```

If you're having trouble running the sample app, review the section called "Troubleshooting problems with the sample app" (p. 59).

You can also add the `--verbosity debug` parameter to the command line so the sample app displays detailed messages about what it's doing. That information might provide you the help you need to correct the problem.

JavaScript

To install and run the sample app

1. In your command line window, navigate to the `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub` directory that the SDK created and install the sample app by using these commands.

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. In the command line window, replace *`your-iot-endpoint`* as indicated and run this command.

   ```
   node dist/index.js --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

3. Observe that the sample app:

   1. Connects to the AWS IoT service for your account.
   2. Subscribes to the message topic, **topic_1**, and displays the messages it receives on that topic.
   3. Publishes 10 messages to the topic, **topic_1**.
   4. Displays output similar to the following:

   ```
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":1}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":2}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":3}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":4}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":5}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":6}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":7}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":8}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":9}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":10}
   ```

   If you're having trouble running the sample app, review the section called "Troubleshooting problems with the sample app" (p. 59).

   You can also add the `--verbosity debug` parameter to the command line so the sample app displays detailed messages about what it's doing. That information might provide you the help you need to correct the problem.

View messages from the sample app in the AWS IoT console

You can see the sample app's messages as they pass through the message broker by using the **MQTT client** in the **AWS IoT console**.

To view the MQTT messages published by the sample app

1. Review View MQTT messages with the AWS IoT MQTT client (p. 60). This helps you learn how to use the **MQTT client** in the **AWS IoT console** to view MQTT messages as they pass through the message broker.
2. Open the **MQTT client** in the **AWS IoT console**.
3. Subscribe to the topic, **topic_1**.
4. In your command line window, run the sample app again and watch the messages in the **MQTT client** in the **AWS IoT console**.

   Python

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

   JavaScript

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   node dist/index.js --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

Troubleshooting problems with the sample app

If you encounter an error when you try to run the sample app, here are some things to check.

Check the certificate

If the certificate is not active, AWS IoT will not accept any connection attempts that use it for authorization. When creating your certificate, it's easy to overlook the **Activate** button. Fortunately, you can activate your certificate from the AWS IoT console.

To check your certificate's activation

1. In the AWS IoT console, in the left menu, choose **Secure**, and then choose **Certificates**.
2. In the list of certificates, find the certificate you created for the exercise and check its status in the **Status** column.

   If you don't remember the certificate's name, check for any that are **Inactive** to see if they might be the one you're using.

   Choose the certificate in the list to open its detail page. In the detail page, you can see its **Create date** to help you identify the certificate.
3. **To activate an inactive certificate**, from the certificate's detail page, choose **Actions** and then choose **Activate**.

If you found the correct certificate and its active, but you're still having problems running the sample app, check its policy as the next step describes.

You can also try to create a new thing and a new certificate by following the steps in the section called "Create a thing object" (p. 36). If you create a new thing, you will need to give it new thing name and download the new certificate files to your device.

Check the policy attached to the certificate

Policies authorize actions in AWS IoT. If the certificate used to connect to AWS IoT does not have a policy, or does not have a policy that allows it to connect, the connection will be refused, even if the certificate is active.

To check the policies attached to a certificate

1. Find the certificate as described in the previous item and open its details page.
2. In the left menu of the certificate's details page, choose **Policies** to see the policies attached to the certificate
3. If there are no policies attached to the certificate, add one by choosing the **Actions** menu, and then choosing **Attach policy**.

   Choose the policy that you created earlier in the section called “Create AWS IoT resources” (p. 34).
4. If there is a policy attached, choose the policy tile to open its details page.

   In the details page, review the **Policy document** to make sure it contains the same information as the one you created in the section called “Create an AWS IoT policy” (p. 35).

Check the command line

Make sure you used the correct command line for your system. The commands used on Linux/macOS systems are often different from those used on Windows systems.

Check the endpoint address

Review the command you entered and double-check the endpoint address in your command to the one in your AWS IoT console.

Check the filenames of the certificate files

Compare the filenames in the command you entered to the filenames of the certificate files in the `certs` directory.

Some systems might require the filenames to be in quotes to work correctly.

Check the SDK installation

Make sure that your SDK installation is complete and correct.

If in doubt, reinstall the SDK on your device. In most cases, that's a matter of finding the section of the tutorial titled **Install the AWS IoT Device SDK for *`SDK language`*** and following the procedure again.

If you are using the **AWS IoT Device SDK for JavaScript**, remember to install the sample apps before you try to run them. Installing the SDK doesn't automatically install the sample apps. The sample apps must be installed manually after the SDK has been installed.

View MQTT messages with the AWS IoT MQTT client

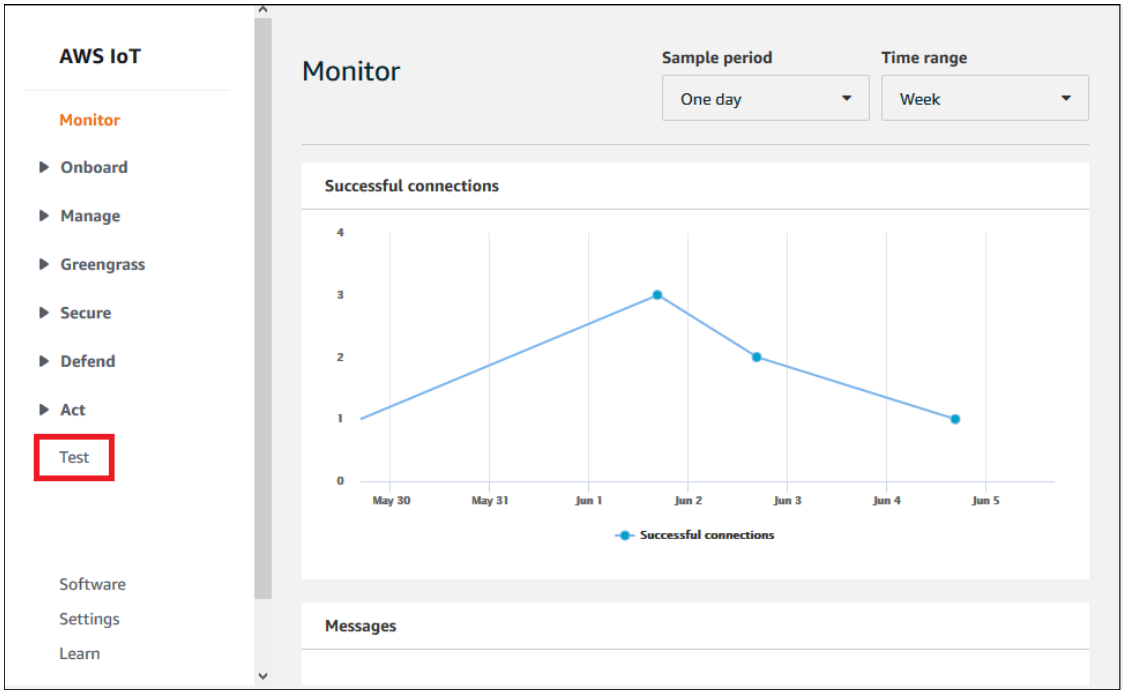
This section describes how to use the AWS IoT MQTT client in the AWS IoT console to watch the MQTT messages sent and received by AWS IoT. The example used in this section relates to the examples used in Getting started with AWS IoT Core (p. 17); however, you can replace the `topicName` used in the examples with any topic name or topic filter (p. 85) used by your IoT solution.

Devices publish MQTT messages that are identified by topics (p. 85) to communicate their state to AWS IoT, and AWS IoT publishes MQTT messages to inform the devices and apps of changes and events. You can use the MQTT client to subscribe to these topics and watch the messages as they occur. You can also use the MQTT client to publish MQTT messages to subscribed devices and services in your AWS account.

Viewing MQTT messages in the MQTT client

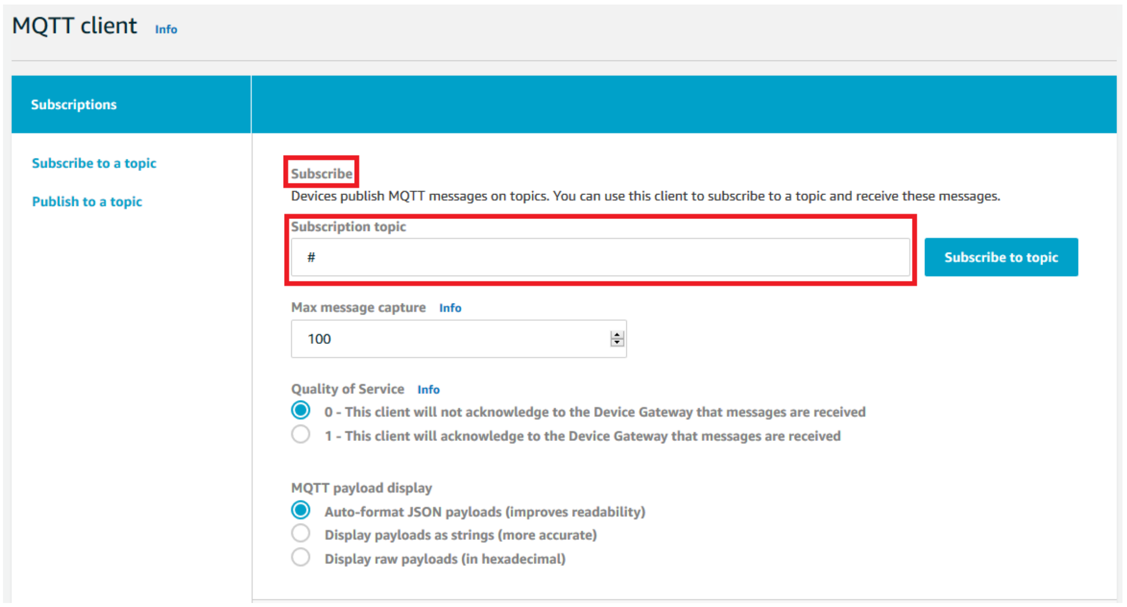
To view MQTT messages in the MQTT client

1. In the AWS IoT console, in the left menu, choose **Test**.



2. Subscribe to a `topicName` on which your device publishes. For the getting started sample app, subscribe to **#**, which subscribes to all message topics.

   Continuing with the getting started example, on the **Subscribe** page, in the **Subscription topic** field, enter **#**, and then choose **Subscribe to topic**.



Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.

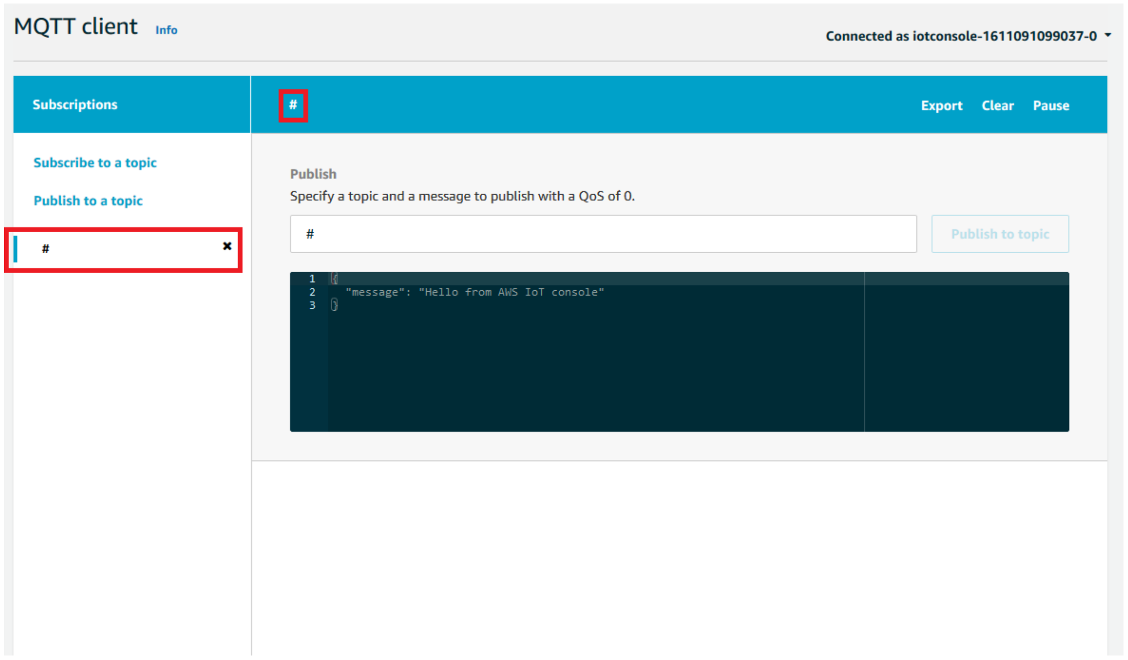
Subscription topic: #

Max message capture: 100

Quality of Service
[✓ 0 - This client will not acknowledge to the Device Gateway that messages are received]
1 - This client will acknowledge to the Device Gateway that messages are received

MQTT payload display
[✓ Auto-format JSON payloads (improves readability)]
Display payloads as strings (more accurate)
Display raw payloads (in hexadecimal)

The topic message log page, **#** opens and **#** appears in the **Subscriptions** column.



Specify a topic and a message to publish with a QoS of 0.

Topic: #

```
{
  "message": "Hello from AWS IoT console"
}
```

3. If the device that you configured in the section called “Configure your device” (p. 38) is running the example program, you should see the messages it sends to AWS IoT in the **#** message log. The message log entries will appear below the **Publish** section when messages with the subscribed topic are received by AWS IoT.
4. On the **#** message log page, you can also publish messages to a topic, but you'll need to specify the topic name. You cannot publish to the **#** topic.

   Messages published to subscribed topics appear in the message log as they are received, with the most recent message first.

Troubleshooting MQTT messages

Use the wild card topic filter

If your messages are not showing up in the message log as you expect, try subscribing to a wild card topic filter as described in Topic filters (p. 86). The MQTT multi-level wild card topic filter is the hash or pound sign (#) and can be used as the topic filter in the **Subscription topic** field.

Subscribing to the # topic filter subscribes to every topic received by the message broker. You can narrow the filter down by replacing elements of the topic filter path with a # multi-level wild card character or the '+' single-level wild-card character.

When using wild cards in a topic filter

- The multi-level wild card character must be the last character in the topic filter.
- The topic filter path can have only one single-level wild card character per topic level.

For example:

Topic filter	Displays messages with
`#`	Any topic name
`topic_1/#`	A topic name that starts with `topic_1/`
`topic_1/level_2/#`	A topic name that starts with `topic_1/level_2/`
`topic_1/+/level_3`	A topic name that starts with `topic_1/`, ends with `/level_3`, and has one element of any value in between.

For more information on topic filters, see Topic filters (p. 86).

Check for topic name errors

MQTT topic names and topic filters are case sensitive. If, for example, your device is publishing messages to `Topic_1` (with a capital *T*) instead of `topic_1`, the topic to which you subscribed, its messages would not appear in the MQTT client. Subscribing to the wild card topic filter, however would show that the device is publishing messages and you could see that it was using a topic name that was not the one you expected.

Publishing MQTT messages from the MQTT client

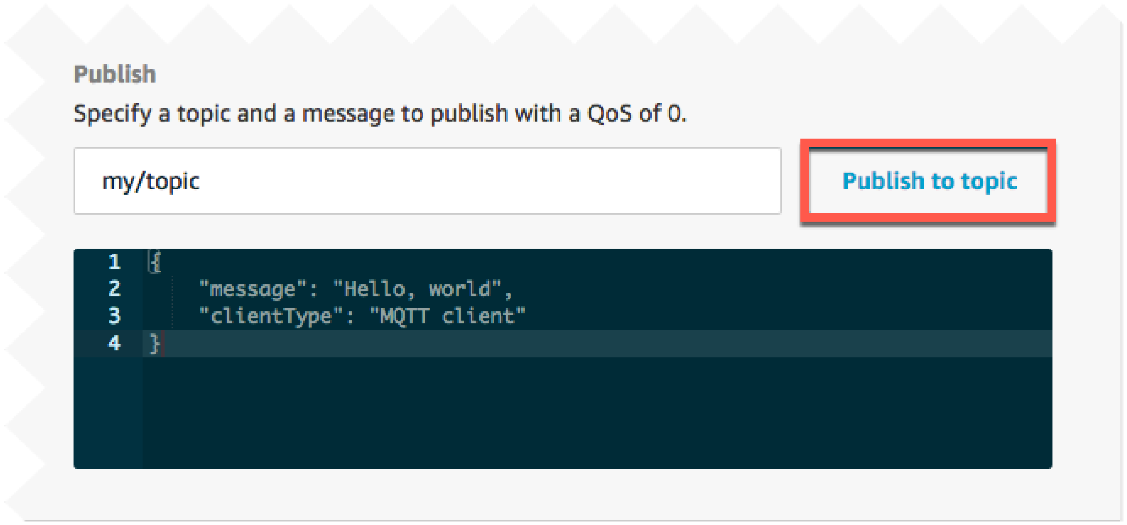
To publish a message to an MQTT topic

1. On the MQTT client page, in the **Publish** section, in the **Specify a topic and a message to publish** field, enter the *`topicName`* of your message. In this example, use **`my/topic`**.

   Note
   Do not use personally identifiable information in topic names, whether using them in the MQTT client or in your system implementation. Topic names can appear in unencrypted communications and reports.
2. In the message payload window, enter the following JSON:

```
{
    "message": "Hello, world",
    "clientType": "MQTT client"
}
```

3. Choose **Publish to topic** to publish your message to AWS IoT.



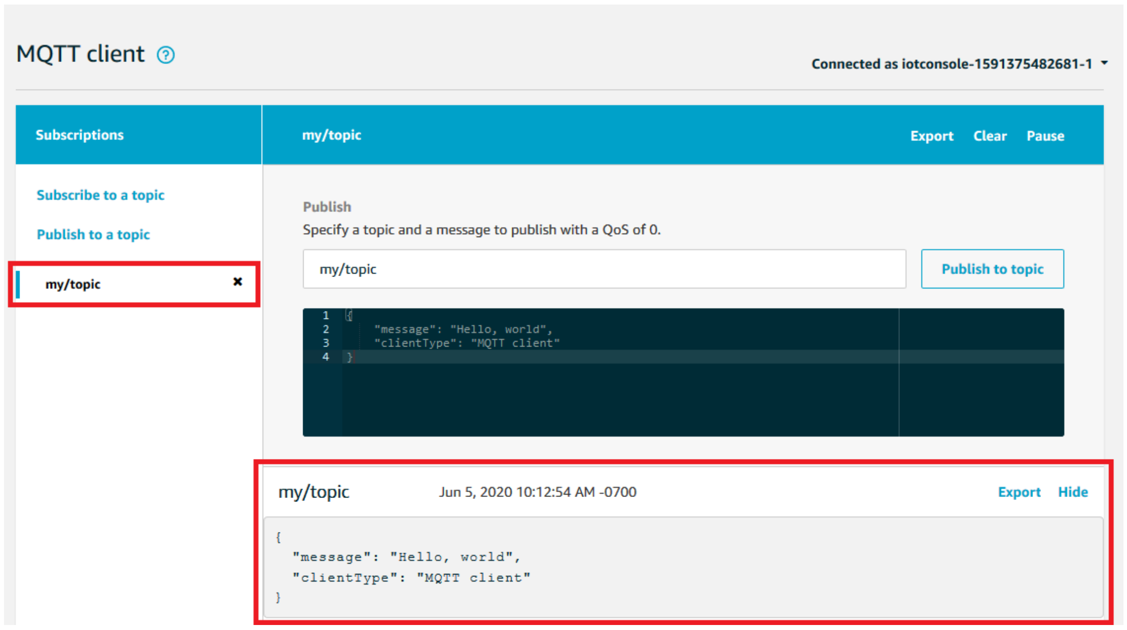
Publish

Specify a topic and a message to publish with a QoS of 0.

my/topic

```
{
    "message": "Hello, world",
    "clientType": "MQTT client"
}
```

4. In the **Subscriptions** column, choose **my/topic** to see the message. You should see the message appear in the MQTT client below the publish message payload window.



Publish

Specify a topic and a message to publish with a QoS of 0.

my/topic

```
{
    "message": "Hello, world",
    "clientType": "MQTT client"
}
```

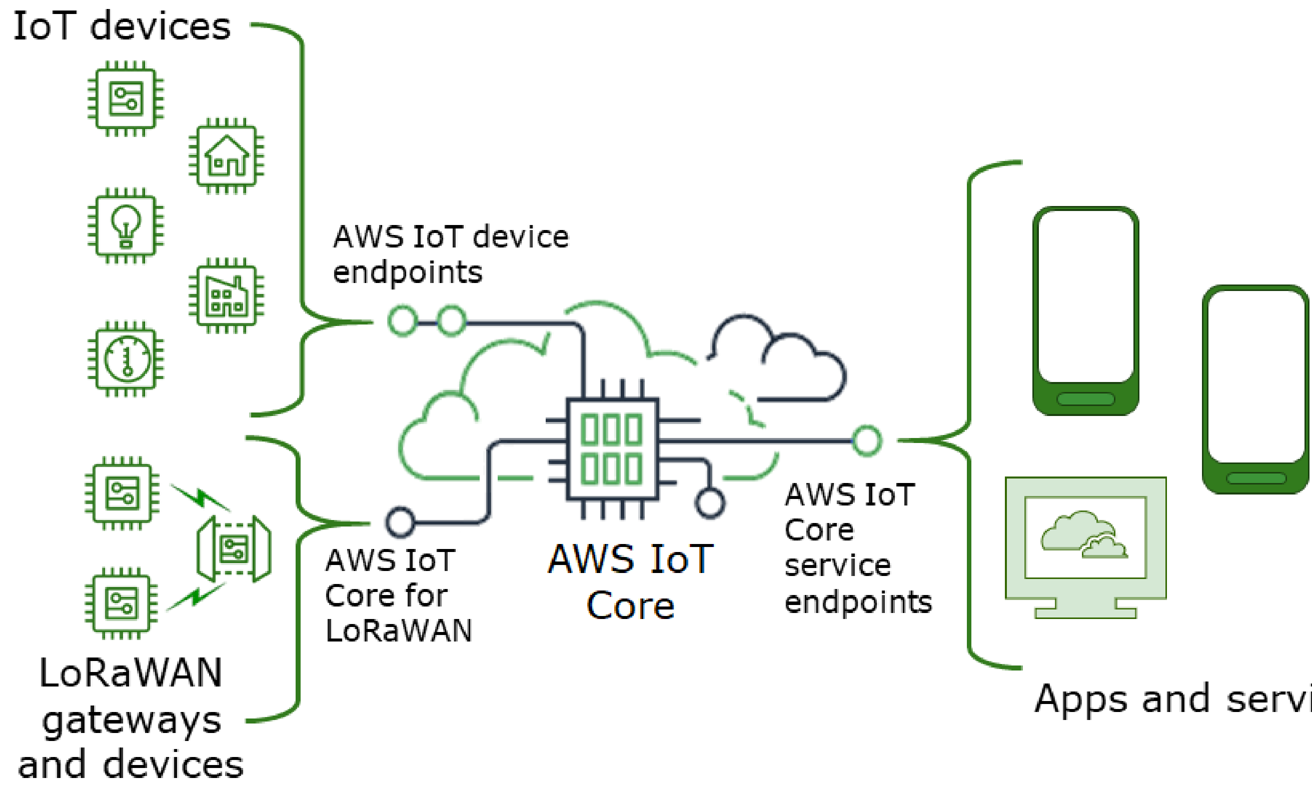
my/topic Jun 5, 2020 10:12:54 AM -0700

```
{
  "message": "Hello, world",
  "clientType": "MQTT client"
}
```

You can publish MQTT messages to other topics by changing the `topicName` in the **Specify a topic and a message to publish** field and choosing the **Publish to topic** button.

Connecting to AWS IoT Core

AWS IoT Core supports connections with IoT devices, wireless gateways, services, and apps. Devices connect to the AWS IoT Core so they can send data to and receive data from AWS IoT services and other devices. Apps and other services also connect to AWS IoT Core to control and manage the IoT devices and process the data from your IoT solution. This section describes how to choose the best way to connect and communicate with AWS IoT Core for each aspect of your IoT solution.



There are several ways to interact with AWS IoT. Apps and services can use the AWS IoT Core service endpoints (p. 65) and devices can connect to AWS IoT Core by using the AWS IoT device endpoints (p. 66) or AWS IoT Core for LoRaWAN gateways and devices (p. 67).

AWS IoT Core service endpoints

The AWS IoT Core service endpoints provide access to functions that control and manage your AWS IoT solution.

- **Endpoints**

  The AWS IoT Core service endpoints are Region specific and are listed in AWS IoT Core Endpoints and Quotas. The formats of the AWS IoT Core service endpoints are as follows.

Endpoint purpose	Endpoint format	Serves
AWS IoT Core control	`iot.aws-region.amazonaws.com`	AWS IoT Control Plane API
AWS IoT Core data	See ??? (p. 73).	AWS IoT Data Plane API
AWS IoT Core jobs data	See ??? (p. 73).	AWS IoT Jobs Data Plane API
AWS IoT Core secure tunneling	`api.tunneling.iot.aws-region.amazonaws.com`	AWS Secure Tunneling API

- **SDKs and tools**

  The AWS SDKs provide language-specific support for the AWS IoT Core APIs, and the APIs of other AWS services. The AWS Mobile SDKs provide app developers with platform-specific support for the AWS IoT Core API, and other AWS services on mobile devices.

  The AWS CLI provides command-line access to the functions provided by the AWS IoT service endpoints. AWS Tools for PowerShell provides tools to manage AWS services and resources in the PowerShell scripting environment.
- **Authentication**

  The service endpoints use IAM users and AWS credentials to authenticate users.
- **Learn more**

  For more information and links to SDK references, see the section called “Connecting to AWS IoT Core service endpoints” (p. 67).

AWS IoT device endpoints

The AWS IoT device endpoints support communication between your IoT devices and AWS IoT.

- **Endpoints**

  The device endpoints are specific to your account and you can see what they are by using the **describe-endpoint** command.

  For more information about these endpoints and the functions that they support, see the section called “AWS IoT device data and service endpoints” (p. 73).
- **SDKs**

  The AWS IoT Device SDKs (p. 75) provide language-specific support for the Message Queueing Telemetry Transport (MQTT) and WebSocket Secure (WSS) protocols, which devices use to communicate with AWS IoT. AWS Mobile SDKs (p. 72) also provide support for MQTT device communications, AWS IoT APIs, and the APIs of other AWS services on mobile devices.
- **Authentication**

  The device endpoints use X.509 certificates or AWS IAM users with credentials to authenticate users.
- **Learn more**

  For more information and links to SDK references, see the section called “AWS IoT Device SDKs” (p. 75).

AWS IoT Core for LoRaWAN gateways and devices

AWS IoT Core for LoRaWAN connects wireless gateways and devices to AWS IoT Core.

- **Endpoints**

  AWS IoT Core for LoRaWAN manages the gateway connections to account and Region-specific AWS IoT Core endpoints. Gateways can connect to your account's Configuration and Update Server (CUPS) endpoint that AWS IoT Core for LoRaWAN provides.

Endpoint purpose	Endpoint format	Serves
Configuration and Update Server (CUPS)	*account-specific-prefix*.cups.lorawan.*aws-region*.amazonaws.com:443	Gateway communication with the Configuration and Update Server provided by AWS IoT Core for LoRaWAN
LoRaWAN Network Server (LNS)	*account-specific-prefix*.gateway.lorawan.*aws-region*.amazonaws.com:443	Gateway communication with the LoRaWAN Network Server provided by AWS IoT Core for LoRaWAN

- **SDKs**

  The AWS IoT Wireless API that AWS IoT Core for LoRaWAN is built on is supported by the AWS SDK. For more information, see AWS SDKs and Toolkits.
- **Authentication**

  AWS IoT Core for LoRaWAN device communications use X.509 certificates to secure communications with AWS IoT.
- **Learn more**

  For more information about configuring and connecting wireless devices, see the section called "Connecting devices and gateways to AWS IoT Core for LoRaWAN" (p. 104).

Connecting to AWS IoT Core service endpoints

You can access the features that the AWS IoT Core services provide by using the AWS CLI, the AWS SDK for your preferred language, or by calling the REST API directly. We recommend using the AWS CLI or an AWS SDK to interact with AWS IoT Core because they incorporate the best practices for calling AWS services. Calling the REST APIs directly is an option, but you must provide the necessary security credentials that enable access to the API.

Note

IoT devices should use AWS IoT Device SDKs (p. 75). The Device SDKs are optimized for use on devices, support MQTT communication with AWS IoT, and support the AWS IoT APIs most used by devices. For more information about the Device SDKs and the features they provide, see AWS IoT Device SDKs (p. 75).

Mobile devices should use AWS Mobile SDKs (p. 72). The Mobile SDKs provide support for AWS IoT APIs, MQTT device communications, and the APIs of other AWS services on mobile devices. For more information about the Mobile SDKs and the features they provide, see AWS Mobile SDKs (p. 72).

The following sections describe the tools and SDKs that you can use to develop and interact with AWS IoT and other AWS services. For the complete list of AWS tools and development kits that are available to build and manage apps on AWS, see Tools to Build on AWS.

AWS CLI for AWS IoT Core

The AWS CLI provides command-line access to AWS APIs.

- **Installation**

  For information about how to install the AWS CLI, see Installing the AWS CLI.
- **Authentication**

  The AWS CLI uses credentials from your AWS account.
- **Reference**

  For information about the AWS CLI commands for these AWS IoT Core services, see:
  - AWS CLI Command Reference for IoT
  - AWS CLI Command Reference for IoT data
  - AWS CLI Command Reference for IoT jobs data
  - AWS CLI Command Reference for IoT secure tunneling

For tools to manage AWS services and resources in the PowerShell scripting environment, see AWS Tools for PowerShell.

AWS SDKs

With AWS SDKs, your apps and compatible devices can call AWS IoT APIs and the APIs of other AWS services. This section provides links to the AWS SDKs and to the API reference documentation for the APIs of the AWS IoT Core services.

The AWS SDKs support these AWS IoT Core APIs

- AWS IoT
- AWS IoT Data Plane
- AWS IoT Jobs Data Plane
- AWS IoT Secure Tunneling
- AWS IoT Wireless

C++

To install the AWS SDK for C++ and use it to connect to AWS IoT:

1. Follow the instructions in Getting Started Using the AWS SDK for C++

   These instructions describe how to:
   - Install and build the SDK from source files
   - Provide credentials to use the SDK with your AWS account
   - Initialize and shutdown the SDK in your app or service
   - Create a CMake project to build your app or service
2. Create and run a sample app. For sample apps that use the AWS SDK for C++, see AWS SDK for C++ Code Examples.

Documentation for the AWS IoT Core services that the AWS SDK for C++ supports

- Aws::IoT::IoTClient reference documentation
- Aws::IoTDataPlane::IoTDataPlaneClient reference documentation
- Aws::IoTJobsDataPlane::IoTJobsDataPlaneClient reference documentation
- Aws::IoTSecureTunneling::IoTSecureTunnelingClient reference documentation

Go

To install the AWS SDK for Go and use it to connect to AWS IoT:

1. Follow the instructions in Getting Started with the AWS SDK for Go

   These instructions describe how to:
   - Install the AWS SDK for GO
   - Get access keys for the SDK to access your AWS account
   - Import packages into the source code of our apps or services
2. Create and run a sample app. For sample apps that use the AWS SDK for Go, see AWS SDK for Go Code Examples.

Documentation for the AWS IoT Core services that the AWS SDK for Go supports

- IoT reference documentation
- IoTDataPlane reference documentation
- IoTJobsDataPlane reference documentation
- IoTSecureTunneling reference documentation

Java

To install the AWS SDK for Java and use it to connect to AWS IoT:

1. Follow the instructions in Getting Started with AWS SDK for Java 2.0

   These instructions describe how to:
   - Sign up for AWS and Create an IAM User
   - Download the SDK
   - Set up AWS Credentials and Region
   - Use the SDK with Apache Maven
   - Use the SDK with Gradle
2. Create and run a sample app using one of the AWS SDK for Java 2.0 Code Examples.
3. Review the SDK API reference documentation

Documentation for the AWS IoT Core services that the AWS SDK for Java supports

- IotClient reference documentation
- IotDataPlaneClient reference documentation
- IotJobsDataPlaneClient reference documentation
- IoTSecureTunnelingClient reference documentation

JavaScript

To install the AWS SDK for JavaScript and use it to connect to AWS IoT:

1. Follow the instructions in Setting Up the AWS SDK for JavaScript. These instructions apply to using the AWS SDK for JavaScript in the browser and with Node.JS. Make sure you follow the directions that apply to your installation.

   These instructions describe how to:
   - Check for the prerequisites
   - Install the SDK for JavaScript
   - Load the SDK for JavaScript
2. Create and run a sample app to get started with the SDK as the getting started option for your environment describes.
   - Get started with the AWS SDK for JavaScript in the Browser, or
   - Get started with the AWS SDK for JavaScript in Node.js

Documentation for the AWS IoT Core services that the AWS SDK for JavaScript supports

- AWS.Iot reference documentation
- AWS.IotData reference documentation
- AWS.IotJobsDataPlane reference documentation
- AWS.IotSecureTunneling reference documentation

.NET

To install the AWS SDK for .NET and use it to connect to AWS IoT:

1. Follow the instructions in Setting up your AWS SDK for .NET environment
2. Follow the instructions in Setting up your AWS SDK for .NET project

   These instructions describe how to:
   - Start a new project
   - Obtain and configure AWS credentials
   - Install AWS SDK packages
3. Create and run one of the sample programs in Working with AWS services in the AWS SDK for .NET
4. Review the SDK API reference documentation

Documentation for the AWS IoT Core services that the AWS SDK for .NET supports

- Amazon.IoT.Model reference documentation
- Amazon.IotData.Model reference documentation
- Amazon.IoTJobsDataPlane.Model reference documentation
- Amazon.IoTSecureTunneling.Model reference documentation

PHP

To install the AWS SDK for PHP and use it to connect to AWS IoT:

1. Follow the instructions in Getting Started with the AWS SDK for PHP Version 3

These instructions describe how to:

- Check for the prerequisites
- Install the SDK
- Apply the SDK to a PHP script

2. Create and run a sample app using one of the AWS SDK for PHP Version 3 Code Examples

Documentation for the AWS IoT Core services that the AWS SDK for PHP supports

- IoTClient reference documentation
- IoTDataPlaneClient reference documentation
- IoTJobsDataPlaneClient reference documentation
- IoTSecureTunnelingClient reference documentation

Python

To install the AWS SDK for Python (Boto3) and use it to connect to AWS IoT:

1. Follow the instructions in the AWS SDK for Python Quickstart

   These instructions describe how to:

   - Install the SDK
   - Configure the SDK
   - Use the SDK in your code

2. Create and run a sample program that uses the AWS SDK for Python

   This program displays the account's currently configured logging options. After you install the SDK and configure it for your account, you should be able to run this program.

```
import boto3
import json

# initialize client
iot = boto3.client('iot')

# get current logging levels, format them as JSON, and write them to stdout
response = iot.get_v2_logging_options()
print(json.dumps(response, indent=4))
```

   For more information about the function used in this example, see the section called “Configure AWS IoT logging” (p. 310).

Documentation for the AWS IoT Core services that the AWS SDK for Python supports

- IoT reference documentation
- IoTDataPlane reference documentation
- IoTJobsDataPlane reference documentation
- IoTSecureTunneling reference documentation

Ruby

To install the AWS SDK for Ruby and use it to connect to AWS IoT:

- Follow the instructions in Getting Started with the AWS SDK for Ruby

  These instructions describe how to:
  - Install the SDK
  - Configure the SDK
- Create and run the Hello World Tutorial

Documentation for the AWS IoT Core services that the AWS SDK for Ruby supports

- Aws::IoT::Client reference documentation
- Aws::IoTDataPlane::Client reference documentation
- Aws::IoTJobsDataPlane::Client reference documentation
- Aws::IoTSecureTunneling::Client reference documentation

AWS Mobile SDKs

The AWS Mobile SDKs provide mobile app developers platform-specific support for the APIs of the AWS IoT Core services, IoT device communication using MQTT, and the APIs of other AWS services.

Android

AWS SDK for Android

The AWS SDK for Android contains a library, samples, and documentation for developers to build connected mobile applications using AWS. This SDK also includes support for MQTT device communications and calling the APIs of the AWS IoT Core services. For more information, see the following:

- AWS Mobile SDK for Android on GitHub
- AWS Mobile SDK for Android Readme
- AWS Mobile SDK for Android Samples
- AWS SDK for Android API reference
- AWSIoTClient Class reference documentation

iOS

AWS SDK for iOS

The AWS SDK for iOS is an open-source software development kit, distributed under an Apache Open Source license. The SDK for iOS provides a library, code samples, and documentation to help developers build connected mobile applications using AWS. This SDK also includes support for MQTT device communications and calling the APIs of the AWS IoT Core services. For more information, see the following:

- AWS SDK for iOS on GitHub
- AWS SDK for iOS Readme
- AWS SDK for iOS Samples
- AWSIoT Class reference docs in the AWS SDK for iOS

REST APIs of the AWS IoT Core services

The REST APIs of the AWS IoT Core services can be called directly by using HTTP requests.

- **Endpoint URL**

  The service endpoints that expose the REST APIs of the AWS IoT Core services vary by Region and are listed in AWS IoT Core Endpoints and Quotas. You must use the endpoint for the Region that has the AWS IoT resources that you want to access, because AWS IoT resources are Region specific.
- **Authentication**

  The REST APIs of the AWS IoT Core services use AWS IAM credentials for authentication. For more information, see Signing AWS API requests in the AWS General Reference.
- **API reference**

  For information about the specific functions provided by the REST APIs of the AWS IoT Core services, see:
  - API reference for IoT.
  - API reference for IoT data.
  - API reference for IoT jobs data.
  - API reference for IoT secure tunneling.

Connecting devices to AWS IoT

Devices connect to AWS IoT and other services through AWS IoT Core. Through AWS IoT Core, devices send and receive messages using device endpoints that are specific to your account. The the section called "AWS IoT Device SDKs" (p. 75) support device communications using the MQTT and WSS protocols. For more information about the protocols that devices can use, see the section called "Device communication protocols" (p. 76).

The message broker

AWS IoT manages device communication through a message broker. Devices and clients publish messages to the message broker and also subscribe to messages that the message broker publishes. Messages are identified by an application-defined *topic* (p. 85). When the message broker receives a message published by a device or client, it republishes that message to the devices and clients that have subscribed to the message's topic. The message broker also forwards messages to the AWS IoT rules (p. 352) engine, which can act on the content of the message.

AWS IoT message security

Device connections to AWS IoT use the section called "X.509 client certificates" (p. 203) and AWS signature V4 for authentication. Device communications are secured by TLS version 1.2 and AWS IoT requires devices to send the Server Name Indication (SNI) extension when they connect. For more information, see Transport Security in AWS IoT.

AWS IoT device data and service endpoints

Each account has several device endpoints that are unique to the account and support specific IoT functions. The AWS IoT device data endpoints support a publish/subscribe protocol that is designed for the communication needs of IoT devices; however, other clients, such as apps and services, can also use this interface if their application requires the specialized features that these endpoints provide. The AWS IoT device service endpoints support device-centric access to security and management services.

To learn your account's device endpoint for a specific purpose, use the **describe-endpoint** CLI command shown here, or the `DescribeEndpoint` REST API, and provide the *`endpointType`* parameter value from the following table.

```
aws iot describe-endpoint --endpoint-type endpointType
```

This command returns an *`iot-endpoint`* in the following format: *`account-specific-prefix`*`.iot.`*`aws-region`*`.amazonaws.com`.

The `DescribeEndpoint` API does not have to be queried every time a new device is connected. The endpoints that you create presist forever and do not change once they are created.

Every customer has an `iot:Data-ATS` and an `iot:Data` endpoint. Each endpoint uses an X.509 certificate to authenticate the client. We strongly recommend that customers use the newer `iot:Data-ATS` endpoint type to avoid issues related to the widespread distrust of Symantec certificate authorities. We provide the `iot:Data` endpoint for devices to retrieve data from old endpoints that use VeriSign certificates for backward compatibility. For more information, see Server Authentication.

AWS IoT endpoints for devices

Endpoint purpose	***`endpointType`*** value	Description
IoT data	`iot:Data-ATS`	Used to send and receive data to and from the message broker, Device Shadow (p. 491), and Rules Engine (p. 352) components of AWS IoT. `iot:Data-ATS` returns an ATS signed data endpoint.
IoT data (legacy)	`iot:Data`	`iot:Data` returns a VeriSign signed data endpoint provided for backward compatibility.
IoT credential access	`iot:CredentialProvider`	Used to exchange a device's built-in X.509 certificate for temporary credentials to connect directly with other AWS services. For more information about connecting to other AWS services, see Authorizing Direct Calls to AWS Services (p. 269).
IoT job management	`iot:Jobs`	Used to enable devices to interact with the AWS IoT Jobs service using the Jobs Device HTTPS APIs (p. 616).
IoT device advisor (preview)	`iot:DeviceAdvisor`	A test endpoint type used for testing devices with Device Advisor. For more information, see ??? (p. 898).
IoT data beta (preview)	`iot:Data-Beta`	An endpoint type reserved for beta releases. For information about its current use, see ??? (p. 100).

You can also use your own fully-qualified domain name (FQDN), such as `example.com`, and the associated server certificate to connect devices to AWS IoT by using the section called “Configurable endpoints (beta)” (p. 100), which is currently in public beta.

AWS IoT Device SDKs

The AWS IoT Device SDKs help you connect your IoT devices to AWS IoT Core and they support MQTT and MQTT over WSS protocols.

The AWS IoT Device SDKs differ from the AWS SDKs in that the AWS IoT Device SDKs support the specialized communications needs of IoT devices, but don't support all of the services supported by the AWS SDKs. The AWS IoT Device SDKs are compatible with the AWS SDKs that support all of the AWS services; however, they use different authentication methods and connect to different endpoints, which could make using the AWS SDKs impractical on an IoT device.

Mobile devices

The the section called “AWS Mobile SDKs” (p. 72) support both MQTT device communications, some of the AWS IoT service APIs, and the APIs of other AWS services. If you're developing on a supported mobile device, review its SDK to see if it's the best option for developing your IoT solution.

C++

AWS IoT C++ Device SDK

The AWS IoT C++ Device SDK allows developers to build connected applications using AWS and the APIs of the AWS IoT Core services. Specifically, this SDK was designed for devices that are not resource constrained and require advanced features such as message queuing, multi-threading support, and the latest language features. For more information, see the following:

- AWS IoT C++ Device SDK v2 on GitHub
- AWS IoT C++ Device SDK v2 Readme
- AWS IoT C++ Device SDK v2 Samples

Python

AWS IoT Device SDK for Python

The AWS IoT Device SDK for Python makes it possible for developers to write Python scripts to use their devices to access the AWS IoT platform through MQTT or MQTT over the WebSocket protocol. By connecting their devices to the APIs of the AWS IoT Core services, users can securely work with the message broker, rules, and Device Shadow service that AWS IoT Core provides and with other AWS services like AWS Lambda, Amazon Kinesis, and Amazon S3, and more.

- AWS IoT Device SDK for Python v2 on GitHub
- AWS IoT Device SDK for Python v2 Readme
- AWS IoT Device SDK for Python v2 Samples
- AWS IoT Device SDK for Python v2 API documentation

JavaScript

AWS IoT Device SDK for JavaScript

The AWS IoT Device SDK for JavaScript makes it possible for developers to write JavaScript applications that access APIs of the AWS IoT Core using MQTT or MQTT over the WebSocket protocol. It can be used in Node.js environments and browser applications. For more information, see the following:

- AWS IoT Device SDK for JavaScript v2 on GitHub
- AWS IoT Device SDK for JavaScript v2 Readme
- AWS IoT Device SDK for JavaScript v2 Samples
- AWS IoT Device SDK for JavaScript v2 API documentation

Java

AWS IoT Device SDK for Java

The AWS IoT Device SDK for Java makes it possible for Java developers to access the APIs of the AWS IoT Core through MQTT or MQTT over the WebSocket protocol. The SDK supports the Device Shadow service. You can access shadows by using HTTP methods, including GET, UPDATE, and DELETE. The SDK also supports a simplified shadow access model, which allows developers to exchange data with shadows by using getter and setter methods, without having to serialize or deserialize any JSON documents. For more information, see the following:

- AWS IoT Device SDK for Java v2 on GitHub
- AWS IoT Device SDK for Java v2 Readme
- AWS IoT Device SDK for Java v2 Samples

Embedded C

AWS IoT Device SDK for Embedded C

> **Important**
> This SDK is intended for use by experienced embedded-software developers.

The AWS IoT Device SDK for Embedded C (C-SDK) is a collection of C source files under the MIT open source license that can be used in embedded applications to securely connect IoT devices to AWS IoT Core. It includes an MQTT, JSON Parser, and AWS IoT Device Shadow library. It is distributed in source form and intended to be built into customer firmware along with application code, other libraries and, optionally, an RTOS (Real Time Operating System).

For Fleet Provisioning, use the `v4_beta_deprecated` version of the AWS IoT Device SDK for Embedded C at https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/v4_beta_deprecated.

The AWS IoT Device SDK for Embedded C is generally targeted at resource constrained devices that require an optimized C language runtime. You can use the SDK on any operating system and host it on any processor type (for example, MCUs and MPUs). If your device has sufficient memory and processing resources available, we recommend that you use one of the other AWS IoT Device and Mobile SDKs, such as the AWS IoT Device SDK for C++, Java, JavaScript, or Python.

For more information, see the following:

- AWS IoT Device SDK for Embedded C on GitHub
- AWS IoT Device SDK for Embedded C Readme
- AWS IoT Device SDK for Embedded C Samples

Device communication protocols

AWS IoT Core supports devices and clients that use the MQTT and the MQTT over WebSocket Secure (WSS) protocols to publish and subscribe to messages, and devices and clients that use the HTTPS protocol to publish messages. All protocols support IPv4 and IPv6. This section describes the different connection options for devices and clients.

TLS v1.2

AWS IoT Core uses TLS version 1.2 to encrypt all communication. Clients must also send the Server Name Indication (SNI) TLS extension. Connection attempts that don't include the SNI are refused. For more information, see Transport Security in AWS IoT.

The AWS IoT Device SDKs (p. 75) support MQTT and MQTT over WSS and support the security requirements of client connections. We recommend using the AWS IoT Device SDKs (p. 75) to connect clients to AWS IoT.

Protocols, port mappings, and authentication

How a device or client connects to the message broker by using a device endpoint depends on the protocol it uses. The following table lists the protocols that the AWS IoT device endpoints support and the authentication methods and ports they use.

Protocols, authentication, and port mappings

Protocol	Operations supported	Authentication	Port	ALPN protocol name
MQTT over WebSocket	Publish, Subscribe	Signature Version 4	443	N/A
MQTT over WebSocket	Publish, Subscribe	Custom authentication	443	N/A
MQTT	Publish, Subscribe	X.509 client certificate	443[†]	`x-amzn-mqtt-ca`
MQTT	Publish, Subscribe	X.509 client certificate	8883	N/A
MQTT	Publish, Subscribe	Custom authentication	443[†]	`mqtt`
HTTPS	Publish only	Signature Version 4	443	N/A
HTTPS	Publish only	X.509 client certificate	443[†]	`x-amzn-http-ca`
HTTPS	Publish only	X.509 client certificate	8443	N/A
HTTPS	Publish only	Custom authentication	443	N/A

Application Layer Protocol Negotiation (ALPN)

[†]Clients that connect on port 443 with X.509 client certificate authentication must implement the Application Layer Protocol Negotiation (ALPN) TLS extension and use the ALPN protocol name listed in the ALPN ProtocolNameList sent by the client as part of the `ClientHello` message.
On port 443, the IoT:Data-ATS (p. 74) endpoint supports ALPN x-amzn-http-ca HTTP, but the IoT:Jobs (p. 74) endpoint does not.
On port 8443 HTTPS and port 443 MQTT with ALPN x-amzn-mqtt-ca, custom authentication (p. 222) can't be used.

Clients connect to their AWS account's device endpoints. See the section called “AWS IoT device data and service endpoints” (p. 73) for information about how to find your account's device endpoints.

Connecting to AWS IoT Core

Protocol	Endpoint or URL
MQTT	`iot-endpoint`
MQTT over WSS	`wss://iot-endpoint/mqtt`
HTTPS	`https://iot-endpoint/topics`

Choosing a protocol for your device communication

For most IoT device communication through the device endpoints, you'll want to use the MQTT or MQTT over WSS protocols; however, the device endpoints also support HTTPS. The following table compares how AWS IoT Core uses the two protocols for device communication.

AWS IoT device protocols side-by-side

Feature	MQTT (p. 78)	HTTPS (p. 82)
Publish/Subscribe support	Publish and subscribe	Publish only
SDK support	AWS Device SDKs (p. 75) support MQTT and WSS protocols	No SDK support, but you can use language-specific methods to make HTTPS requests
Quality of Service support	MQTT QoS levels 0 and 1 (p. 80)	No QoS support
Can receive messages missed while device was offline	Yes	No
`clientId` field support	Yes	No
Device disconnection detection	Yes	No
Secure communications	Yes. See Protocols, port mappings, and authentication (p. 77)	Yes. See Protocols, port mappings, and authentication (p. 77)
Duration of connection	Up to several weeks	Up to 24 hours
Topic definitions	Application defined	Application defined
Message data format	Application defined	Application defined
Protocol overhead	Lower	Higher
Power consumption	Lower	Higher

MQTT

MQTT is a lightweight and widely adopted messaging protocol that is designed for constrained devices. AWS IoT support for MQTT is based on the MQTT v3.1.1 specification, with some differences. For information about how AWS IoT differs from the MQTT v3.1.1 specification, see the section called "AWS IoT differences from MQTT version 3.1.1 specification" (p. 82).

AWS IoT Core supports device connections that use the MQTT protocol and MQTT over WSS protocol. The AWS IoT Device SDKs (p. 75) support both protocols and are the recommended ways to connect devices to AWS IoT. The AWS IoT Device SDKs support the functions necessary for devices and clients to connect to and access AWS IoT Core services and they support the authentication protocols that the AWS IoT services require. For information about how to connect to AWS IoT using the AWS Device SDKs and links to examples of AWS IoT in the supported languages, see the section called “Connecting with MQTT using the AWS IoT Device SDKs” (p. 79). For more information about authentication methods and the port mappings for MQTT messages, see Protocols, port mappings, and authentication (p. 77).

While we recommend using the AWS IoT Device SDKs to connect to AWS IoT, they are not required. If you do not use the AWS IoT Device SDKs, however, you must provide the necessary connection and communication security. Clients must send the Server Name Indication (SNI) TLS extension in the connection request. Connection attempts that don't include the SNI are refused. For more information, see Transport Security in AWS IoT. Clients that use IAM users and AWS credentials to authenticate clients must provide the correct Signature Version 4 authentication.

Connecting with MQTT using the AWS IoT Device SDKs

This section contains links to the AWS IoT Device SDKs and to the source code of sample programs that illustrate how to connect a device to AWS IoT. The sample apps linked here show how to connect to AWS IoT using the MQTT protocol and MQTT over WSS.

C++

Using the AWS IoT C++ Device SDK to connect devices

- Source code of a sample app that shows an MQTT connection example in C++
- AWS IoT C++ Device SDK v2 on GitHub

Python

Using the AWS IoT Device SDK for Python to connect devices

- Source code of a sample app that shows an MQTT connection example in Python
- AWS IoT Device SDK for Python v2 on GitHub

JavaScript

Using the AWS IoT Device SDK for JavaScript to connect devices

- Source code of a sample app that shows an MQTT connection example in JavaScript
- AWS IoT Device SDK for JavaScript v2 on GitHub

Java

Using the AWS IoT Device SDK for Java to connect devices

- Source code of a sample app that shows an MQTT connection example in Java
- AWS IoT Device SDK for Java v2 on GitHub

Embedded C

Using the AWS IoT Device SDK for Embedded C to connect devices

Important
This SDK is intended for use by experienced embedded-software developers.

- Source code of a sample app that shows an MQTT connection example in Embedded C
- AWS IoT Device SDK for Embedded C on GitHub

MQTT Quality of Service (QoS) options

AWS IoT and the AWS IoT Device SDKs support the MQTT Quality of Service (QoS) levels `0` and `1`. The MQTT protocol defines a third level of QoS, level `2`, but AWS IoT does not support it. Only the MQTT protocol supports the QoS feature. HTTPS does not support QoS.

This table describes how each QoS level affects messages published to and by the message broker.

With a QoS level of...	The message is...	Comments
QoS level 0	Sent zero or more times	This level should be used for messages that are sent over reliable communication links or that can be missed without a problem.
QoS level 1	Sent at least one time, and then repeatedly until a `PUBACK` response is received	The message is not considered complete until the sender receives a `PUBACK` response to indicate successful delivery.

Using MQTT persistent sessions

Persistent sessions store a client's subscriptions and messages, with a quality of service (QoS) of 1, that have not been acknowledged by the client. When a disconnected device reconnects to a persistent session, the session resumes, its subscriptions are reinstated, and subscribed messages received prior to the reconnection and that have not been acknowledged by the client are sent to the client.

Creating a persistent session

You create an MQTT persistent session by sending a `CONNECT` message and setting the `cleanSession` flag to `0`. If no session exists for the client sending the `CONNECT` message, a new persistent session is created. If a session already exists for the client, the client resumes the existing session.

Operations during a persistent session

Clients use the `sessionPresent` attribute in the connection acknowledged (`CONNACK`) message to determine if a persistent session is present. If `sessionPresent` is `1`, a persistent session is present and any stored messages for the client are delivered to the client immediately after the client receives the `CONNACK`, as described in Message traffic after reconnection to a persistent session (p. 80). If `sessionPresent` is `1`, there is no need for the client to resubscribe. However, if `sessionPresent` is `0`, no persistent session is present and the client must resubscribe to its topic filters.

After the client joins a persistent session, it can publish messages and subscribe to topic filters without any additional flags on each operation.

Message traffic after reconnection to a persistent session

A persistent session represents an ongoing connection between a client and an MQTT message broker. When a client connects to the message broker using a persistent session, the message broker saves all subscriptions that the client makes during the connection. When the client disconnects, the message broker stores unacknowledged QoS 1 messages and new QoS 1 messages published to topics to which the client is subscribed. Messages are stored according to account limit, messages that exceed that limit will be dropped. For more information about persistent message limits, see AWS IoT Core endpoints and quotas. When the client reconnects to its persistent session, all subscriptions are reinstated and all stored messages are sent to the client at a maximum rate of 10 messages per second.

After reconnection, the stored messages are sent to the client, at a rate that is limited to 10 stored messages per second, along with any current message traffic until the `Publish requests per second per connection` limit is reached. Because the delivery rate of the stored messages is limited it will take several seconds to deliver all stored messages if a session has more than 10 stored messages to deliver after reconnection.

Ending a persistent session

The following conditions describe how persistent sessions can end.

- When the persistent session expiration time elapses. The persistent session expiration timer starts when the message broker detects that a client has disconnected, either by the client disconnecting or the connection timing out.
- When the client sends a `CONNECT` message that sets the `cleanSession` flag to `1`.

> **Note**
> The stored messages waiting to be sent to the client when a session ends are discarded; however, they are still billed at the standard messaging rate, even though they could not be sent. For more information about message pricing, see AWS IoT Core Pricing. You can configure the expiration time interval.

Reconnection after a persistent session has expired

If a client doesn't reconnect to its persistent session before it expires, the session ends and its stored messages are discarded. When a client reconnects after the session has expired with a `cleanSession` flag to `0`, the service creates a new persistent session. Any subscriptions or messages from the previous session are not available to this session because they were discarded when the previous session expired.

Persistent session message charges

Messages are charged to your AWS account when the message broker receives a message from a client and sends a message to a client. When a device with a persistent session is not connected, the messages that are stored to be sent when the client reconnects are also charged to your account. When a persistent session expires, messages that have been stored for this session are discarded; however, your account is still charged for their storage. For more information about message pricing, see AWS IoT Core pricing - Messaging.

The default persistent session expiration time of one hour can be increased by using the standard limit increase process. Note that increasing the session expiration time might increase your message charges. The additional time could allow for more messages to be stored for the disconnected device and those additional messages would be charged to your account. The session expiration time is approximate and a session could persist for up to 30 minutes longer than the account limit; however, a session will not be shorter than the account limit. For more information about session limits, see AWS Service Quotas.

Using connectAttributes

`ConnectAttributes` allow you to specify what attributes you want to use in your connect message in your IAM policies such as `PersistentConnect` and `LastWill`. With `ConnectAttributes`, you can build policies that don't give devices access to new features by default, which can be helpful if a device is compromised.

`connectAttributes` supports the following features:

`PersistentConnect`

Use the `PersistentConnect` feature to save all subscriptions the client makes during the connection when the connection between the client and broker is interrupted.

`LastWill`

Use the `LastWill` feature to publish a message to the `LastWillTopic` when a client unexpectedly disconnects.

By default, your policy has non-persistent connection and there are no attributes passed for this connection. You must specify a persistent connection in your IAM policy if you want to have one.

For `ConnectAttributes` examples, see Connect Policy Examples (p. 245).

AWS IoT differences from MQTT version 3.1.1 specification

The message broker implementation is based on the MQTT v3.1.1 specification, but it differs from the specification in these ways:

- AWS IoT supports MQTT quality of service (QoS) levels 0 and 1 only. AWS IoT doesn't support publishing or subscribing with QoS level 2. When QoS level 2 is requested, the message broker doesn't send a PUBACK or SUBACK.
- In AWS IoT, subscribing to a topic with QoS level 0 means that a message is delivered zero or more times. A message might be delivered more than once. Messages delivered more than once might be sent with a different packet ID. In these cases, the DUP flag is not set.
- When responding to a connection request, the message broker sends a CONNACK message. This message contains a flag to indicate if the connection is resuming a previous session.
- When a client subscribes to a topic, there might be a delay between the time the message broker sends a SUBACK and the time the client starts receiving new matching messages.
- The MQTT specification provides a provision for the publisher to request that the broker retain the last message sent to a topic and send it to all future topic subscribers. AWS IoT doesn't support retained messages. If a request is made to retain messages, the connection is disconnected.
- The message broker uses the client ID to identify each client. The client ID is passed in from the client to the message broker as part of the MQTT payload. Two clients with the same client ID can't be connected concurrently to the message broker. When a client connects to the message broker using a client ID that another client is using, the new client connection is accepted and the previously connected client is disconnected.
- On rare occasions, the message broker might resend the same logical PUBLISH message with a different packet ID.
- The message broker doesn't guarantee the order in which messages and ACK are received.

HTTPS

Clients can publish messages by making requests to the REST API using the HTTP 1.0 or 1.1 protocols. For the authentication and port mappings used by HTTP requests, see Protocols, port mappings, and authentication (p. 77).

Note
Unlike MQTT, HTTPS does not support a `clientId` value. So, while a `clientId` is available when using MQTT, it's not available when using HTTPS.

HTTPS message URL

Devices and clients publish their messages by making POST requests to a client-specific endpoint and a topic-specific URL:

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1"
```

- *IoT_data_endpoint* is the AWS IoT device data endpoint (p. 73). You can find the endpoint in the AWS IoT console on the thing's details page or on the client by using the AWS CLI command:

  aws iot describe-endpoint --endpoint-type iot:Data-ATS

  The endpoint should look something like this: `a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`
- *url_encoded_topic_name* is the full topic name (p. 85) of the message being sent.

HTTPS message code examples

These are some examples of how to send an HTTPS message to AWS IoT.

Python

```
import requests
import argparse

# define command-line parameters
parser = argparse.ArgumentParser(description="Send messages through an HTTPS
 connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom
 endpoint, not including a port. " +
                                                      "Ex: \"abcdEXAMPLExyz-ats.iot.us-
east-1.amazonaws.com\"")
parser.add_argument('--cert', required=True, help="File path to your client
 certificate, in PEM format.")
parser.add_argument('--key', required=True, help="File path to your private key, in PEM
 format.")
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to
 publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Message to publish. " +
                                                      "Specify empty string to publish
 nothing.")

# parse and load command-line parameter values
args = parser.parse_args()

# create and format values for HTTPS request
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')

# make request
publish = requests.request('POST',
            publish_url,
            data=publish_msg,
            cert=[args.cert, args.key])

# print results
print("Response status: ", str(publish.status_code))
if publish.status_code == 200:
        print("Response body:", publish.text)
```

CURL

You can use curl from a client or device to send a message to AWS IoT.

To use curl to send a message from an AWS IoT client device

1. Check the **curl** version.

   a. On your client, run this command at a command prompt.

curl --help

In the help text, look for the TLS options. You should see the `--tlsv1.2` option.

b. If you see the `--tlsv1.2` option, continue.

c. If you don't see the `--tlsv1.2` option or you get a `command not found` error, you might need to update or install curl on your client or install `openssl` before you continue.

2. Install the certificates on your client.

   Copy the certificate files that you created when you registered your client (thing) in the AWS IoT console. Make sure you have these three certificate files on your client before you continue.

   - The CA certificate file (*`Amazon-root-CA-1.pem`* in this example).
   - The client's certificate file (*`device.pem.crt`* in this example).
   - The client's private key file (*`private.pem.key`* in this example).

3. Create the **curl** command line, replacing the replaceable values for those of your account and system.

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://IoT_data_endpoint:8443/topics/topic?qos=1"
```

   --tlsv1.2

      Use TLS 1.2 (SSL).

   --cacert *`Amazon-root-CA-1.pem`*

      The file name and path, if necessary, of the CA certificate to verify the peer.

   --cert *`device.pem.crt`*

      The client's certificate file name and path, if necessary.

   --key *`private.pem.key`*

      The client's private key file name and path, if necessary.

   --request POST

      The type of HTTP request (in this case, POST).

   --data "*`{ \"message\": \"Hello, world\" }`*"

      The HTTP POST data you want to publish. In this case, it's a JSON string, with the internal quotation marks escaped with the backslash character (\).

   "https://*`IoT_data_endpoint`*:8443/topics/*`topic`*?qos=1"

      The URL of your client's AWS IoT device data endpoint, followed by the HTTPS port, `:8443`, which is then followed by the keyword, `/topics/` and the topic name, `topic`, in this case. Specify the Quality of Service as the query parameter, `?qos=1`.

4. Open the MQTT test client in the AWS IoT console.

   Follow the instructions in View MQTT messages with the AWS IoT MQTT client (p. 60) and configure the console to subscribe to messages with the topic name of *`topic`* used in your **curl** command, or use the wildcard topic filter of `#`.

5. Test the command.

While monitoring the topic in the test client of the AWS IoT console, go to your client and issue the curl command line that you created in step 3. You should see your client's messages in the console.

MQTT topics

MQTT topics identify AWS IoT messages. AWS IoT clients identify the messages they publish by giving the messages topic names. Clients identify the messages to which they want to subscribe (receive) by registering a topic filter with AWS IoT Core. The message broker uses topic names and topic filters to route messages from publishing clients to subscribing clients.

The message broker uses topics to identify messages sent using MQTT and sent using HTTP to the HTTPS message URL (p. 82).

While AWS IoT supports some reserved system topics (p. 87), most MQTT topics are created and managed by you, the system designer. AWS IoT uses topics to identify messages received from publishing clients and select messages to send to subscribing clients, as described in the following sections. Before you create a topic namespace for your system, review the characteristics of MQTT topics to create the hierarchy of topic names that works best for your IoT system.

Topic names

Topic names and topic filters are UTF-8 encoded strings. They can represent a hierarchy of information by using the forward slash (/) character to separate the levels of the hierarchy. For example, this topic name could refer to a temperature sensor in room 1:

- `sensor/temperature/room1`

In this example, there might also be other types of sensors in other rooms with topic names such as:

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

Note
As you consider topic names for the messages in your system, keep in mind:

- Topic names and topic filters are case sensitive.
- Topic names must not contain personally identifiable information.
- Topic names that begin with a $ are reserved topics (p. 87) to be used only by AWS IoT Core.
- AWS IoT Core can't send or receive messages between AWS accounts or Regions.

For more information on designing your topic names and namespace, see our whitepaper, Designing MQTT Topics for AWS IoT Core.

For examples of how apps can publish and subscribe to messages, start with Getting started with AWS IoT Core (p. 17) and AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client (p. 990).

Important
The topic namespace is limited to an AWS account and Region. For example, the `sensor/temp/room1` topic used by an AWS account in one Region is distinct from the `sensor/temp/room1` topic used by the same AWS account in another Region or used by any other AWS account in any Region.

Topic ARN

All topic ARNs (Amazon Resource Names) have the following form:

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

For example, `arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor` is an ARN for the topic `application/topic/device/sensor`.

Topic filters

Subscribing clients register topic filters with the message broker to specify the message topics that the message broker should send to them. A topic filter can be a single topic name to subscribe to a single topic name or it can include wildcard characters to subscribe to multiple topic names at the same time.

Publishing clients can't use wildcard characters in the topic names they publish.

The following table lists the wildcard characters that can be used in a topic filter.

Topic wildcards

Wildcard character	Matches	Notes
#	All strings at and below its level in the topic hierarchy.	Must be the last character in the topic filter. Must be the only character in its level of the topic hierarchy. Can be used in a topic filter that also contains the + wildcard character.
+	Any string in the level that contains the character.	Must be the only character in its level of the topic hierarchy. Can be used in multiple levels of a topic filter.

Using wildcards with the previous sensor topic name examples:

- A subscription to `sensor/#` receives messages published to `sensor/`, `sensor/temperature`, `sensor/temperature/room1`, but not messages published to `Sensor`.
- A subscription to `sensor/+/room1` receives messages published to `sensor/temperature/room1` and `sensor/humidity/room1`, but not messages sent to `sensor/temperature/room2` or `sensor/humidity/room2`.

Topic filter ARN

All topic filter ARNs (Amazon Resource Names) have the following form:

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

For example, `arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/#/sensor` is an ARN for the topic filter `application/topic/#/sensor`.

MQTT message payload

The message payload sent in your MQTT messages is not specified by AWS IoT, unless the message payload is for one of the the section called “Reserved topics” (p. 87). Rather, you define the message payload for your topics to best accommodate your application's needs, within the constraints of the AWS IoT Core Service Quotas for Protocols.

Using a JSON format for your message payload enables the AWS IoT rules engine to parse your messages and apply SQL queries to it. If your application doesn't require the Rules engine to apply SQL queries to your message payloads, you can use any data format that your application requires. For information about limitations and reserved characters in a JSON document used in SQL queries, see JSON extensions (p. 484).

For more information about designing your MQTT topics and their corresponding message payloads, see Designing MQTT Topics for AWS IoT Core.

Reserved topics

Topics that begin with a dollar sign ($) are reserved for use by AWS IoT. You can subscribe and publish to these reserved topics as they allow; however, you can't create new topics that begin with a dollar sign. Unsupported publish or subscribe operations to reserved topics can result in a terminated connection.

Asset model topics

Topic	Client operations allowed	Description
$aws/sitewise/asset-models/*assetModelId*/assets/*assetId*/properties/*propertyId*	Subscribe	AWS IoT SiteWise publishes asset property notifications to this topic. For more information, see Interacting with other AWS services in the *AWS IoT SiteWise User Guide*.

Device Defender topics

These messages support response buffers in Concise Binary Object Representation (CBOR) format and JavaScript Object Notation (JSON), depending on the *payload-format* of the topic.

payload-format	Response format data type
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Object Notation (JSON)

For more information, see Sending metrics from devices (p. 859).

Topic	Allowed operations	Description
$aws/things/*thingName*/defender/metrics/*payload-format*	Publish	Device Defender agents publish metrics to this topic. For more information, see Sending metrics from devices (p. 859).

Topic	Allowed operations	Description
$aws/things/*thingName*/defender/metrics/*payload-format*/accepted	Subscribe	AWS IoT publishes to this topic after a Device Defender agent publishes a successful message to $aws/things/*thingName*/defender/metrics/*payload-format*. For more information, see Sending metrics from devices (p. 859).
$aws/things/*thingName*/defender/metrics/*payload-format*/rejected	Subscribe	AWS IoT publishes to this topic after a Device Defender agent publishes an unsuccessful message to $aws/things/*thingName*/defender/metrics/*payload-format*. For more information, see Sending metrics from devices (p. 859).

Event topics

Topic	Client operations allowed	Description
$aws/events/certificates/registered/*caCertificateId*	Subscribe	AWS IoT publishes this message when AWS IoT automatically registers a certificate and when a client presents a certificate with the `PENDING_ACTIVATION` status. For more information, see the section called "Configure the first connection by a client for automatic registration" (p. 214).
$aws/events/presence/connected/*clientId*	Subscribe	AWS IoT publishes to this topic when an MQTT client with the specified client ID connects to AWS IoT. For more information, see Connect/Disconnect events (p. 976).
$aws/events/presence/disconnected/*clientId*	Subscribe	AWS IoT publishes to this topic when an MQTT client with the specified client ID disconnects to AWS IoT. For more information, see Connect/Disconnect events (p. 976).

Topic	Client operations allowed	Description
$aws/events/subscriptions/subscribed/*clientId*	Subscribe	AWS IoT publishes to this topic when an MQTT client with the specified client ID subscribes to an MQTT topic. For more information, see Subscribe/Unsubscribe events (p. 978).
$aws/events/subscriptions/unsubscribed/*clientId*	Subscribe	AWS IoT publishes to this topic when an MQTT client with the specified client ID unsubscribes to an MQTT topic. For more information, see Subscribe/Unsubscribe events (p. 978).
$aws/events/thing/*thingName*/created	Subscribe	AWS IoT publishes to this topic when the *thingName* thing is created. For more information, see the section called "Registry events" (p. 966).
$aws/events/thing/*thingName*/updated	Subscribe	AWS IoT publishes to this topic when the *thingName* thing is updated. For more information, see the section called "Registry events" (p. 966).
$aws/events/thing/*thingName*/deleted	Subscribe	AWS IoT publishes to this topic when the *thingName* thing is deleted. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingGroup/*thingGroupName*/created	Subscribe	AWS IoT publishes to this topic when thing group, *thingGroupName*, is created. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingGroup/*thingGroupName*/updated	Subscribe	AWS IoT publishes to this topic when thing group, *thingGroupName*, is updated. For more information, see the section called "Registry events" (p. 966).

Topic	Client operations allowed	Description
$aws/events/thingGroup/*thingGroupName*/deleted	Subscribe	AWS IoT publishes to this topic when thing group, *thingGroupName*, is deleted. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingType/*thingTypeName*/created	Subscribe	AWS IoT publishes to this topic when the *thingTypeName* thing type is created. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingType/*thingTypeName*/updated	Subscribe	AWS IoT publishes to this topic when the *thingTypeName* thing type is updated. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingType/*thingTypeName*/deleted	Subscribe	AWS IoT publishes to this topic when the *thingTypeName* thing type is deleted. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingTypeAssociation/thing/*thingName*/*thingTypeName*	Subscribe	AWS IoT publishes to this topic when thing, *thingName*, is associated with or disassociated from thing type, *thingTypeName*. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/added	Subscribe	AWS IoT publishes to this topic when thing, *thingName*, is added to thing group, *thingGroupName*. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/removed	Subscribe	AWS IoT publishes to this topic when thing, *thingName*, is removed from thing group, *thingGroupName*. For more information, see the section called "Registry events" (p. 966).

Topic	Client operations allowed	Description
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/added	Subscribe	AWS IoT publishes to this topic when thing group, *childThingGroupName*, is added to thing group, *parentThingGroupName*. For more information, see the section called "Registry events" (p. 966).
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/removed	Subscribe	AWS IoT publishes to this topic when thing group, *childThingGroupName*, is removed from thing group, *parentThingGroupName*. For more information, see the section called "Registry events" (p. 966).

Fleet provisioning topics

These messages support response buffers in Concise Binary Object Representation (CBOR) format and JavaScript Object Notation (JSON), depending on the *payload-format* of the topic.

payload-format	Response format data type
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Object Notation (JSON)

For more information, see Device provisioning MQTT API (p. 678).

Topic	Client operations allowed	Description
$aws/certificates/create/*payload-format*	Publish	Publish to this topic to create a certificate from a certificate signing request (CSR).
$aws/certificates/create/*payload-format*/accepted	Subscribe	AWS IoT publishes to this topic after a successful call to $aws/certificates/create/*payload-format*.
$aws/certificates/create/*payload-format*/rejected	Subscribe	AWS IoT publishes to this topic after an unsuccessful call to $aws/certificates/create/*payload-format*.
$aws/certificates/create-from-csr/*payload-format*	Publish	Publishes to this topic to create a certificate from a CSR.
$aws/certificates/create-from-csr/*payload-format*/accepted	Subscribe	AWS IoT publishes to this topic a successful call to

Topic	Client operations allowed	Description
		$aws/certificates/create-from-csr/*payload-format*.
$aws/certificates/create-from-csr/*payload-format*/rejected	Subscribe	AWS IoT publishes to this topic an unsuccessful call to $aws/certificates/create-from-csr/*payload-format*.
$aws/events/presence/connected/*clientId*	Subscribe	AWS IoT publishes to this topic when an MQTT client with the specified client ID connects to AWS IoT. For more information, see Connect/Disconnect events (p. 976).
$aws/provisioning-templates/*templateName*/provision/*payload-format*	Publish	Publish to this topic to register a thing.
$aws/provisioning-templates/*templateName*/provision/*payload-format*/accepted	Subscribe	AWS IoT publishes to this topic after a successful call to $aws/provisioning-templates/*templateName*/provision/*payload-format*.
$aws/provisioning-templates/*templateName*/provision/*payload-format*/rejected	Subscribe	AWS IoT publishes to this topic after an unsuccessful call to $aws/provisioning-templates/*templateName*/provision/*payload-format*.

Job topics

Note

The client operations noted as **Receive** in this table indicate topics that AWS IoT publishes directly to the client that requested it, whether the client has subscribed to the topic or not. Clients should also expect to receive these response messages even if they have not subscribed to them.

These response messages do not pass through the message broker and they cannot be subscribed to by other clients or rules. To subscribe to job activity related messages, use the `notify` and `notify-next` topics.

For more information, see Jobs device MQTT and HTTPS APIs (p. 616).

Topic	Client operations allowed	Description
$aws/things/*thingName*/jobs/get	Publish	Devices publish a message to this topic to make a `GetPendingJobExecutions` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/get/accepted	Subscribe, Receive	Devices subscribe to this topic to receive

Topic	Client operations allowed	Description
		successful responses from a `GetPendingJobExecutions` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/get/rejected	Subscribe, Receive	Devices subscribe to this topic when a `GetPendingJobExecutions` request is rejected. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/start-next	Publish	Devices publish a message to this topic to make a `StartNextPendingJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/start-next/accepted	Subscribe, Receive	Devices subscribe to this topic to receive successful responses to a `StartNextPendingJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/start-next/rejected	Subscribe, Receive	Devices subscribe to this topic when a `StartNextPendingJobExecution` request is rejected. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/*jobId*/get	Publish	Devices publish a message to this topic to make a `DescribeJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/*jobId*/get/accepted	Subscribe, Receive	Devices subscribe to this topic to receive successful responses to a `DescribeJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).

Topic	Client operations allowed	Description
$aws/things/*thingName*/jobs/*jobId*/get/rejected	Subscribe, Receive	Devices subscribe to this topic when a `DescribeJobExecution` request is rejected. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/*jobId*/update	Publish	Devices publish a message to this topic to make an `UpdateJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/*jobId*/update/accepted	Subscribe, Receive	Devices subscribe to this topic to receive successful responses to an `UpdateJobExecution` request. For more information, see Jobs device MQTT and HTTPS APIs (p. 616). **Note** Only the device that publishes to $aws/things/*thingName*/jobs/*jobId*/update receives messages on this topic.
$aws/things/*thingName*/jobs/*jobId*/update/rejected	Subscribe, Receive	Devices subscribe to this topic when an `UpdateJobExecution` request is rejected. For more information, see Jobs device MQTT and HTTPS APIs (p. 616). **Note** Only the device that publishes to $aws/things/*thingName*/jobs/*jobId*/update receives messages on this topic.

Topic	Client operations allowed	Description
$aws/things/*thingName*/jobs/notify	Subscribe	Devices subscribe to this topic to receive notifications when a job execution is added or removed to the list of pending executions for a thing. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/things/*thingName*/jobs/notify-next	Subscribe	Devices subscribe to this topic to receive notifications when the next pending job execution for the thing is changed. For more information, see Jobs device MQTT and HTTPS APIs (p. 616).
$aws/events/job/*jobId*/completed	Subscribe	The Jobs service publishes an event on this topic when a job completes. For more information, see Jobs events (p. 973).
$aws/events/job/*jobId*/canceled	Subscribe	The Jobs service publishes an event on this topic when a job is canceled. For more information, see Jobs events (p. 973).
$aws/events/job/*jobId*/deleted	Subscribe	The Jobs service publishes an event on this topic when a job is deleted. For more information, see Jobs events (p. 973).
$aws/events/job/*jobId*/cancellation_in_progress	Subscribe	The Jobs service publishes an event on this topic when a job cancellation begins. For more information, see Jobs events (p. 973).
$aws/events/job/*jobId*/deletion_in_progress	Subscribe	The Jobs service publishes an event on this topic when a job deletion begins. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/succeeded	Subscribe	The Jobs service publishes an event on this topic when job execution succeeds. For more information, see Jobs events (p. 973).

Topic	Client operations allowed	Description
$aws/events/jobExecution/*jobId*/failed	Subscribe	The Jobs service publishes an event on this topic when a job execution fails. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/rejected	Subscribe	The Jobs service publishes an event on this topic when a job execution is rejected. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/canceled	Subscribe	The Jobs service publishes an event on this topic when a job execution is canceled. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/timed_out	Subscribe	The Jobs service publishes an event on this topic when a job execution times out. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/removed	Subscribe	The Jobs service publishes an event on this topic when a job execution is removed. For more information, see Jobs events (p. 973).
$aws/events/jobExecution/*jobId*/deleted	Subscribe	The Jobs service publishes an event on this topic when a job execution is deleted. For more information, see Jobs events (p. 973).

Rule topics

Topic	Client operations allowed	Description
$aws/rules/*ruleName*	Publish	A device or an application publishes to this topic to trigger rules directly. For more information, see Reducing messaging costs with basic ingest (p. 429).

Secure tunneling topics

Topic	Client operations allowed	Description
$aws/things/*thing-name*/tunnels/notify	Subscribe	AWS IoT publishes this message for an IoT agent

Topic	Client operations allowed	Description
		to start a local proxy on the remote device. For more information, see the section called "IoT agent snippet" (p. 656).

Shadow topics

The topics in this section are used by named and unnamed shadows. The topics used by each differ only in the topic prefix. This table shows the topic prefix used by each shadow type.

ShadowTopicPrefix **value**	**Shadow type**
$aws/things/*thingName*/shadow	Unnamed (classic) shadow
$aws/things/*thingName*/shadow/name/*shadowName*	Named shadow

To create a complete topic, select the *ShadowTopicPrefix* for the type of shadow to which you want to refer, replace *thingName* and if applicable, *shadowName*, with their corresponding values, and then append that with the topic stub as shown in the following table. Remember that topics are case sensitive.

Topic	Client operations allowed	Description
ShadowTopicPrefix/delete	Publish/Subscribe	A device or an application publishes to this topic to delete a shadow. For more information, see /delete (p. 525).
ShadowTopicPrefix/delete/accepted	Subscribe	The Device Shadow service sends messages to this topic when a shadow is deleted. For more information, see /delete/accepted (p. 525).
ShadowTopicPrefix/delete/rejected	Subscribe	The Device Shadow service sends messages to this topic when a request to delete a shadow is rejected. For more information, see /delete/rejected (p. 526).
ShadowTopicPrefix/get	Publish/Subscribe	An application or a thing publishes an empty message to this topic to get a shadow. For more information, see Device Shadow MQTT topics (p. 519).
ShadowTopicPrefix/get/accepted	Subscribe	The Device Shadow service sends messages to this topic

Topic	Client operations allowed	Description
		when a request for a shadow is made successfully. For more information, see /get/accepted (p. 520).
ShadowTopicPrefix/get/rejected	Subscribe	The Device Shadow service sends messages to this topic when a request for a shadow is rejected. For more information, see /get/rejected (p. 521).
ShadowTopicPrefix/update	Publish/Subscribe	A thing or application publishes to this topic to update a shadow. For more information, see /update (p. 521).
ShadowTopicPrefix/update/accepted	Subscribe	The Device Shadow service sends messages to this topic when an update is successfully made to a shadow. For more information, see /update/accepted (p. 523).
ShadowTopicPrefix/update/rejected	Subscribe	The Device Shadow service sends messages to this topic when an update to a shadow is rejected. For more information, see /update/rejected (p. 524).
ShadowTopicPrefix/update/delta	Subscribe	The Device Shadow service sends messages to this topic when a difference is detected between the reported and desired sections of a shadow. For more information, see /update/delta (p. 522).
ShadowTopicPrefix/update/documents	Subscribe	AWS IoT publishes a state document to this topic whenever an update to the shadow is successfully performed. For more information, see /update/documents (p. 524).

Streaming service topics

These messages support response buffers in Concise Binary Object Representation (CBOR) format and JavaScript Object Notation (JSON), depending on the *payload-format* of the topic.

payload-format	**Response format data type**
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Object Notation (JSON)

Topic	**Client operations allowed**	**Description**
$aws/things/*ThingName*/streams/*StreamId*/data/*payload-format*	Subscribe	The AWS Streaming service publishes to this topic if the "GetStream" request from a device is accepted. The payload contains the stream data. For more information, see Using the AWS IoT Streaming service in devices (p. 706).
$aws/things/*ThingName*/streams/*StreamId*/get/*payload-format*	Publish	A device publishes to this topic to perform a "GetStream" request. For more information, see Using the AWS IoT Streaming service in devices (p. 706).
$aws/things/*ThingName*/streams/*StreamId*/description/*payload-format*	Subscribe	The AWS Streaming service publishes to this topic if the "DescribeStream" request from a device is accepted. The payload contains the stream description. For more information, see Using the AWS IoT Streaming service in devices (p. 706).
$aws/things/*ThingName*/streams/*StreamId*/describe/*payload-format*	Publish	A device publishes to this topic to perform a "DescribeStream" request. For more information, see Using the AWS IoT Streaming service in devices (p. 706).
$aws/things/*ThingName*/streams/*StreamId*/rejected/*payload-format*	Subscribe	The AWS Streaming service publishes to this topic if a "DescribeStream" or "GetStream" request from a device is rejected. For more information, see Using the AWS IoT Streaming service in devices (p. 706).

Reserved topic ARN

All reserved topic ARNs (Amazon Resource Names) have the following form:

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

For example, `arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted` is an ARN for the reserved topic `$aws/things/thingName/jobs/get/accepted`.

Configurable endpoints (beta)

This feature is currently in public beta and is available only in the US East (N. Virginia) Region.

You can create multiple endpoints for connecting devices to AWS IoT. You can also configure some details (such as domain name) by using domain configurations. You can use domain configurations to simplify tasks such as the following.

- Migrate devices to AWS IoT.
- Support heterogeneous device fleets by maintaining separate domain configurations for separate device types.
- Maintain brand identity (for example, through domain name) while migrating application infrastructure to AWS IoT.

In the beta release, you can configure a fully qualified domain name (FQDN) and the associated server certificate too. You can also associate a custom authorizer. For more information, see Custom authentication (p. 222).

Note
AWS IoT uses the server name indication (SNI) TLS extension to apply domain configurations. Devices must use this extension when connecting and pass a server name that is identical to the domain name that is specified in the domain configuration. To test this service, use the v2 version of each AWS IoT Device SDK in GitHub.

Topics

Creating and configuring AWS-managed domains

This feature is currently in public beta and is available only in the US East (N. Virginia) Region.

You create a configurable endpoint on an AWS-managed domain by using the CreateDomainConfiguration API. A domain configuration for an AWS-managed domain consists of the following:

- `domainConfigurationName` – A user-defined name that identifies the domain configuration.

  Note
  Domain configuration names that start with `IoT:` are reserved for default endpoints and can't be used. Also, this value must be unique to your Region.
- `defaultAuthorizerName` – The name of the custom authorizer to use on the endpoint.
- `allowAuthorizerOverride` – A Boolean value that specifies whether devices can override the default authorizer by specifying a different authorizer in the HTTP header of the request. This value is required if a value for `defaultAuthorizerName` is specified.
- `serviceType` – Possible values are `DATA`, `CREDENTIAL_PROVIDER`, and `JOB`. If you specify `DATA`, AWS IoT returns an endpoint with an endpoint type of `iot:Data-Beta`. This is a special endpoint type for the configurable endpoints beta release. The endpoint serves ATS-signed server certificates. You can't create a configurable `iot:Data` (VeriSign) endpoint.

Note
AWS IoT currently supports only the DATA service type.

The following AWS CLI command creates domain configuration for a Data endpoint.

```
aws iot create-domain-configuration --domain-configuration-name "myDomainConfigurationName"
 --service-type "DATA"
```

Creating and configuring custom domains

This feature is currently in public beta and is available only in the US East (N. Virginia) Region.

Domain configurations let you specify a custom fully qualified domain name (FQDN) to connect to AWS IoT. Custom domains enable you to manage your own server certificates so that you can manage details, such as the root certificate authority (CA) used to sign the certificate, the signature algorithm, the certificate chain depth, and the lifecycle of the certificate.

The workflow to set up a domain configuration with a custom domain consists of the following three stages.

1. Registering Server Certificates in AWS Certificate Manager (p. 101)
2. Creating a Domain Configuration (p. 102)
3. Creating DNS Records (p. 103)

Registering server certificates in AWS certificate manager

Before you create a domain configuration with a custom domain, you must register your server certificate chain in AWS Certificate Manager (ACM). You can use three types of server certificates.

- ACM Generated Public Certificates (p. 101)
- External Certificates Signed by a Public CA (p. 102)
- External Certificates Signed by a Private CA (p. 102)

Note
AWS IoT considers a certificate to be signed by a public CA if it's included in Mozilla's trusted ca-bundle.

Using one certificate for multiple domains

If you plan to use one certificate to cover multiple subdomains, use a wildcard domain in the common name (CN) or Subject Alternative Names (SAN) field. For example, use **`*.iot.example.com`** to cover dev.iot.example.com, qa.iot.example.com, and prod.iot.example.com. Each FQDN requires its own domain configuration, but more than one domain configuration can use the same wildcard value. Either the CN or the SAN must cover the FQDN that you want to use as a custom domain. This coverage can be an exact match or a wildcard match.

The following sections describe how to get each type of certificate. Every certificate resource requires an Amazon Resource Name (ARN) registered with ACM that you use when you create your domain configuration.

ACM-generated public certificates

You can generate a public certificate for your custom domain by using the RequestCertificate API. When you generate a certificate in this way, ACM validates your ownership of the custom domain. For more information, see Request a Public Certificate in the *AWS Certificate Manager User Guide*.

External certificates signed by a public CA

If you already have a server certificate that is signed by a public CA (a CA that is included in Mozilla's trusted ca-bundle), you can import the certificate chain directly into ACM by using the ImportCertificate API. To learn more about this task and the prerequisites and certificate format requirements, see Importing Certificates.

External certificates signed by a private CA

If you already have a server certificate that is signed by a private CA or self-signed, you can use the certificate to create your domain configuration, but you also must create an extra public certificate in ACM to validate ownership of your domain. To do this, register your server certificate chain in ACM using the ImportCertificate API. To learn more about this task and the prerequisites and certificate format requirements, see Importing Certificates.

After you import your certificate to ACM, generate a public certificate for your custom domain by using the RequestCertificate API. When you generate a certificate in this way, ACM validates your ownership of the custom domain. For more information, see Request a Public Certificate. When you create your domain configuration, use this public certificate as your validation certificate.

Creating a domain configuration

You create a configurable endpoint on a custom domain by using the CreateDomainConfiguration API. A domain configuration for a custom domain consists of the following:

- `domainConfigurationName` – A user-defined name that identifies the domain configuration.

  Note
  Domain configuration names starting with `IoT:` are reserved for default endpoints and can't be used. Also, this value must be unique to your Region.
- `domainName` – The FQDN that your devices use to connect to AWS IoT.

  Note
  AWS IoT leverages the server name indication (SNI) TLS extension to apply domain configurations. Devices must use this extension when connecting and pass a server name that is identical to the domain name that is specified in the domain configuration.
- `serverCertificateArns` – The ARN of the server certificate chain that you registered with ACM. The beta release supports only one server certificate.
- `validationCertificateArn` – The ARN of the public certificate that you generated in ACM to validate ownership of your custom domain. This argument isn't required if you use a publicly signed or ACM-generated server certificate.
- `defaultAuthorizerName` – The name of the custom authorizer to use on the endpoint.
- `allowAuthorizerOverride` – A Boolean value that specifies whether devices can override the default authorizer by specifying a different authorizer in the HTTP header of the request. This value is required if a value for `defaultAuthorizerName` is specified.
- `serviceType` – Possible values are `DATA`, `CREDENTIAL_PROVIDER`, and `JOB`. If you specify `DATA`, AWS IoT returns an endpoint with an endpoint type of `iot:Data-Beta`. This is a special endpoint type for the configurable endpoints beta release. You can't create a configurable `iot:Data` (VeriSign) endpoint.

  Note
  AWS IoT currently supports only the `DATA` service type.

The following AWS CLI command creates a domain configuration for **iot.example.com**.

```
aws iot create-domain-configuration --domain-configuration-name "myDomainConfigurationName"
 --service-type "DATA"
```

```
--domain-name "iot.example.com" --server-certificate-arns serverCertARN --validation-
certificate-arn validationCertArn
```

Note
After you create your domain configuration, it might take up to 15 minutes until AWS IoT serves your custom server certificates.

Creating DNS records

After you register your server certificate chain and create your domain configuration, create a DNS record so that your custom domain points to an AWS IoT domain. This record must point to an AWS IoT endpoint of type `iot:Data-Beta`. This is a special endpoint type for the configurable endpoints beta release. You can get your beta endpoint by using the DescribeEndpoint API.

The following AWS CLI command shows how to get your beta endpoint.

```
aws iot describe-endpoint --endpoint-type iot:Data-Beta
```

After you get your `iot:Data-Beta` endpoint, create a `CNAME` record from your custom domain to this AWS IoT endpoint. If you create multiple custom domains in the same account, alias them to this same `iot:Data-Beta` endpoint.

Managing domain configurations

This feature is currently in public beta and is available only in the US East (N. Virginia) Region.

You can manage the lifecycles of existing configurations by using the following APIs.

- ListDomainConfigurations
- DescribeDomainConfiguration
- UpdateDomainConfiguration
- DeleteDomainConfiguration

Viewing domain configurations

Use the ListDomainConfigurations API to return a paginated list of all domain configurations in your account. You can see the details of a particular domain configuration using the DescribeDomainConfiguration API. This API takes a single `domainConfigurationName` parameter and returns the details of the specified configuration.

Updating domain configurations

To update the status or the custom authorizer of your domain configuration, use the UpdateDomainConfiguration API. You can set the status to `ENABLED` or `DISABLED`. If you disable the domain configuration, devices connected to that domain receive an authentication error.

Note
Currently you can't update the server certificate in your domain configuration. To change the certificate of a domain configuration, you must delete and recreate it.

Deleting domain configurations

Before you delete a domain configuration, use the UpdateDomainConfiguration API to set the status to `DISABLED`. This helps you avoid accidentally deleting the endpoint. After you disable the domain configuration, delete it by using the DeleteDomainConfiguration API.

After you delete a domain configuration, AWS IoT no longer serves the server certificate associated with that custom domain.

Connecting devices and gateways to AWS IoT Core for LoRaWAN

Long range WAN (LoRaWAN) devices and gateways can connect to AWS IoT Core by using AWS IoT Core for LoRaWAN. The LoRa Alliance describes LoRaWAN as, *"a Low Power, Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated 'things' to the internet in regional, national or global networks, and targets key Internet of Things (IoT) requirements such as bi-directional communication, end-to-end security, mobility and localization services."*

LoRaWAN devices communicate with AWS IoT Core through LoRaWAN gateways. AWS IoT rules send LoRaWAN device messages to other AWS services and can process the device messages to format the data for the services.

AWS IoT Core for LoRaWAN manages the service and device policies that AWS IoT Core requires to manage and communicate with the LoRaWAN gateways and devices. AWS IoT Core for LoRaWAN also manages the destinations that describe the AWS IoT rules that send device data to other services.

To get started using AWS IoT Core for LoRaWAN

- **Select the wireless devices and LoRaWAN gateways that you'll need**

  The AWS Partner Device Catalog contains gateways and developer kits that are qualified for use with AWS IoT Core for LoRaWAN.
- **Add your wireless devices and LoRaWAN gateways to AWS IoT Core for LoRaWAN**

  Adding LoRaWAN gateways and devices (p. 104) describes how to add your wireless devices and LoRaWAN gateways to AWS IoT Core for LoRaWAN. You'll also learn how to configure the other AWS IoT Core for LoRaWAN resources that you'll need to manage these devices and send their data to AWS services.
- **Complete your AWS IoT Core for LoRaWAN solution**

  Start with our sample AWS IoT Core for LoRaWAN solution and make it yours.

In this section

Adding LoRaWAN gateways and devices

If you're using AWS IoT Core for LoRaWAN for the first time, you can add your first LoRaWAN gateway and device by using the AWS IoT Core for LoRaWAN Intro page of the AWS IoT console and choosing **Get started**.

Whether you use the console (p. 105) or use the API (p. 114) to add your AWS IoT Core for LoRaWAN resources, consider the following topics before you get started. Adding the resources can be easier when you have the following information ready before you start.

1. **The naming conventions for your devices, gateways, profiles, and destinations**

   AWS IoT Core for LoRaWAN assigns unique IDs to the resources you create for wireless devices, gateways, and profiles; however, you can also give your resources more descriptive names to make it easier to identify them. Before you add devices, gateways, profiles, and destinations to AWS IoT Core for LoRaWAN, consider how you'll name them to make them easier to manage.

You can also add tags to the resources you create. Before you add your LoRaWAN devices, consider how you might use tags to identify and manage your AWS IoT Core for LoRaWAN resources. Tags can be modified after you add them.

For more information about naming and tagging, see Describe your AWS IoT Core for LoRaWAN resources (p. 106).

2. **The configuration parameters of the wireless devices**

   Some wireless device configuration parameters are shared by many devices and can be stored in AWS IoT Core for LoRaWAN as device and service profiles. Collecting these parameters in advance can make it easier to identify and enter them.

   - Wireless device configuration parameters include: the device's EUI, application and session security keys, and device profile settings such as data rate and channel information.
   - Wireless gateway configuration parameters include: the gateway's EUI and its LoRa frequency band.
   - Refer to the documentation about each device that its vendor provides for the complete listing of its specifications and configuration parameters.

   Having the configuration parameters that are unique to each device ready to enter in advance makes entering the data into the console go more smoothly. The specific parameters that you need to enter depend on the LoRaWAN specification that the device uses.

3. **The configuration parameters of the LoRaWAN gateways**

   Having the configuration parameters that are unique to each gateway ready to enter in advance makes entering the data into the console go more smoothly.

4. **The mapping of the device data to service data**

   The data from LoRaWAN wireless devices is often encoded to optimize bandwidth. These encoded messages arrive at AWS IoT Core for LoRaWAN in a format that might not be easily used by other AWS services. AWS IoT Core for LoRaWAN uses AWS IoT rules that can use AWS Lambda functions to process and decode the device messages to a format that other AWS services can use.

   To transform device data and send it to other AWS services, you need to know:

   - The format and contents of the data that the wireless devices send.
   - The service to which you want to send the data.
   - The format that service requires.

   Using that information, you can create the AWS IoT rule that performs the conversion and sends the converted data to the AWS services that will use it.

Using the console to add AWS IoT Core for LoRaWAN resources

This section provides additional detail about the information you need to add and modify AWS IoT Core for LoRaWAN resources by using the AWS IoT Core for LoRaWAN section of the AWS IoT console. Much of the data that you enter when configuring AWS IoT Core for LoRaWAN resources is provided by the devices' vendors and is specific to the LoRaWAN specifications they support.

The console interface is most practical when managing a few AWS IoT Core for LoRaWAN resources at a time. When managing large numbers of AWS IoT Core for LoRaWAN resources, consider creating more automated solutions by using the AWS IoT Wireless API (p. 114).

Topics

Describe your AWS IoT Core for LoRaWAN resources

AWS IoT Core for LoRaWAN provides several options to identify the resources you create. While AWS IoT Core for LoRaWAN resources are given a unique ID when they're created, this ID is not descriptive nor can it be changed after the resource is created. You can also assign a name, add a description, and attach tags and tag values to most AWS IoT Core for LoRaWAN resources to make it more convenient to select, identify, and manage your AWS IoT Core for LoRaWAN resources.

- **Resource names (p. 106)**

  For gateways, devices, and profiles, the resource name is an optional field that you can change after the resource is created. The name appears in the lists displayed on the resource hub pages.

  For destinations, you provide a name that is unique in your AWS account and AWS Region. You can't change the destination name after you create the destination resource.

  While a name can have up to 256 characters, the display space in the resource hub is limited. Make sure that the distinguishing part of the name appears in the first 20 to 30 characters, if possible.
- **Resource tags (p. 107)**

  Tags are key-value pairs of metadata that can be attached to AWS resources. You choose both tag keys and their corresponding values.

  Gateways, destinations, and profiles can have up to 50 tags attached to them. Devices don't support tags.

Resource names

AWS IoT Core for LoRaWAN resource support for name

Resource	Name field support	
Destination	Name is unique ID of resource and can't be changed.	
Device	Name is optional descriptor of resource and can be changed.	
Gateway	Name is optional descriptor of resource and can be changed.	
Profile	Name is optional descriptor of resource and can be changed.	

The name field appears in resource hub lists of resources; however, the space is limited and so only the first 15-30 characters of the name might be visible.

When selecting names for your resources, consider how you want them to identify the resources and how they'll be displayed in the console.

Description

Destination, device, and gateway resources also support a description field, which can accept up to 2,048 characters. The description field appears only in the individual resource's detail page. While the description field can hold a lot of information, because it only appears in the resource's detail page, it isn't convenient for scanning in the context of multiple resources.

Resource tags

AWS IoT Core for LoRaWAN resource support for AWS tags

Resource	AWS tag support	
Destination	Up to 50 AWS tags can be added to the resource.	
Device	This resource doesn't support AWS tags.	
Gateway	Up to 50 AWS tags can be added to the resource.	
Profile	Up to 50 AWS tags can be added to the resource.	

Tags are words or phrases that act as metadata that you can use to identify and organize your AWS resources. You can think of the tag key as a category of information and the tag value as a specific value in that category.

For example, you might have a tag value of *color* and then give some resources a value of *blue* for that tag and others a value of *red*. With that, you could use the Tag editor in the AWS console to find the resources with a *color* tag value of *blue*.

For more information about tagging and tagging strategies, see Tag editor.

Add your gateways and wireless devices to AWS IoT Core for LoRaWAN

LoRaWAN gateways connect wireless devices to AWS IoT Core for LoRaWAN. The data from wireless devices must be processed by an AWS IoT rule before it can be used by AWS IoT and other services. Adding a gateway to AWS IoT Core for LoRaWAN lets AWS IoT communicate with and manage the gateway. Adding devices to AWS IoT Core for LoRaWAN lets AWS IoT process the messages received from the devices for use by AWS IoT and other services.

Add your gateways to AWS IoT Core for LoRaWAN

Adding a gateway to AWS IoT Core for LoRaWAN so that AWS IoT can communicate with and manage the gateway requires the following information.

- **Device configuration data**

  These parameters are found on the gateway or in the gateway's documentation that its vendor provides.

  - **Gateway EUI**

    The EUI of the individual gateway device.
  - **Frequency band (RFRegion)**

    The gateway's frequency band.
- **Your wireless system configuration data**

  The information in these optional fields comes from how you organize and describe the elements in your wireless system. For more information on naming and describing your resources, see Describe your AWS IoT Core for LoRaWAN resources (p. 106).
  - **Name**

    The name you can assign to the gateway.
  - **Description**

    Information about the gateway.
  - **Tags**

    Key/value pairs of metadata about the gateway.
- **Thing association**

  Whether to create an AWS IoT thing and associate it with the gateway. Associating a thing with your gateway lets the gateway access other AWS IoT Core features.
- **An IAM role that allows the Configuration and Update Server (CUPS) to manage gateway credentials**

  You need to add an IAM role that allows the Configuration and Update Server (CUPS) to manage gateway credentials. You must do this before a LoRaWAN gateway tries to connect with AWS IoT Core for LoRaWAN; however, you need to do it only once.

  For more information about adding the IAM role, see Add an IAM role to allow the Configuration and Update Server (CUPS) to manage gateway credentials (p. 109).
- **Gateway device documentation**

  After adding the gateway's information to AWS IoT Core for LoRaWAN, add some AWS IoT Core for LoRaWAN information to the gateway device. The documentation provided by the gateway's vendor should describe the process for uploading the certificate files to the gateway and configuring the gateway device to communicate with AWS IoT Core for LoRaWAN.

Add your wireless devices to AWS IoT Core for LoRaWAN

Before you add a wireless device to AWS IoT Core for LoRaWAN so that AWS IoT can process data from the device for use by AWS IoT and other services, make sure you have the following information.

- **LoRaWAN specification and wireless device configuration**

  The device's LoRaWAN specification is provided by the device's vendor in its documentation or on the device itself. The device's configuration parameters depend on the LoRaWAN specification it uses and are also found in the device's documentation or on the device itself.
- **Device name and description**

  The information in these optional fields comes from how you organize and describe the elements in your wireless system. For more information on naming and describing your resources, see Describe your AWS IoT Core for LoRaWAN resources (p. 106).
- **Thing association**

Whether to create an AWS IoT thing and associate it with the device. Associating a thing with your gateway lets the gateway access other AWS IoT Core features.

- **Device and service profiles**

  The device's configuration parameters depend on the LoRaWAN specification it uses. The configuration parameters are found in the device's documentation or on the device itself. You'll want to identify a device profile that matches the configuration parameters of the device, or create one if necessary, before you add the device.
- **AWS IoT Core for LoRaWAN destination**

  Each device must be assigned to a destination that will process its messages to send to AWS IoT and other services. The AWS IoT rules that process and send the device messages are specific to the device's message format. To process the messages from the device and send them to the correct service, identify the destination you created to use with the device's messages and assign it to the device.

  If you don't have a destination defined for this device, you'll need to add it before you add the device.

Add an IAM role to allow the Configuration and Update Server (CUPS) to manage gateway credentials

This procedure describes how to add an IAM role that will allow the Configuration and Update Server (CUPS) to manage gateway credentials. Make sure you perform this procedure before a LoRaWAN gateway tries to connect with AWS IoT Core for LoRaWAN; however, you need to do this only once.

To add the IAM role to allow the Configuration and Update Server (CUPS) to manage gateway credentials:

1. Open the Roles hub of the IAM console and choose **Create role**.
2. If you think that you might have already added the **IoTWirelessGatewayCertManagerRole** role, in the search bar, enter **`IoTWirelessGatewayCertManagerRole`**.

   If you see an **IoTWirelessGatewayCertManagerRole** role in the search results, you have the necessary IAM role. You can leave the procedure now.

   If the search results are empty, you don't have the necessary IAM role. Continue the procedure to add it.
3. In **Select type of trusted entity**, choose **Another AWS account**.
4. In **Account ID**, enter your AWS account ID, and then choose **Next: Permissions**.
5. In the search box, enter **`AWSIoTWirelessGatewayCertManager`**.
6. In the list of search results, select the policy named **AWSIoTWirelessGatewayCertManager**.
7. Choose **Next: Tags**, and then choose **Next: Review**.
8. In **Role name**, enter **`IoTWirelessGatewayCertManagerRole`**, and then choose **Create role**.
9. To edit the new role, in the confirmation message, choose **IoTWirelessGatewayCertManagerRole**.
10. In **Summary**, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
11. In **Policy Document**, change the `Principal` property to look like this example.

```
"Principal": {
    "Service": "iotwireless.amazonaws.com"
},
```

   After you change the `Principal` property, the complete policy document should look like this example.

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

12. To save your changes and exit, choose **Update Trust Policy**.

You've now created the **IoTWirelessGatewayCertManagerRole**. You won't need to do this again.

If you performed this procedure while you were adding a gateway, you can close this window and the IAM console and return to the AWS IoT console to finish adding the gateway.

Add profiles to AWS IoT Core for LoRaWAN

Device and service profiles can be defined to describe common device configurations. These profiles describe configuration parameters that are shared by devices to make it easier to add those devices. AWS IoT Core for LoRaWAN supports device profiles and service profiles.

The configuration parameters and the values to enter into these profiles are provided by the device's manufacturer.

Device profiles

Device profiles contain the communication and protocol parameter values the device needs to communicate with the network server. The parameters used in a profile depend on the protocol type, version, and supported options.

Service profiles

Service profiles describe the communication parameters the device needs to communicate with the application server.

Add destinations to AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN destinations describe the AWS IoT rule that processes a device's data for use by AWS services.

Because most LoRaWAN devices don't send data to AWS IoT Core for LoRaWAN in a format that can be used by AWS services, an AWS IoT rule must process it first. The AWS IoT rule contains the SQL statement that interprets the device's data and the topic rule actions that send the result of the SQL statement to the services that will use it.

To process a device's data, an AWS IoT Core for LoRaWAN destination contains the following elements.

- **Role name**

  The IAM role that gives the device's data permission to access the rule named in **Rule name**. For more information about the details that a definition requires in the role, see Create an IAM roles for your destinations (p. 111)

For more information about IAM roles, see Using IAM roles.

- **Rule name**

  The AWS IoT rule that is configured to process the device's data.

  For more information about AWS IoT rules for destinations, see Create rules to process LoRaWAN device messages (p. 112).

Create an IAM roles for your destinations

AWS IoT Core for LoRaWAN destinations require IAM roles that give AWS IoT Core for LoRaWAN the permissions necessary to send data to the AWS IoT rule. If such a role is not already defined, you'll need to define it so that it will appear in the list of roles.

To create an IAM policy for your AWS IoT Core for LoRaWAN destination role

1. Open the Policies hub of the IAM console.
2. Choose **Create policy**, and choose the **JSON** tab.
3. In the editor, delete any content from the editor and paste this policy document.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeEndpoint",
                "iot:Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Choose **Review policy**, and in **Name**, enter a name for this policy. You'll need this name to use in the next procedure.

   You can also describe this policy in **Description**, if you want.
5. Choose **Create policy**.

To create an IAM role for an AWS IoT Core for LoRaWAN destination

1. Open the Roles hub of the IAM console and choose **Create role**.
2. In **Select type of trusted entity**, choose **Another AWS account**.
3. In **Account ID**, enter your AWS account ID, and then choose **Next: Permissions**.
4. In the search box, enter the name of the IAM policy that you created in the previous procedure.
5. In the search results, check the IAM policy that you created in the previous procedure.
6. Choose **Next: Tags**, and then choose **Next: Review**.
7. In **Role name**, enter the name of this role, and then choose **Create role**.
8. In the confirmation message, choose the name of the role you created to edit the new role.
9. In **Summary**, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
10. In **Policy Document**, change the `Principal` property to look like this example.

```
"Principal": {
    "Service": "iotwireless.amazonaws.com"
},
```

After you change the `Principal` property, the complete policy document should look like this example.

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

11. To save your changes and exit, choose **Update Trust Policy**.

With this role defined, you can find it in the list of roles when you configure your AWS IoT Core for LoRaWAN destinations.

Create rules to process LoRaWAN device messages

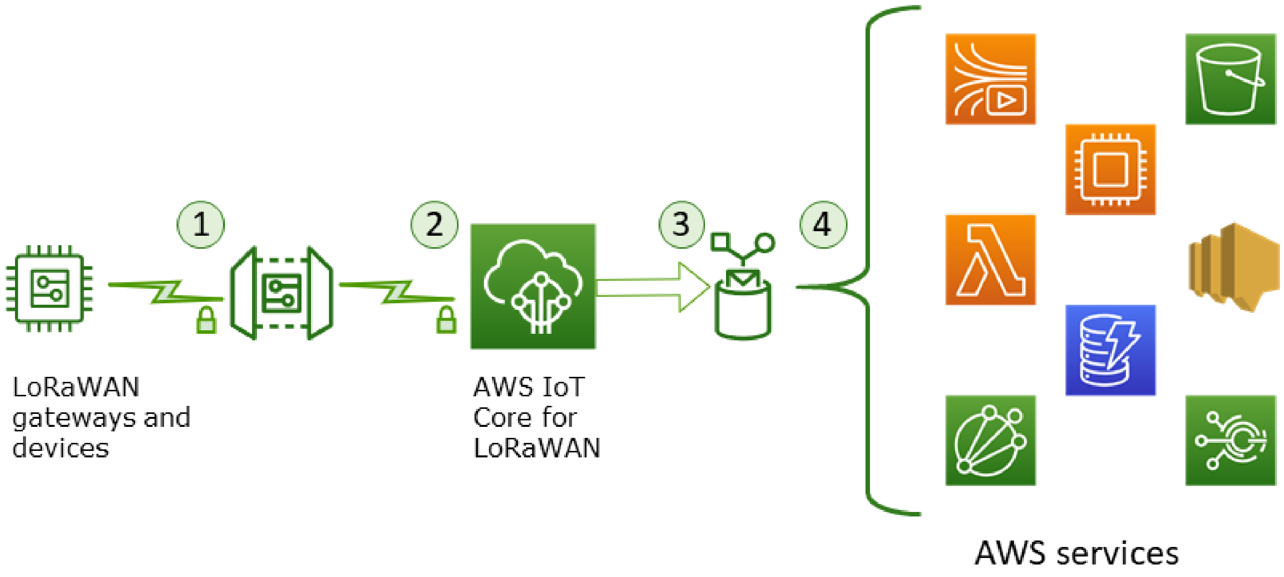
AWS IoT rules send device messages to other services. AWS IoT rules can also process the binary messages received from a LoRaWAN device to convert the messages to other formats that can make them easier for other services to use.

AWS IoT Core for LoRaWAN destinations (p. 110) associate a wireless device with the rule that processes the device's message data to send to other services. The rule acts on the device's data as soon as AWS IoT Core for LoRaWAN receives it. AWS IoT Core for LoRaWAN destinations (p. 110) can be shared by all devices whose messages have the same data format and that send their data to the same service.

How AWS IoT rules process device messages

How an AWS IoT rule processes a device's message data depends on the service that will receive the data, the format of the device's message data, and the data format that the service requires. Typically, the rule calls an AWS Lambda function to convert the device's message data to the format a service requires, and then sends the result to the service.

The following illustration shows how message data is secured and processed as it moves from the wireless device to an AWS service.



1. The LoRaWAN wireless device encrypts its binary messages using AES128 CTR mode before it transmits them.
2. AWS IoT Core for LoRaWAN decrypts the binary message and encodes the decrypted binary message payload as a base64 string.
3. The resulting base64-encoded message is sent as a binary message payload (a message payload that is not formatted as a JSON document) to the AWS IoT rule described in the destination assigned to the device.
4. The AWS IoT rule directs the message data to the service described in the rule's configuration.

The encrypted binary payload received from the wireless device is not altered or interpreted by AWS IoT Core for LoRaWAN. The decrypted binary message payload is encoded only as a base64 string. For services to access the data elements in the binary message payload, the data elements must be parsed out of the payload by a function called by the rule. The base64-encoded message payload is an ASCII string, so it could be stored as such to be parsed later.

Create rules for LoRaWAN

AWS IoT Core for LoRaWAN uses AWS IoT rules to securely send device messages directly to other AWS services without the need to use the message broker. By removing the message broker from the ingestion path, it reduces costs and optimizes the data flow.

For an AWS IoT Core for LoRaWAN rule to send device messages to other AWS services, it requires an AWS IoT Core for LoRaWAN destination and an AWS IoT rule assigned to that destination. The AWS IoT rule must contain a SQL query statement and at least one rule action.

Typically, the AWS IoT rule query statement consists of:

- A SQL SELECT clause that selects and formats the data from the message payload
- A topic filter (the FROM object in the rule query statement) that identifies the messages to use
- An optional conditional statement (a SQL WHERE clause) that specifies conditions on which to act

Here is an example of a rule query statement:

```
SELECT temperature FROM iot/topic' WHERE temperature > 50
```

When building AWS IoT rules to process payloads from LoRaWAN devices, you do not have to specify the FROM clause as part of the rule query object. The rule query statement must have the SQL SELECT clause and can optionally have the WHERE clause. If the query statement uses the FROM clause, it is ignored.

Here is an example of a rule query statement that can process payloads from LoRaWAN devices:

```
SELECT WirelessDeviceId, WirelessMetadata.LoRaWAN.FPort as FPort,
       WirelessMetadata.LoRaWAN.DevEui as DevEui,
       PayloadData
```

In this example, the `PayloadData` is a base64-encoded binary payload sent by your LoRaWAN device.

Here is an example rule query statement that can perform a binary decoding of the incoming payload and transform it into a different format such as JSON:

```
SELECT WirelessDeviceId, WirelessMetadata.LoRaWAN.FPort as FPort,
       WirelessMetadata.LoRaWAN.DevEui as DevEui,
       aws_lambda("arn:aws:lambda:<region>:<account>:function:<name>",

                  {
                   "PayloadData":PayloadData,
                   "Fport": WirelessMetadata.LoRaWAN.FPort
                  })
```

For more information on using the SELECT AND WHERE clauses, see AWS IoT SQL reference (p. 430)

For information about AWS IoT rules and how to create and use them, see Rules for AWS IoT (p. 352) and AWS IoT rules tutorials (p. 134).

For information about creating and using AWS IoT Core for LoRaWAN destinations, see Add destinations to AWS IoT Core for LoRaWAN (p. 110).

For information about using binary message payloads in a rule, see Working with binary payloads (p. 488).

For more information about the data security and encryption used to protect the message payload on its journey, see Data protection in AWS IoT Core (p. 274).

For a reference architecture that shows a binary decoding and implementation example for IoT rules, see AWS IoT Core for LoRaWAN Solution Samples on GitHub

Using the API to manage AWS IoT Core for LoRaWAN resources

The AWS IoT Wireless API provides the actions to create and manage AWS IoT Core for LoRaWAN resources.

The AWS SDK supports the AWS IoT Wireless API. The following lists describe the API actions that perform the tasks described in Using the console to add AWS IoT Core for LoRaWAN resources (p. 105).

AWS IoT Wireless API actions to create AWS IoT Core for LoRaWAN resources

- CreateDestination

- CreateDeviceProfile
- CreateServiceProfile
- CreateWirelessDevice
- CreateWirelessGateway

AWS IoT Wireless API actions to list AWS IoT Core for LoRaWAN resources

- ListDestinations
- ListDeviceProfiles
- ListServiceProfiles
- ListWirelessDevices
- ListWirelessGateways

AWS IoT Wireless API actions to update AWS IoT Core for LoRaWAN resources

- UpdateDestination
- UpdateWirelessDevice
- UpdateWirelessGateway

AWS IoT Wireless API actions to delete AWS IoT Core for LoRaWAN resources

- DeleteDestination
- DeleteDeviceProfile
- DeleteServiceProfile
- DeleteWirelessDevice
- DeleteWirelessGateway

For the complete list of the actions and data types available to create and manage AWS IoT Core for LoRaWAN resources, see the AWS IoT Wireless API reference.

Data security with AWS IoT Core for LoRaWAN

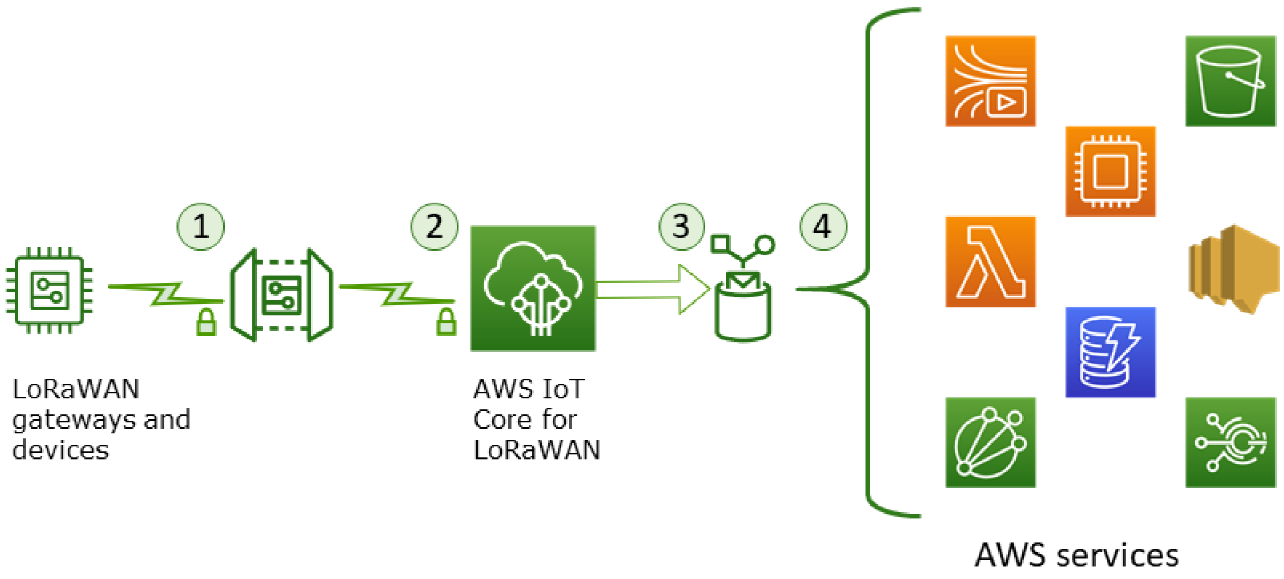
Two methods secure the data from your AWS IoT Core for LoRaWAN devices:

- **The security that wireless devices use to communicate with the gateways.**

  The LoRaWAN devices follow the security practices described in LoRaWAN ™ SECURITY: A White Paper Prepared for the LoRa Alliance™ by Gemalto, Actility, and Semtech.
- **The security that AWS IoT Core uses to connect gateways to AWS IoT Core for LoRaWAN and send the data to other AWS services.**

  AWS IoT Core security is described in Data protection in AWS IoT Core (p. 274).

This diagram identifies the key elements in a LoRaWAN system connected to AWS IoT Core for LoRaWAN to identify how data is secured throughout.



1. The LoRaWAN wireless device encrypts its binary messages using AES128 CTR mode before it transmits them.
2. Gateway connections to AWS IoT Core for LoRaWAN are secured by TLS as described in Transport security in AWS IoT (p. 275). AWS IoT Core for LoRaWAN decrypts the binary message and encodes the decrypted binary message payload as a base64 string.
3. The resulting base64-encoded message is sent as the message payload to the AWS IoT rule described in the destination assigned to the device. Data within AWS is encrypted using AWS-owned keys.
4. The AWS IoT rule directs the message data to the services described in the rule's configuration. Data within AWS is encrypted using AWS-owned keys.

LoRaWAN device and gateway transport security

LoRaWAN devices and AWS IoT Core for LoRaWAN store pre-shared root keys. Session keys are derived by both LoRaWAN devices and AWS IoT Core for LoRaWAN following the protocols. The symmetric session keys are used for encryption and decryption in a standard AES-128 CTR mode. A 4-byte message integrity code (MIC) is also used to check the data integrity following a standard AES-128 CMAC algorithm. The session keys can be updated by using the Join/Rejoin process.

The security practice for LoRa gateways is described in the LoRaWAN specifications. LoRa gateways connect to AWS IoT Core for LoRaWAN through a web socket using a `Basics Station`. AWS IoT Core for LoRaWAN supports only `Basics Station` version 2.0.4 and later.

Before the web socket connection is established, AWS IoT Core for LoRaWAN uses the TLS Server and Client Authentication mode (p. 275) to authenticate the gateway. AWS IoT Core for LoRaWAN also maintains a Configuration and Update Server (CUPS) that configures and updates the certificates and keys used for TLS authentication.

AWS IoT Tutorials

These tutorials can help you learn how to apply a specific aspect of AWS IoT to your solution.

Topics

Connect a device to AWS IoT Core by using the AWS IoT Device SDK

This tutorial demonstrates how to connect a device to AWS IoT Core so that it can send and receive data to and from AWS IoT. After you complete this tutorial, your device will be configured to connect to AWS IoT Core and you'll understand how devices communicate with AWS IoT.

In this tutorial, you'll:

1. the section called "Prepare your device for AWS IoT" (p. 117)
2. the section called "Review the MQTT protocol" (p. 118)
3. the section called "Review the pubsub.py Device SDK sample app" (p. 118)
4. the section called "Connect your device and communicate with AWS IoT Core" (p. 124)
5. the section called "Review the results" (p. 130)

This tutorial takes about an hour to complete.

Before you start this tutorial, make sure that you have:

- **Completed Getting started with AWS IoT Core (p. 17)**

  In the section of that tutorial where you must the section called "Configure your device" (p. 38), select the the section called "Connect a Raspberry Pi or another device" (p. 51) option for your device and use the Python language options to configure your device.

  Be sure to keep open the terminal window you use in that tutorial because you'll also use it in this tutorial.
- **A device that can run the AWS IoT Device SDK v2 for Python.**

  This tutorial shows how to connect a device to AWS IoT Core by using Python code examples, which require a relatively powerful device, as IoT and embedded devices go.

  If you are working with resource-constrained devices, these code examples might not work on them. In that case, you might have more success by the section called "Using the AWS IoT Device SDK for Embedded C" (p. 130) tutorial.

Prepare your device for AWS IoT

In Getting started with AWS IoT Core (p. 17), you prepared your device and AWS account so they could communicate. This section reviews the aspects of that preparation that apply to any device connection with AWS IoT Core.

For a device to connect to AWS IoT Core:

1. You must have an **AWS account**.

   The procedure in Set up your AWS account (p. 18) describes how to create an AWS account if you don't already have one.
2. In that account, you must have the following **AWS IoT resources** defined for the device in your AWS account and Region.

   The procedure in Create AWS IoT resources (p. 34) describes how to create these resources for the device in your AWS account and Region.
   - A **device certificate** registered with AWS IoT and activated to authenticate the device.

     The certificate is often created with, and attached to, an **AWS IoT thing object**. While a thing object is not required for a device to connect to AWS IoT, it makes additional AWS IoT features available to the device.
   - A **policy** attached to the device certificate that authorizes it to connect to AWS IoT Core and perform all the actions that you want it to.
3. An **internet connection** that can access your AWS account's device endpoints.

   The device endpoints are described in AWS IoT device data and service endpoints (p. 73) and can be seen in the settings page of the AWS IoT console.
4. **Communication software** such as the AWS IoT Device SDKs provide. This tutorial uses the AWS IoT Device SDK v2 for Python.

Review the MQTT protocol

Before we talk about the sample app, it helps to understand the MQTT protocol. The MQTT protocol offers some advantages over other network communication protocols, such as HTTP, which makes it a popular choice for IoT devices. This section reviews the key aspects of MQTT that apply to this tutorial. For information about how MQTT compares to HTTP, see Choosing a protocol for your device communication (p. 78).

MQTT uses a publish/subscribe communication model

The MQTT protocol uses a publish/subscribe communication model with its host. This model differs from the request/response model that HTTP uses. With MQTT, devices establish a session with the host that is identified by a unique client ID. To send data, devices publish messages identified by topics to a message broker in the host. To receive messages from the message broker, devices subscribe to the topics they will receive by sending topic filters in subscription requests to the message broker.

MQTT supports persistent sessions

The message broker receives messages from devices and publishes messages to devices that have subscribed to them. With persistent sessions (p. 80) —sessions that remain active even when the initiating device is disconnected—devices can retrieve messages that were published while they were disconnected. On the device side, MQTT supports Quality of Service levels (QoS (p. 80)) that ensure the host receives messages sent by the device.

Review the pubsub.py Device SDK sample app

This section reviews the `pubsub.py` sample app from the **AWS IoT Device SDK v2 for Python** used in this tutorial. Here, we'll review how it connects to AWS IoT Core to publish and subscribe to MQTT messages. The next section presents some exercises to help you explore how a device connects and communicates with AWS IoT Core.

The pubsub.py sample app demonstrates these aspects of an MQTT connection with AWS IoT Core:

- Communication protocols (p. 119)
- Persistent sessions (p. 121)
- Quality of Service (p. 122)
- Message publish (p. 122)
- Message subscription (p. 123)
- Device disconnection and reconnection (p. 124)

Communication protocols

The `pubsub.py` sample demonstrates an MQTT connection using the MQTT and MQTT over WSS protocols. The AWS common runtime (AWS-CRT) library provides the low-level communication protocol support and is included with the AWS IoT Device SDK v2 for Python.

MQTT

The `pubsub.py` sample calls `mtls_from_path` (shown here) in the `mqtt_connection_builder` to establish a connection with AWS IoT Core by using the MQTT protocol. `mtls_from_path` uses X.509 certificates and TLS v1.2 to authenticate the device. The AWS-CRT library handles the lower-level details of that connection.

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.root_ca,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

Your AWS account's IoT device endpoint

In the sample app, this value is passed in from the command line.

`cert_filepath`

The path to the device's certificate file

In the sample app, this value is passed in from the command line.

`pri_key_filepath`

The path to the device's private key file that was created with its certificate file

In the sample app, this value is passed in from the command line.

`ca_filepath`

The path to the Root CA file. Required only if the MQTT server uses a certificate that's not already in your trust store.

In the sample app, this value is passed in from the command line.

`client_bootstrap`

The common runtime object that handles socket communication activities

In the sample app, this object is instantiated just prior to the call to `mqtt_connection_builder.mtls_from_path`.

`on_connection_interrupted`, `on_connection_resumed`

The callback functions to call when the device's connection is interrupted and resumed

`client_id`

The ID that uniquely identifies this device in the AWS Region

In the sample app, this value is passed in from the command line.

`clean_session`

Whether to start a new persistent session, or, if one is present, reconnect to an existing one

`keep_alive_secs`

The keep alive value, in seconds, to send in the `CONNECT` request. A ping will automatically be sent at this interval. The server assumes the connection is lost if it doesn't receive a ping after 1.5 times this value.

MQTT over WSS

The `pubsub.py` sample calls `websockets_with_default_aws_signing` (shown here) in the `mqtt_connection_builder` to establish a connection with AWS IoT Core using the MQTT protocol over WSS. `websockets_with_default_aws_signing` creates an MQTT connection over WSS using Signature V4 to authenticate the device.

```
mqtt_connection = mqtt_connection_builder.websockets_with_default_aws_signing(
    endpoint=args.endpoint,
    client_bootstrap=client_bootstrap,
    region=args.signing_region,
    credentials_provider=credentials_provider,
    websocket_proxy_options=proxy_options,
    ca_filepath=args.root_ca,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

Your AWS account's IoT device endpoint

In the sample app, this value is passed in from the command line.

`client_bootstrap`

The common runtime object that handles socket communication activities

In the sample app, this object is instantiated just prior to the call to `mqtt_connection_builder.websockets_with_default_aws_signing`.

`region`

The AWS signing Region used by Signature V4 authentication. In `pubsub.py`, it passes the parameter entered in the command line.

In the sample app, this value is passed in from the command line.

`credentials_provider`

The AWS credentials provided to use for authentication

In the sample app, this object is instantiated just prior to the call to `mqtt_connection_builder.websockets_with_default_aws_signing`.

`websocket_proxy_options`

HTTP proxy options, if using a proxy host

In the sample app, this value is initialized just prior to the call to `mqtt_connection_builder.websockets_with_default_aws_signing`.

`ca_filepath`

The path to the Root CA file. Required only if the MQTT server uses a certificate that's not already in your trust store.

In the sample app, this value is passed in from the command line.

`on_connection_interrupted`, `on_connection_resumed`

The callback functions to call when the device's connection is interrupted and resumed

`client_id`

The ID that uniquely identifies this device in the AWS Region.

In the sample app, this value is passed in from the command line.

`clean_session`

Whether to start a new persistent session, or, if one is present, reconnect to an existing one

`keep_alive_secs`

The keep alive value, in seconds, to send in the `CONNECT` request. A ping will automatically be sent at this interval. The server assumes the connection is lost if it doesn't receive a ping after 1.5 times this value.

HTTPS

What about HTTPS? AWS IoT Core supports devices that publish HTTPS requests. From a programming perspective, devices send HTTPS requests to AWS IoT Core as would any other application. For an example of a Python program that sends an HTTP message from a device, see the HTTPS code example (p. 83) using Python's `requests` library. This example sends a message to AWS IoT Core using HTTPS such that AWS IoT Core interprets it as an MQTT message.

While AWS IoT Core supports HTTPS requests from devices, be sure to review the information about Choosing a protocol for your device communication (p. 78) so that you can make an informed decision on which protocol to use for your device communications.

Persistent sessions

In the sample app, setting the `clean_session` parameter to `False` indicates that the connection should be persistent. In practice, this means that the connection opened by this call reconnects to an existing persistent session, if one exists. Otherwise, it creates and connects to a new persistent session.

With a persistent session, messages that are sent to the device are stored by the message broker while the device is not connected. When a device reconnects to a persistent session, the message broker sends to the device any stored messages to which it has subscribed.

Without a persistent session, the device will not receive messages that are sent while the device isn't connected. Which option to use depends on your application and whether messages that occur while a device is not connected must be communicated. For more information, see Using MQTT persistent sessions (p. 80).

Quality of Service

When the device publishes and subscribes to messages, the preferred Quality of Service (QoS) can be set. AWS IoT supports QoS levels 0 and 1 for publish and subscribe operations. For more information about QoS levels in AWS IoT, see MQTT Quality of Service (QoS) options (p. 80).

The AWS-CRT runtime for Python defines these constants for the QoS levels that it supports:

Python Quality of Service levels

MQTT QoS level	Python symbolic value used by SDK	Description
QoS level 0	`mqtt.QoS.AT_MOST_ONCE`	Only one attempt to send the message will be made, whether it is received or not. The message might not be sent at all, for example, if the device is not connected or there's a network error.
QoS level 1	`mqtt.QoS.AT_LEAST_ONCE`	The message is sent repeatedly until a `PUBACK` acknowledgement is received.

In the sample app, the publish and subscribe requests are made with a QoS level of 1 (`mqtt.QoS.AT_LEAST_ONCE`).

- **QoS on publish**

  When a device publishes a message with QoS level 1, it sends the message repeatedly until it receives a `PUBACK` response from the message broker. If the device isn't connected, the message is queued to be sent after it reconnects.
- **QoS on subscribe**

  When a device subscribes to a message with QoS level 1, the message broker saves the messages to which the device is subscribed until they can be sent to the device. The message broker resends the messages until it receives a `PUBACK` response from the device.

Message publish

After successfully establishing a connection to AWS IoT Core, devices can publish messages. The `pubsub.py` sample does this by calling the `publish` method of the `mqtt_connection` object.

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

`topic`

The message's topic name that identifies the message

In the sample app, this is passed in from the command line.

`payload`

The message payload formatted as a string (for example, a JSON document)

In the sample app, this is passed in from the command line.

A JSON document is a common payload format, and one that is recognized by other AWS IoT services; however, the data format of the message payload can be anything that the publishers and subscribers agree upon. Other AWS IoT services, however, only recognize JSON, and CBOR, in some cases, for most operations.

`qos`

The QoS level for this message

Message subscription

To receive messages from AWS IoT and other services and devices, devices subscribe to those messages by their topic name. Devices can subscribe to individual messages by specifying a topic name (p. 85), and to a group of messages by specifying a topic filter (p. 86), which can include wild card characters. The `pubsub.py` sample uses the code shown here to subscribe to messages and register the callback functions to process the message after it's received.

```
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=args.topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received
)
subscribe_result = subscribe_future.result()
```

`topic`

The topic to subscribe to. This can be a topic name or a topic filter.

In the sample app, this is passed in from the command line.

`qos`

Whether the message broker should store these messages while the device is disconnected.

A value of `mqtt.QoS.AT_LEAST_ONCE` (QoS level 1), requires a persistent session to be specified (`clean_session=False`) when the connection is created.

`callback`

The function to call to process the subscribed message.

The `mqtt_connection.subscribe` function returns a future and a packet ID. If the subscription request was initiated successfully, the packet ID returned is greater than 0. To make sure the subscription was received and registered by the message broker, you must wait for the result of the asynchronous operation to return, as shown in the code example.

The callback function

The callback in the `pubsub.py` sample processes the subscribed messages as the device receives them.

```
def on_message_received(topic, payload, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == args.count:
        received_all_event.set()
```

`topic`

The message's topic

This is the specific topic name of the message received, even if you subscribed to a topic filter.

`payload`

The message payload

The format for this is application specific.

`kwargs`

Possible additional arguments as described in `mqtt.Connection.subscribe`.

In the `pubsub.py` sample, `on_message_received` only displays the topic and its payload. It also counts the messages received to end the program after the limit is reached.

Your app would evaluate the topic and the payload to determine what actions to perform.

Device disconnection and reconnection

The `pubsub.py` sample includes callback functions that are called when the device is disconnected and when the connection is re-established. What actions your device takes on these events is application specific.

When a device connects for the first time, it must subscribe to topics to receive. If a device's session is present when it reconnects, its subscriptions are restored, and any stored messages from those subscriptions are sent to the device after it reconnects.

If a device's session no longer exists when it reconnects, it must resubscribe to its subscriptions. Persistent sessions have a limited lifetime and can expire when the device is disconnected for too long.

Connect your device and communicate with AWS IoT Core

This section presents some exercises to help you explore different aspects of connecting your device to AWS IoT Core. For these exercises, you'll use the MQTT test client in the AWS IoT console to see what your device publishes and to publish messages to your device. These exercises use the `pubsub.py` sample from the AWS IoT Device SDK v2 for Python and build on your experience with the Getting started tutorials.

In this section, you'll:

- Subscribe to wild card topic filters (p. 125)
- Process topic filter subscriptions (p. 126)
- Publish messages from your device (p. 128)

For these exercises, you'll start from the `pubsub.py` sample program.

> **Note**
> These exercises assume that you completed the Getting started tutorials and use the terminal window for your device from that tutorial.

Subscribe to wild card topic filters

In this exercise, you'll modify the command line used to call `pubsub.py` to subscribe to a wild card topic filter and process the messages received based on the message's topic.

Exercise procedure

For this exercise, imagine that your device contains a temperature control and a light control. It uses these topic names to identify the messages about them.

1. Before starting the exercise, try running this command from the Getting started tutorials on your device to make sure everything is ready for the exercise.

```
cd ~/aws-iot-device-sdk-python-v2/samples
python3 pubsub.py --topic topic_1 --root-ca ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key —endpoint your-iot-endpoint
```

   You should see the same output as you saw in the Getting started tutorial (p. 56).
2. For this exercise, change these command line parameters.

Action	Command line parameter	Effect
add	**--message** `""`	Configure `pubsub.py` to listen only
add	**--count** `2`	End the program after receiving two messages
change	**--topic** `device/+/details`	Define the topic filter to subscribe to

   Making these changes to the initial command line results in this command line. Enter this command in the terminal window for your device.

```
python3 pubsub.py --message "" --count 2 --topic device/+/details --root-ca ~/certs/
Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key —
endpoint your-iot-endpoint
```

   The program should display something like this:

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-24d7cdcc-cc01-458c-8488-2d05849691e1'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
```

   If you see something like this on your terminal, your device is ready and listening for messages where the topic names start with `topic-1/` and end with `/detail`. So, let's test that.

3. Here are a couple of messages that your device might receive.

Topic name	Message payload
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

4. Using the MQTT test client in the AWS IoT console, send the messages described in the previous step to your device.

   a. Open the MQTT test client in the AWS IoT console.
   b. In **Subscribe to a topic**, in the **Subscription topic field**, enter the topic filter: **`device/+/details`**, and then choose **Subscribe to topic**.
   c. In the **Subscriptions** column of the MQTT test client, choose **device/+/details**.
   d. For each of the topics in the preceding table, do the following in the MQTT test client:

      1. In **Publish**, enter the value from the **Topic name** column in the table.
      2. In the message payload field below the topic name, enter the value from the **Message payload** column in the table.
      3. Watch the terminal window where `pubsub.py` is running and, in the MQTT test client, choose **Publish to topic**.

      You should see that the message was received by `pubsub.py` in the terminal window.

Exercise result

With this, `pubsub.py`, subscribed to the messages using a wild card topic filter, received them, and displayed them in the terminal window. Notice how you subscribed to a single topic filter, and the callback function was called to process messages having two distinct topics.

Process topic filter subscriptions

Building on the previous exercise, modify the `pubsub.py` sample app to evaluate the message topics and process the subscribed messages based on the topic.

Exercise procedure

To evaluate the message topic

1. Copy `pubsub.py` to `pubsub2.py`.
2. Open `pubsub2.py` in your favorite text editor or IDE.
3. In `pubsub2.py`, find the `on_message_received` function.
4. In `on_message_received`, insert the following code after the line that starts with `print("Received message` and before the line that starts with `global received_count`.

```
    topic_parsed = False
    if "/" in topic:
        parsed_topic = topic.split("/")
        if len(parsed_topic) == 3:
            # this topic has the correct format
            if (parsed_topic[0] == 'device') and (parsed_topic[2] == 'details'):
                # this is a topic we care about, so check the 2nd element
                if (parsed_topic[1] == 'temp'):
                    print("Received temperature request: {}".format(payload))
                    topic_parsed = True
                if (parsed_topic[1] == 'light'):
                    print("Received light request: {}".format(payload))
                    topic_parsed = True
    if not topic_parsed:
        print("Unrecognized message topic.")
```

5. Save your changes and run the modified program by using this command line.

```
python3 pubsub2.py --message "" --count 2 --topic device/+/details --root-ca ~/certs/
Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key —
endpoint your-iot-endpoint
```

6. In the AWS IoT console, open the MQTT test client.
7. In **Subscribe to a topic**, in the **Subscription topic field**, enter the topic filter: **`device/+/details`**, and then choose **Subscribe to topic**.
8. In the **Subscriptions** column of the MQTT test client, choose **device/+/details**.
9. For each of the topics in this table, do the following in the MQTT test client:

Topic name	Message payload
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

   1. In **Publish**, enter the value from the **Topic name** column in the table.
   2. In the message payload field below the topic name, enter the value from the **Message payload** column in the table.
   3. Watch the terminal window where `pubsub.py` is running and, in the MQTT test client, choose **Publish to topic**.

   You should see that the message was received by `pubsub.py` in the terminal window.

You should see something similar to this in your terminal window.

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID 'test-
af794be0-7542-45a0-b0af-0b0ea7474517'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
Received message from topic 'device/light/details': b'{ "desiredLight": 100,
 "currentLight": 50 }'
Received light request: b'{ "desiredLight": 100, "currentLight": 50 }'
Received message from topic 'device/temp/details': b'{ "desiredTemp": 20, "currentTemp":
 15 }'
Received temperature request: b'{ "desiredTemp": 20, "currentTemp": 15 }'
2 message(s) received.
Disconnecting...
Disconnected!
```

Exercise result

In this exercise, you added code so the sample app would recognize and process multiple messages in the callback function. With this, your device could receive messages and act on them.

Another way for your device to receive and process multiple messages would be to subscribe to different messages separately and assign each subscription to its own callback function.

Publish messages from your device

You can use the pubsub.py sample app to publish messages from your device. While it will publish messages as it is, the messages can't be read as JSON documents. This exercise modifies the sample app to be able to publish JSON documents in the message payload that can be read by AWS IoT Core.

Exercise procedure

In this exercise, the following message will be sent with the `device/data` topic.

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
    ]
}
```

To prepare your MQTT test client to monitor the messages from this exercise

1. In **Subscribe to a topic**, in the **Subscription topic field**, enter the topic filter: **`device/data`**, and then choose **Subscribe to topic**.
2. In the **Subscriptions** column of the MQTT test client, choose **device/data**.
3. Leave the MQTT test client window open to wait for messages from your device.

To send JSON documents with the pubsub.py sample app

1. On your device, copy `pubsub.py` to `pubsub3.py`.
2. Edit `pubsub3.py` to change how it formats the messages it publishes.

    a. Open `pubsub3.py` in a text editor.

    b. Locate this line of code:

    ```
    message = "{} [{}]".format(args.message, publish_count)
    ```

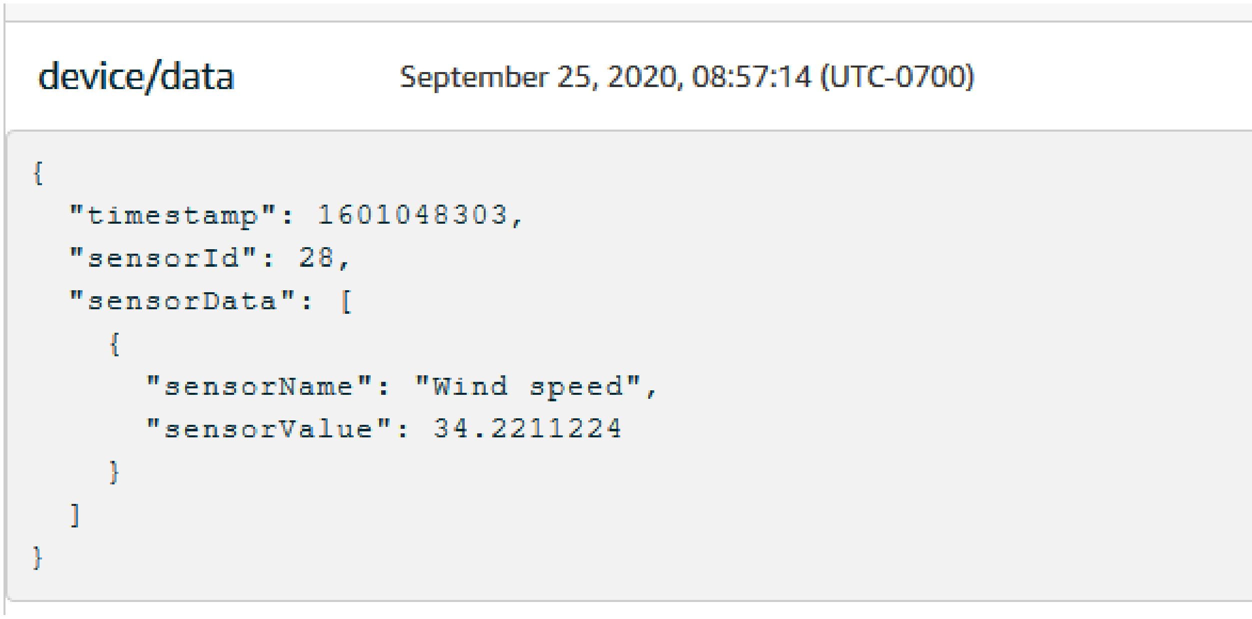
    c. Change it to:

    ```
    message = "{}".format(args.message)
    ```

    d. Save your changes.
3. On your device, run this command to send the message two times.

```
python3 pubsub3.py  --root-ca ~/certs/Amazon-root-CA-1.pem  --cert ~/certs/
device.pem.crt  --key ~/certs/private.pem.key  --topic device/data  --count 2 --
message '{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'  --endpoint your-iot-endpoint
```

4. In the MQTT test client, check to see that it has interpreted and formatted the JSON document in the message payload, such as this:



device/data September 25, 2020, 08:57:14 (UTC-0700)

```
{
  "timestamp": 1601048303,
  "sensorId": 28,
  "sensorData": [
    {
      "sensorName": "Wind speed",
      "sensorValue": 34.2211224
    }
  ]
}
```

By default, `pubsub3.py` also subscribes to the messages it sends. You should see that it received the messages in the app's output. The terminal window should look something like this.

```
Connecting to a3qj468xinsffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-5cff18ae-1e92-4c38-a9d4-7b9771afc52f'...
Connected!
Subscribing to topic 'device/data'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 2 message(s)
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
2 message(s) received.
Disconnecting...
Disconnected!
```

Exercise result

With this, your device can generate messages to send to AWS IoT Core to test basic connectivity and provide device messages for AWS IoT Core to process. For example, you could use this app to send test data from your device to test AWS IoT rule actions.

Review the results

The examples in this tutorial gave you hands-on experience with the basics of how devices can communicate with AWS IoT Core—a fundamental part of your AWS IoT solution. When your devices are able to communicate with AWS IoT Core, they can pass messages to AWS services and other devices on which they can act. Likewise, AWS services and other devices can process information that results in messages sent back to your devices.

When you are ready to explore AWS IoT Core further, try these tutorials:

- the section called "Send an Amazon SNS notification" (p. 141)
- the section called "Store device data in a DynamoDB table" (p. 148)
- the section called "Format a notification by using an AWS Lambda function" (p. 154)

Using the AWS IoT Device SDK for Embedded C

This section describes how to run the AWS IoT Device SDK for Embedded C.

Install the AWS IoT Device SDK for Embedded C

The AWS IoT Device SDK for Embedded C is generally targeted at resource constrained devices that require an optimized C language runtime. You can use the SDK on any operating system and host it on any processor type (for example, MCUs and MPUs). If you have more memory and processing resources available, we recommend that you use one of the higher order AWS IoT Device and Mobile SDKs (for example, C++, Java, JavaScript, and Python).

In general, the AWS IoT Device SDK for Embedded C is intended for systems that use MCUs or low-end MPUs that run embedded operating systems. For the programming example in this section, we assume your device uses Linux.

Example

1. Download the AWS IoT Device SDK for Embedded C to your device from GitHub.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-submodules
   ```

   This creates a directory named `aws-iot-device-sdk-embedded-c` in the current directory.
2. Navigate to that directory and checkout the latest release. Please see github.com/aws/aws-iot-device-sdk-embedded-C/tags for the latest release tag.

   ```
   cd aws-iot-device-sdk-embedded-C
   git checkout latest-release-tag
   ```
3. Install OpenSSL version 1.1.0 or later. The OpenSSL development libraries are usually called "libssl-dev" or "openssl-devel" when installed through a package manager.

   ```
   sudo apt-get install libssl-dev
   ```

Sample app configuration

The AWS IoT Device SDK for Embedded C includes sample applications for you to try. For simplicity, this tutorial uses the `mqtt_demo_mutual_auth` application, that illustrates how to connect to the AWS IoT Core message broker and subscribe and publish to MQTT topics.

1. Copy the certificate and private key you created in Getting started with AWS IoT Core (p. 17) into the `build/bin/certificates` directory.

   Note
   Device and root CA certificates are subject to expiration or revocation. If these certificates expire or are revoked, you must copy a new CA certificate or private key and device certificate onto your device.
2. You must configure the sample with your personal AWS IoT Core endpoint, private key, certificate, and root CA certificate. Navigate to the `aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth` directory.

   If you have the AWS CLI installed, you can use the **aws iot describe-endpoint --endpoint-type iot:Data-ATS** command to find your personal endpoint URL. If you don't have the AWS CLI installed, open your AWS IoT console. From the navigation pane, choose **Manage**, and then choose **Things**. Choose the IoT thing for your device, and then choose **Interact**. Your endpoint is displayed in the **HTTPS** section of the thing details page.
3. Open the `demo_config.h` file and update the values for the following:

   AWS_IOT_ENDPOINT

      Your personal endpoint.

   CLIENT_CERT_PATH

      Your certificate file path, for example `certificates/device.pem.crt"`.

   CLIENT_PRIVATE_KEY_PATH

      Your private key file name, for example `certificates/private.pem.key`.

   For example:

   ```
   // Get from demo_config.h
   // =================================================
   #define AWS_IOT_ENDPOINT               "my-endpoint-ats.iot.us-east-1.amazonaws.com"
   #define AWS_MQTT_PORT                  8883
   #define CLIENT_IDENTIFIER              "testclient"
   #define ROOT_CA_CERT_PATH              "certificates/AmazonRootCA1.crt"
   #define CLIENT_CERT_PATH               "certificates/my-device-cert.pem.crt"
   #define CLIENT_PRIVATE_KEY_PATH        "certificates/my-device-private-key.pem.key"
   // =================================================
   ```
4. Check to see if you have CMake installed on your device by using this command.

   ```
   cmake --version
   ```

   If you see the version information for the compiler, you can continue to the next section.

   If you get an error or don't see any information, then you'll need to install the cmake package using this command.

   ```
   sudo apt-get install cmake
   ```

   Run the **cmake --version** command again and confirm that CMake has been installed and that you are ready to continue.
5. Check to see if you have the development tools installed on your device by using this command.

   ```
   gcc --version
   ```

If you see the version information for the compiler, you can continue to the next section.

If you get an error or don't see any compiler information, you'll need to install the `build-essential` package using this command.

```
sudo apt-get install build-essential
```

Run the **gcc --version** command again and confirm that the build tools have been installed and that you are ready to continue.

Build and run the sample application

To run the AWS IoT Device SDK for Embedded C sample applications

1. Navigate to `aws-iot-device-sdk-embedded-c` and create a build directory.

   ```
   mkdir build && cd build
   ```

2. Enter the following CMake command to generate the Makefiles needed to build.

   ```
   cmake ..
   ```

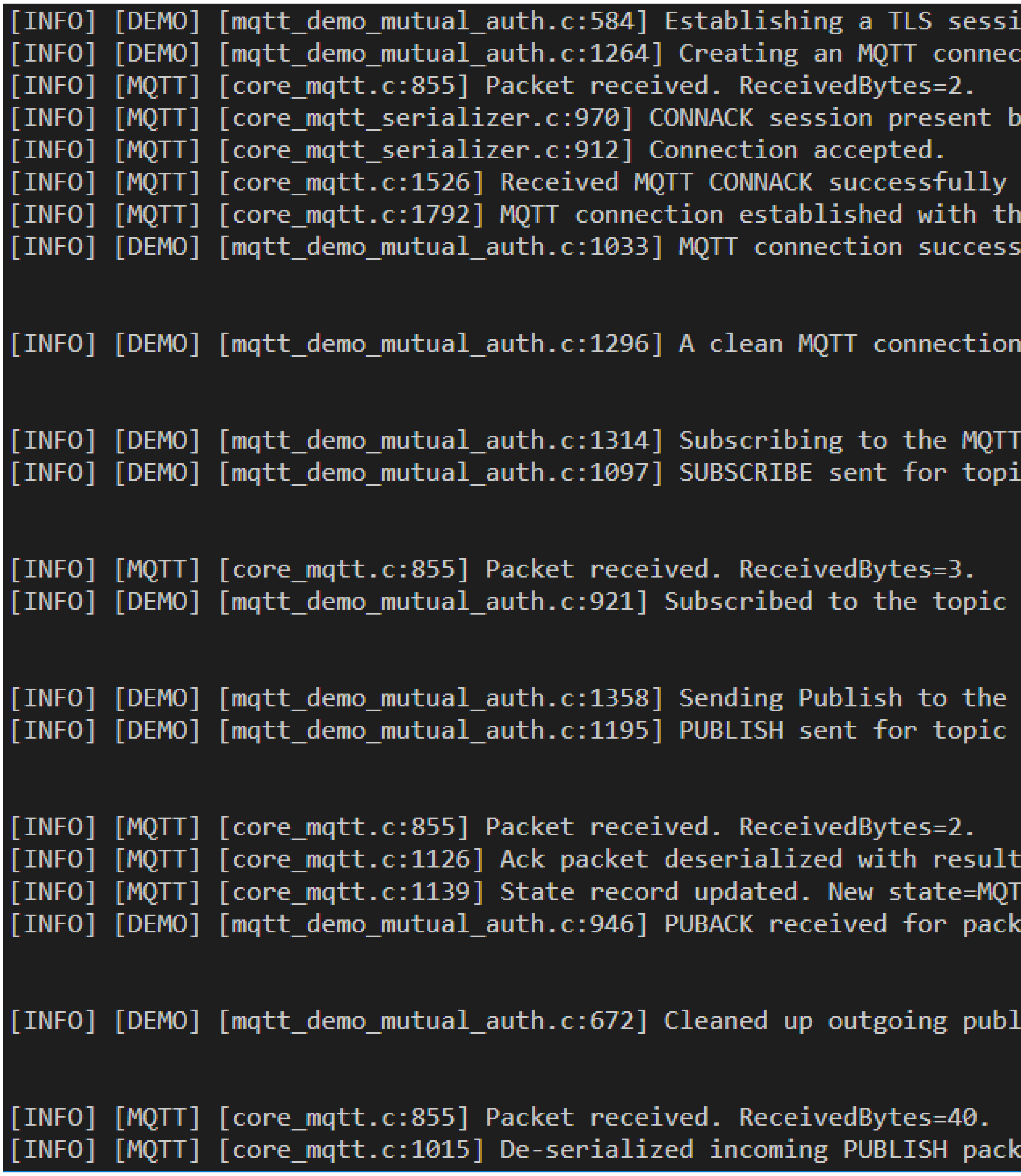
3. Enter the following command to build the executable app file.

   ```
   make
   ```

4. Run the `mqtt_demo_mutual_auth` app with this command.

   ```
   cd bin
   ./mqtt_demo_mutual_auth
   ```

   You should see output similar to the following:



```
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:584] Establishing a TLS sessi
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1264] Creating an MQTT connec
[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt_serializer.c:970] CONNACK session present b
[INFO] [MQTT] [core_mqtt_serializer.c:912] Connection accepted.
[INFO] [MQTT] [core_mqtt.c:1526] Received MQTT CONNACK successfully
[INFO] [MQTT] [core_mqtt.c:1792] MQTT connection established with th
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1033] MQTT connection success


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1296] A clean MQTT connection


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1314] Subscribing to the MQTT
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1097] SUBSCRIBE sent for topi


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:921] Subscribed to the topic


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1358] Sending Publish to the
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1195] PUBLISH sent for topic


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt.c:1126] Ack packet deserialized with result
[INFO] [MQTT] [core_mqtt.c:1139] State record updated. New state=MQT
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:946] PUBACK received for pack


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:672] Cleaned up outgoing publ


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=40.
[INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH pack
```

Your device is now connected to AWS IoT using the AWS IoT Device SDK for Embedded C.

You can also use the AWS IoT console to view the MQTT messages that the sample app is publishing. For information about how to use the MQTT client in the AWS IoT console, see the section called “View MQTT messages with the AWS IoT MQTT client” (p. 60) .

AWS IoT rules tutorials

These tutorials show you how to create and test AWS IoT rules using some of the more common rule actions.

AWS IoT rules send data from your devices to other AWS services. They listen for specific MQTT messages, format the data in the message payloads, and send the result to other AWS services.

We recommend that you try these in the order they are shown here, even if your goal is to create a rule that uses a Lambda function or something even more complex. The tutorials are presented in order from the more basic to the more complex. They present new concepts incrementally to help you learn the concepts you can use to create the rule actions that don't have a specific tutorial.

Note
AWS IoT rules help you send the data from your IoT devices to other AWS services. To do that successfully, however, you'll need a working knowledge of the other services where you want to send the data. While these tutorials provide the necessary information to complete the tasks, you might find it helpful to learn more about the services you want to send data to before you use them in your solution. A detailed explanation of the other AWS services is outside of the scope of these tutorials.

Tutorial scenario overview

The scenario for these tutorials is that of a weather sensor device that periodically publishes its data. There are many such sensor devices in this imaginary system. The tutorials in this section, however, focus on a single device while showing how you might accommodate multiple sensors.

The tutorials in this section show you how to use AWS IoT rules to do the following tasks with this imaginary system of weather sensor devices.

- **Republish an MQTT message (p. 135)**

  This tutorial shows how to republish an MQTT message received from the weather sensors as a message that contains only the sensor ID and the temperature value. It uses only AWS IoT Core services and demonstrates a simple SQL query and how to use the MQTT client to test your rule.
- **Send an Amazon SNS notification (p. 141)**

  This tutorial shows how to send an SNS message when a value from a weather sensor device exceeds a specific value. It builds on the concepts presented in the previous tutorial and adds how to work with another AWS service, the Amazon Simple Notification Service (Amazon SNS).

  If you're new to Amazon SNS, review its Getting started exercises before you start this tutorial.
- **Store device data in a DynamoDB table (p. 148)**

  This tutorial shows how to store the data from the weather sensor devices in a database table. It uses the rule query statement and substitution templates to format the message data for the destination service, Amazon DynamoDB.

  If you're new to DynamoDB, review its Getting started exercises before you start this tutorial.
- **Format a notification by using an AWS Lambda function (p. 154)**

  This tutorial shows how to call a Lambda function to reformat the device data and then send it as a text message. It adds a Python script and AWS SDK functions in an AWS Lambda function to format with the message payload data from the weather sensor devices and send a text message.

  If you're new to Lambda, review its Getting started exercises before you start this tutorial.

AWS IoT rule overview

All of these tutorials create AWS IoT rules.

For an AWS IoT rule to send the data from a device to another AWS service, it uses:

- A rule query statement that consists of:
  - A SQL SELECT clause that selects and formats the data from the message payload
  - A topic filter (the FROM object in the rule query statement) that identifies the messages to use
  - An optional conditional statement (a SQL WHERE clause) that specifies specific conditions on which to act
- At least one rule action

Devices publish messages to MQTT topics. The topic filter in the SQL SELECT statement identifies the MQTT topics to apply the rule to. The fields specified in the SQL SELECT statement format the data from the incoming MQTT message payload for use by the rule's actions. For a complete list of rule actions, see AWS IoT Rule Actions (p. 359).

Tutorials in this section

Republish an MQTT message

This tutorial demonstrates how to create an AWS IoT rule that publishes an MQTT message when a specified MQTT message is received. The incoming message payload can be modified by the rule before it's published, making it possible to create messages that are tailored to specific applications without the need to alter your device or its firmware. You can also use the filtering aspect of a rule to publish messages only when a specific condition is met.

The messages republished by a rule act like messages sent by any other AWS IoT device or client. Devices can subscribe to the republished messages just as they can subscribe to any other MQTT message topic.

What you'll learn in this tutorial:

- How to use simple SQL queries and functions in a rule query statement
- How to use the MQTT client to test an AWS IoT rule

This tutorial takes about 30 minutes to complete.

In this tutorial, you'll:

Before you start this tutorial, make sure that you have:

- **Set up your AWS account (p. 18)**

You'll need your AWS account and AWS IoT console to complete this tutorial.

- **Reviewed View MQTT messages with the AWS IoT MQTT client (p. 60)**

  Be sure you can use the MQTT client to subscribe and publish to a topic. You'll use the MQTT client to test your new rule in this procedure.

Review MQTT topics and AWS IoT rules

Before talking about AWS IoT rules, it helps to understand the MQTT protocol. In IoT solutions, the MQTT protocol offers some advantages over other network communication protocols, such as HTTP, which makes it a popular choice for use by IoT devices. This section reviews the key aspects of MQTT as they apply to this tutorial. For information about how MQTT compares to HTTP, see Choosing a protocol for your device communication (p. 78).

MQTT protocol

The MQTT protocol uses a publish/subscribe communication model with its host. To send data, devices publish messages that are identified by topics to the AWS IoT message broker. To receive messages from the message broker, devices subscribe to the topics they will receive by sending topic filters in subscription requests to the message broker. The AWS IoT rules engine receives MQTT messages from the message broker.

AWS IoT rules

AWS IoT rules consist of a rule query statement and one or more rule actions. When the AWS IoT rules engine receives an MQTT message, these elements act on the message as follows.

- **Rule query statement**

  The rule's query statement describes the MQTT topics to use, interprets the data from the message payload, and formats the data as described by a SQL statement that is similar to statements used by popular SQL databases. The result of the query statement is the data that is sent to the rule's actions.
- **Rule action**

  Each rule action in a rule acts on the data that results from the rule's query statement. AWS IoT supports many rule actions (p. 359). In this tutorial, however, you'll concentrate on the Republish (p. 389) rule action, which publishes the result of the query statement as an MQTT message with a specific topic.

Create an AWS IoT rule to republish an MQTT message

The AWS IoT rule that you'll create in this tutorial subscribes to the `device/`*`device_id`*`/data` MQTT topics where *`device_id`* is the ID of the device that sent the message. These topics are described by a topic filter (p. 86) as `device/+/data`, where the `+` is a wildcard character that matches any string between the two forward slash characters.

When the rule receives a message from a matching topic, it takes out the `device_id` and `temperature` values and republishes them as a new MQTT message with the `device/data/temp` topic.

For example, the payload of an MQTT message with the `device/22/data` topic looks like this:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

The rule takes the `temperature` value from the message payload and the `device_id` from the topic and republishes them as an MQTT message with the `device/data/temp` topic and a message payload that looks like this:

```
{
  "device_id": "22",
  "temperature": 28
}
```

With this rule, devices that only need the device's ID and the temperature data subscribe to the `device/data/temp` topic to receive only that information.

To create a rule that republishes an MQTT message

1. Open the **Rules** hub of the AWS IoT console.
2. In **Rules**, choose **Create** and start creating your new rule.
3. In the top part of **Create a rule**:

   a. In **Name**, enter the rule's name. For this tutorial, name it **`republish_temp`**.

      Remember that a rule name must be unique within your Account and Region, and it can't have any spaces. We've used an underscore character in this name to separate the two words in the rule's name.

   b. In **Description**, describe the rule.

      A meaningful description makes it easier to remember what this rule does and why you created it. The description can be as long as needed, so be as detailed as possible.
4. In **Rule query statement** of **Create a rule**:

   a. In **Using SQL version**, select **`2016-03-23`**.

   b. In the **Rule query statement** edit box, enter the statement:

      ```
      SELECT topic(2) as device_id, temperature FROM 'device/+/data'
      ```

      This statement:

      - Listens for MQTT messages with a topic that matches the `device/+/data` topic filter.
      - Selects the second element from the topic string and assigns it to the `device_id` field.
      - Selects the value `temperature` field from the message payload and assigns it to the `temperature` field.
5. In **Set one or more actions**:

   a. To open up the list of rule actions for this rule, choose **Add action**.

   b. In **Select an action**, choose **Republish a message to an AWS IoT topic**.

   c. At the bottom of the action list, choose **Configure action** to open the selected action's configuration page.
6. In **Configure action**:

   a. In **Topic**, enter **`device/data/temp`**. This is the MQTT topic of the message that this rule will publish.

   b. In **Quality of Service**, choose **0 - The message is delivered zero or more times**.
   c. In **Choose or create a role to grant AWS IoT access to perform this action**:
      i. Choose **Create Role**. The **Create a new role** dialog box opens.
      ii. Enter a name that describes the new role. In this tutorial, use **`republish_role`**.

         When you create a new role, the correct policies to perform the rule action are created and attached to the new role. If you change the topic of this rule action or use this role in another rule action, you must update the policy for that role to authorize the new topic or action. To update an existing role, choose **Update role** in this section.
      iii. Choose **Create Role** to create the role and close the dialog box.
   d. Choose **Add action** to add the action to the rule and return to the **Create a rule** page.

7. The **Republish a message to an AWS IoT topic** action is now listed in **Set one or more actions**.

   In the new action's tile, below **Republish a message to an AWS IoT topic**, you can see the topic to which your republish action will publish.

   This is the only rule action you'll add to this rule.
8. In **Create a rule**, scroll down to the bottom and choose **Create rule** to create the rule and complete this step.

Test your new rule

To test your new rule, you'll use the MQTT client to publish and subscribe to the MQTT messages used by this rule.

Open the MQTT client in the AWS IoT console in a new window. This will let you edit the rule without losing the configuration of your MQTT client. The MQTT client does not retain any subscriptions or message logs if you leave it to go to another page in the console.

To use the MQTT client to test your rule

1. In the MQTT client in the AWS IoT console, subscribe to the input topics, in this case, `device/+/data`.
   a. In the MQTT client, under **Subscriptions**, choose **Subscribe to a topic**.
   b. In **Subscription topic**, enter the topic of the input topic filter, **`device/+/data`**.
   c. Keep the rest of the fields at their default settings.
   d. Choose **Subscribe to topic**.

      In the **Subscriptions** column, under **Publish to a topic**, **`device/+/data`** appears.
2. Subscribe to the topic that your rule will publish: `device/data/temp`.
   a. Under **Subscriptions**, choose **Subscribe to a topic** again, and in **Subscription topic**, enter the topic of the republished message, **`device/data/temp`**.
   b. Keep the rest of the fields at their default settings.
   c. Choose **Subscribe to topic**.

      In the **Subscriptions** column, under **device/+/data**, **`device/data/temp`** appears.
3. Publish a message to the input topic with a specific device ID, **`device/22/data`**. You can't publish to MQTT topics that contain wildcard characters.
   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.
   b. In the **Publish** field, enter the input topic name, **`device/22/data`**.

c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. To send your MQTT message, choose **Publish to topic**.

4. Review the messages that were sent.

   a. In the MQTT client, under **Subscriptions**, there is a green dot next to the two topics to which you subscribed earlier.

      The green dots indicate that one or more new messages have been received since the last time you looked at them.

   b. Under **Subscriptions**, choose **device/+/data** to check that the message payload matches what you just published and looks like this:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   c. Under **Subscriptions**, choose **device/data/temp** to check that your republished message payload looks like this:

```
{
  "device_id": "22",
  "temperature": 28
}
```

      Notice that the `device_id` value is a quoted string and the `temperature` value is numeric. This is because the `topic()` function extracted the string from the input message's topic name while the `temperature` value uses the numeric value from the input message's payload.

      If you want to make the `device_id` value a numeric value, replace `topic(2)` in the rule query statement with:

```
cast(topic(2) AS DECIMAL)
```

      Note that casting the `topic(2)` value to a numeric value will only work if that part of the topic contains only numeric characters.

5. If you see that the correct message was published to the **device/data/temp** topic, then your rule worked. See what more you can learn about the Republish rule action in the next section.

   If you don't see that the correct message was published to either the **device/+/data** or **device/data/temp** topics, check the troubleshooting tips.

Troubleshooting your Republish message rule

Here are some things to check in case you're not seeing the results you expect.

- **You got an error banner**

  If an error appeared when you published the input message, correct that error first. The following steps might help you correct that error.
- **You don't see the input message in the MQTT client**

  Every time you publish your input message to the `device/22/data` topic, that message should appear in the MQTT client if you subscribed to the `device/+/data` topic filter as described in the procedure.

  ### Things to check

  - **Check the topic filter you subscribed to**

    If you subscribed to the input message topic as described in the procedure, you should see a copy of the input message every time you publish it.

    If you don't see the message, check the topic name you subscribed to and compare it to the topic to which you published. Topic names are case sensitive and the topic to which you subscribed must be identical to the topic to which you published the message payload.
  - **Check the message publish function**

    In the MQTT client, under **Subscriptions**, choose **device/+/data**, check the topic of the publish message, and then choose **Publish to topic**. You should see the message payload from the edit box below the topic appear in the message list.
- **You don't see your republished message in the MQTT client**

  For your rule to work, it must have the correct policy that authorizes it to receive and republish a message and it must receive the message.

  ### Things to check

  - **Check the AWS Region of your MQTT client and the rule that you created**

    The console in which you're running the MQTT client must be in the same AWS Region as the rule you created.
  - **Check the input message topic in the rule query statement**

    For the rule to work, it must receive a message with the topic name that matches the topic filter in the FROM clause of the rule query statement.

    Check the spelling of the topic filter in the rule query statement with that of the topic in the MQTT client. Topic names are case sensitive and the message's topic must match the topic filter in the rule query statement.
  - **Check the contents of the input message payload**

    For the rule to work, it must find the data field in the message payload that is declared in the SELECT statement.

    Check the spelling of the `temperature` field in the rule query statement with that of the message payload in the MQTT client. Field names are case sensitive and the `temperature` field in the rule query statement must be identical to the `temperature` field in the message payload.

    Make sure that the JSON document in the message payload is correctly formatted. If the JSON has any errors, such as a missing comma, the rule will not be able to read it.

- **Check the republished message topic in the rule action**

  The topic to which the Republish rule action publishes the new message must match the topic to which you subscribed in the MQTT client.

  Open the rule you created in the console and check the topic to which the rule action will republish the message.
- **Check the role being used by the rule**

  The rule action must have permission to receive the original topic and publish the new topic.

  The policies that authorize the rule to receive message data and republish it are specific to the topics used. If you change the topic used to republish the message data, you must update the rule action's role to update its policy to match the current topic.

  If you suspect this is the problem, edit the Republish rule action and create a new role. New roles created by the rule action receive the authorizations necessary to perform these actions.

Review the results and next steps

In this tutorial

- You used a simple SQL query and a couple of functions in a rule query statement to produce a new MQTT message.
- You created a rule that republished that new message.
- You used the MQTT client to test your AWS IoT rule.

Next steps

After you republish a few messages with this rule, try experimenting with it to see how changing some aspects of the tutorial affect the republished message. Here are some ideas to get you started.

- Change the *`device_id`* in the input message's topic and observe the effect in the republished message payload.
- Change the fields selected in the rule query statement and observe the effect in the republished message payload.
- Try the next tutorial in this series and learn how to Send an Amazon SNS notification (p. 141).

The Republish rule action used in this tutorial can also help you debug rule query statements. For example, you can add this action to a rule to see how its rule query statement is formatting the data used by its rule actions.

Send an Amazon SNS notification

This tutorial demonstrates how to create an AWS IoT rule that sends MQTT message data to an Amazon SNS topic so that it can be sent as an SMS text message.

In this tutorial, you create a rule that sends message data from a weather sensor to all subscribers of an Amazon SNS topic, whenever the temperature exceeds the value set in the rule. The rule detects when the reported temperature exceeds the value set by the rule and creates a new message payload that includes only the device ID, the reported temperature, and the temperature limit that was exceeded. The rule sends the new message payload as a JSON document to an SNS topic, which notifies all subscribers to the SNS topic.

What you'll learn in this tutorial:

- How to create and test an Amazon SNS notification
- How to call an Amazon SNS notification from an AWS IoT rule
- How to use simple SQL queries and functions in a rule query statement
- How to use the MQTT client to test an AWS IoT rule

This tutorial takes about 30 minutes to complete.

In this tutorial, you'll:

- Create an Amazon SNS topic that sends an SMS text message (p. 142)
- Create an AWS IoT rule to send the text message (p. 143)
- Test the AWS IoT rule and Amazon SNS notification (p. 145)
- Review the results and next steps (p. 148)

Before you start this tutorial, make sure that you have:

- **Set up your AWS account (p. 18)**

  You'll need your AWS account and AWS IoT console to complete this tutorial.
- **Reviewed View MQTT messages with the AWS IoT MQTT client (p. 60)**

  Be sure you can use the MQTT client to subscribe and publish to a topic. You'll use the MQTT client to test your new rule in this procedure.
- **Reviewed the Amazon Simple Notification Service**

  If you haven't used Amazon SNS before, review Setting up access for Amazon SNS. If you've already completed other AWS IoT tutorials, your AWS account should already be configured correctly.

Create an Amazon SNS topic that sends an SMS text message

To create an Amazon SNS topic that sends an SMS text message

1. **Create an Amazon SNS topic.**

   a. Sign in to the Amazon SNS console.

   b. In the left navigation pane, choose **Topics**.

   c. On the **Topics** page, choose **Create topic**.

   d. In **Details**, choose the **Standard** type. By default, the console creates a FIFO topic.

   e. In **Name**, enter the SNS topic name. For this tutorial, enter **`high_temp_notice`**.

   f. Scroll to the end of the page and choose **Create topic**.

      The console opens the new topic's **Details** page.
2. **Create an Amazon SNS subscription.**

   Note
   The phone number that you use in this subscription might incur text messaging charges from the messages you will send in this tutorial.

   a. In the **high_temp_notice** topic's details page, choose **Create subscription**.

   b. In **Create subscription**, in the **Details** section, in the **Protocol** list, choose **SMS**.

   c. In **Endpoint**, enter the number of a phone that can receive text messages. Be sure to enter it such that it starts with a +, includes the country and area code, and doesn't include any other punctuation characters.
   d. Choose **Create subscription**.

3. **Test the Amazon SNS notification.**

   a. In the Amazon SNS console, in the left navigation pane, choose **Topics**.
   b. To open the topic's details page, in **Topics**, in the list of topics, choose **high_temp_notice**.
   c. To open the **Publish message to topic** page, in the **high_temp_notice** details page, choose **Publish message**.
   d. In **Publish message to topic**, in the **Message body** section, in **Message body to send to the endpoint**, enter a short message.
   e. Scroll down to the bottom of the page and choose **Publish message**.
   f. On the phone with the number you used earlier when creating the subscription, confirm the message was received.

   If you did not receive the test message, double check the phone number and your phone's settings.

   Make sure you can publish test messages from the Amazon SNS console before you continue the tutorial.

Create an AWS IoT rule to send the text message

The AWS IoT rule that you'll create in this tutorial subscribes to the `device/`*`device_id`*`/data` MQTT topics where *`device_id`* is the ID of the device that sent the message. These topics are described in a topic filter as `device/+/data`, where the `+` is a wildcard character that matches any string between the two forward slash characters. This rule also tests the value of the `temperature` field in the message payload.

When the rule receives a message from a matching topic, it takes the *`device_id`* from the topic name, the `temperature` value from the message payload, and adds a constant value for the limit it's testing, and sends these values as a JSON document to an Amazon SNS notification topic.

For example, an MQTT message from weather sensor device number 32 uses the `device/32/data` topic and has a message payload that looks like this:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

The rule's rule query statement takes the `temperature` value from the message payload, the *`device_id`* from the topic name, and adds the constant `max_temperature` value to send a message payload that looks like this to the Amazon SNS topic:

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

To create an AWS IoT rule to detect an over-limit temperature value and create the data to send to the Amazon SNS topic:

1. Open the **Rules** hub of the AWS IoT console.
2. If this is your first rule, choose **Create**, or **Create a rule**.
3. In **Create a rule**:

   a. In **Name**, enter **`temp_limit_notify`**.

      Remember that a rule name must be unique within your AWS Account and Region, and it can't have any spaces. We've used an underscore character in this name to separate the words in the rule's name.

   b. In **Description**, describe the rule.

      A meaningful description makes it easier to remember what this rule does and why you created it. The description can be as long as needed, so be as detailed as possible.
4. In **Rule query statement** of **Create a rule**:

   a. In **Using SQL version**, select **2016-03-23**.
   b. In the **Rule query statement** edit box, enter the statement:

```
SELECT topic(2) as device_id,
    temperature as reported_temperature,
    30 as max_temperature
  FROM 'device/+/data'
  WHERE temperature > 30
```

   This statement:

   - Listens for MQTT messages with a topic that matches the `device/+/data` topic filter and that have a `temperature` value greater than 30.
   - Selects the second element from the topic string and assigns it to the `device_id` field.
   - Selects the value `temperature` field from the message payload and assigns it to the `reported_temperature` field.
   - Creates a constant value `30` to represent the limit value and assigns it to the `max_temperature` field.
5. To open up the list of rule actions for this rule, in **Set one or more actions**, choose **Add action**.
6. In **Select an action**, choose **Send a message as an SNS push notification**.
7. To open the selected action's configuration page, at the bottom of the action list, choose **Configure action**.
8. In **Configure action**:

   a. In **SNS target**, choose **Select**, find your SNS topic named **high_temp_notice**, and choose **Select**.
   b. In **Message format**, choose **RAW**.
   c. In **Choose or create a role to grant AWS IoT access to perform this action**, choose **Create Role**.
   d. In **Create a new role**, in **Name**, enter a unique name for the new role. For this tutorial, use **`sns_rule_role`**.
   e. Choose **Create role**.

   If you're repeating this tutorial or reusing an existing role, choose **Update role** before continuing. This updates the role's policy document to work with the SNS target.

9. Choose **Add action** and return to the **Create a rule** page.

   In the new action's tile, below **Send a message as an SNS push notification**, you can see the SNS topic that your rule will call.

   This is the only rule action you'll add to this rule.
10. To create the rule and complete this step, in **Create a rule**, scroll down to the bottom and choose **Create rule**.

Test the AWS IoT rule and Amazon SNS notification

To test your new rule, you'll use the MQTT client to publish and subscribe to the MQTT messages used by this rule.

Open the MQTT client in the AWS IoT console in a new window. This will let you edit the rule without losing the configuration of your MQTT client. If you leave the MQTT client to go to another page in the console, it won't retain any subscriptions or message logs.

To use the MQTT client to test your rule

1. In the MQTT client in the AWS IoT console, subscribe to the input topics, in this case, `device/+/data`.

   a. In the MQTT client, under **Subscriptions**, choose **Subscribe to a topic**.
   b. In **Subscription topic**, enter the topic of the input topic filter, **`device/+/data`**.
   c. Keep the rest of the fields at their default settings.
   d. Choose **Subscribe to topic**.

      In the **Subscriptions** column, under **Publish to a topic**, **`device/+/data`** appears.
2. Publish a message to the input topic with a specific device ID, **`device/32/data`**. You can't publish to MQTT topics that contain wildcard characters.

   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.
   b. In the **Publish** field, enter the input topic name, **`device/32/data`**.
   c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. Choose **Publish to topic** to publish your MQTT message.
3. Confirm that the text message was sent.

   a. In the MQTT client, under **Subscriptions**, there is a green dot next to the topic to which you subscribed earlier.

      The green dot indicates that one or more new messages have been received since the last time you looked at them.
   b. Under **Subscriptions**, choose **device/+/data** to check that the message payload matches what you just published and looks like this:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. Check the phone that you used to subscribe to the SNS topic and confirm the contents of the message payload look like this:

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

Notice that the `device_id` value is a quoted string and the `temperature` value is numeric. This is because the `topic()` function extracted the string from the input message's topic name while the `temperature` value uses the numeric value from the input message's payload.

If you want to make the `device_id` value a numeric value, replace `topic(2)` in the rule query statement with:

```
cast(topic(2) AS DECIMAL)
```

Note that casting the `topic(2)` value to a numeric, `DECIMAL` value will only work if that part of the topic contains only numeric characters.

4. Try sending an MQTT message in which the temperature does not exceed the limit.

   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.
   b. In the **Publish** field, enter the input topic name, **`device/33/data`**.
   c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. To send your MQTT message, choose **Publish to topic**.

You should see the message that you sent in the **`device/+/data`** subscription, however, because the temperature value is below the max temperature in the rule query statement, you shouldn't receive a text message.

If you don't see the correct behavior, check the troubleshooting tips.

Troubleshooting your SNS message rule

Here are some things to check, in case you're not seeing the results you expect.

- **You got an error banner**

  If an error appeared when you published the input message, correct that error first. The following steps might help you correct that error.
- **You don't see the input message in the MQTT client**

  Every time you publish your input message to the `device/22/data` topic, that message should appear in the MQTT client, if you subscribed to the `device/+/data` topic filter as described in the procedure.

  ### Things to check

  - **Check the topic filter you subscribed to**

    If you subscribed to the input message topic as described in the procedure, you should see a copy of the input message every time you publish it.

    If you don't see the message, check the topic name you subscribed to and compare it to the topic to which you published. Topic names are case sensitive and the topic to which you subscribed must be identical to the topic to which you published the message payload.
  - **Check the message publish function**

    In the MQTT client, under **Subscriptions**, choose **device/+/data**, check the topic of the publish message, and then choose **Publish to topic**. You should see the message payload from the edit box below the topic appear in the message list.
- **You don't receive an SMS message**

  For your rule to work, it must have the correct policy that authorizes it to receive a message and send an SNS notification, and it must receive the message.

  ### Things to check

  - **Check the AWS Region of your MQTT client and the rule that you created**

    The console in which you're running the MQTT client must be in the same AWS Region as the rule you created.
  - **Check that the temperature value in the message payload exceeds the test threshold**

    If the temperature value is less than or equal to 30, as defined in the rule query statement, the rule will not perform any of its actions.
  - **Check the input message topic in the rule query statement**

    For the rule to work, it must receive a message with the topic name that matches the topic filter in the FROM clause of the rule query statement.

    Check the spelling of the topic filter in the rule query statement with that of the topic in the MQTT client. Topic names are case sensitive and the message's topic must match the topic filter in the rule query statement.
  - **Check the contents of the input message payload**

    For the rule to work, it must find the data field in the message payload that is declared in the SELECT statement.

    Check the spelling of the `temperature` field in the rule query statement with that of the message payload in the MQTT client. Field names are case sensitive and the `temperature` field in the rule query statement must be identical to the `temperature` field in the message payload.

Make sure that the JSON document in the message payload is correctly formatted. If the JSON has any errors, such as a missing comma, the rule will not be able to read it.

- **Check the republished message topic in the rule action**

  The topic to which the Republish rule action publishes the new message must match the topic to which you subscribed in the MQTT client.

  Open the rule you created in the console and check the topic to which the rule action will republish the message.
- **Check the role being used by the rule**

  The rule action must have permission to receive the original topic and publish the new topic.

  The policies that authorize the rule to receive message data and republish it are specific to the topics used. If you change the topic used to republish the message data, you must update the rule action's role to update its policy to match the current topic.

  If you suspect this is the problem, edit the Republish rule action and create a new role. New roles created by the rule action receive the authorizations necessary to perform these actions.

Review the results and next steps

In this tutorial:

- You created and tested an Amazon SNS notification topic and subscription.
- You used a simple SQL query and functions in a rule query statement to create a new message for your notification.
- You created an AWS IoT rule to send an Amazon SNS notification that used your customized message payload.
- You used the MQTT client to test your AWS IoT rule.

Next steps

After you send a few text messages with this rule, try experimenting with it to see how changing some aspects of the tutorial affect the message and when it's sent. Here are some ideas to get you started.

- Change the *`device_id`* in the input message's topic and observe the effect in the text message contents.
- Change the fields selected in the rule query statement and observe the effect in the text message contents.
- Change the test in the rule query statement to test for a minimum temperature instead of a maximum temperature. Remember to change the name of `max_temperature`!
- Add a republish rule action to send an MQTT message when an SNS notification is sent.
- Try the next tutorial in this series and learn how to Store device data in a DynamoDB table (p. 148).

Store device data in a DynamoDB table

This tutorial demonstrates how to create an AWS IoT rule that sends message data to a DynamoDB table.

In this tutorial, you create a rule that sends message data from an imaginary weather sensor device to a DynamoDB table. The rule formats the data from many weather sensors such that they can be added to a single database table.

What you'll learn in this tutorial

- How to create a DynamoDB table
- How to send message data to a DynamoDB table from an AWS IoT rule
- How to use substitution templates in an AWS IoT rule
- How to use simple SQL queries and functions in a rule query statement
- How to use the MQTT client to test an AWS IoT rule

This tutorial takes about 30 minutes to complete.

In this tutorial, you'll:

- Create the DynamoDB table for this tutorial (p. 149)
- Create an AWS IoT rule to send data to the DynamoDB table (p. 150)
- Test the AWS IoT rule and DynamoDB table (p. 151)
- Review the results and next steps (p. 154)

Before you start this tutorial, make sure that you have:

- **Set up your AWS account (p. 18)**

  You'll need your AWS account and AWS IoT console to complete this tutorial.
- **Reviewed View MQTT messages with the AWS IoT MQTT client (p. 60)**

  Be sure you can use the MQTT client to subscribe and publish to a topic. You'll use the MQTT client to test your new rule in this procedure.
- **Reviewed the Amazon DynamoDB overview**

  If you've not used DynamoDB before, review Getting Started with DynamoDB to become familiar with the basic concepts and operations of DynamoDB.

Create the DynamoDB table for this tutorial

In this tutorial, you'll create a DynamoDB table with these attributes to record the data from the imaginary weather sensor devices:

- `sample_time` is a primary key and describes the time the sample was recorded.
- `device_id` is a sort key and describes the device that provided the sample
- `device_data` is the data received from the device and formatted by the rule query statement

To create the DynamoDB table for this tutorial

1. Open the DynamoDB console, and then choose **Create table**.
2. In **Create DynamoDB table**:

   a. In **Table name**, enter the table name: **`wx_data`**.

   b. In **Primary key**, in **Partition key**, enter **`sample_time`**, and in the option list next to the field, choose **`Number`**.

   c. Check **Add sort key**.

   d. In the field that appears below **Add sort key**, enter **`device_id`**, and in the option list next to the field, choose **`Number`**.

   e. At the bottom of the page, choose **Create**.

You'll define `device_data` later, when you configure the DynamoDB rule action.

Create an AWS IoT rule to send data to the DynamoDB table

In this step, you'll use the rule query statement to format the data from the imaginary weather sensor devices to write to the database table.

A sample message payload received from a weather sensor devices looks like this:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

For the database entry, you'll use the rule query statement to flatten the structure of the message payload to look like this:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

In this rule, you'll also use a couple of Substitution templates (p. 485). Substitution templates are expressions that let you insert dynamic values from functions and message data.

To create the AWS IoT rule to send data to the DynamoDB table

1. Open the **Rules** hub of the AWS IoT console.
2. To start creating your new rule in **Rules**, choose **Create**.
3. In the top part of **Create a rule**:

   a. In **Name**, enter the rule's name, **`wx_data_ddb`**.

      Remember that a rule name must be unique within your AWS Account and Region, and it can't have any spaces. We've used an underscore character in this name to separate the two words in the rule's name.

   b. In **Description**, describe the rule.

      A meaningful description makes it easier to remember what this rule does and why you created it. The description can be as long as needed, so be as detailed as possible.
4. In **Rule query statement** of **Create a rule**:

   a. In **Using SQL version**, select **`2016-03-23`**.

   b. In the **Rule query statement** edit box, enter the statement:

      ```
      SELECT temperature, humidity, barometer,
        wind.velocity as wind_velocity,
        wind.bearing as wind_bearing,
      FROM 'device/+/data'
      ```

This statement:

- Listens for MQTT messages with a topic that matches the `device/+/data` topic filter.
- Formats the elements of the `wind` attribute as individual attributes.
- Passes the `temperature`, `humidity`, and `barometer` attributes unchanged.

5. In **Set one or more actions**:

   a. To open up the list of rule actions for this rule, choose **Add action**.

   b. In **Select an action**, choose **Insert a message into a DynamoDB table**.

   c. To open the selected action's configuration page, at the bottom of the action list, choose **Configure action**.

6. In **Configure action**:

   a. In **Table name**, choose the name of the DynamoDB table you created in a previous step: **`wx_data`**.

      The **Partition key**, **Partition key type**, **Sort key**, and **Sort key type** fields are filled with the values from your DynamoDB table.

   b. In **Partition key value**. enter **`${timestamp()}`**.

      This is the first of the Substitution templates (p. 485) you'll use in this rule. Instead of using a value from the message payload, it will use the value returned from the timestamp (p. 479) function.

   c. In Sort key value, enter **`${cast(topic(2) AS DECIMAL)}`**.

      This is the second one of the Substitution templates (p. 485) you'll use in this rule. It inserts the value of the second element in topic (p. 479) name, which is the device's ID, after it casts (p. 448) it to a DECIMAL value to match the numeric format of the key.

   d. In **Write message data to this column**, enter **`device_data`**.

      This will create the `device_data` column in the DynamoDB table.

   e. Leave **Operation** blank.

   f. In **Choose or create a role to grant AWS IoT access to perform this action**, choose **Create Role**.

   g. In **Create a new role**, enter **`wx_ddb_role`**, and choose **Create role**.

   h. At the bottom of **Configure action**, choose **Add action**.

   i. To create the rule, at the bottom of **Create a rule**, choose **Create rule**.

Test the AWS IoT rule and DynamoDB table

To test the new rule, you'll use the MQTT client to publish and subscribe to the MQTT messages used in this test.

Open the MQTT client in the AWS IoT console in a new window. This will let you edit the rule without losing the configuration of your MQTT client. The MQTT client does not retain any subscriptions or message logs if you leave it to go to another page in the console. You'll also want a separate console window open to the DynamoDB Tables hub in the AWS IoT console to view the new entries that your rule sends.

To use the MQTT client to test your rule

1. In the MQTT client in the AWS IoT console, subscribe to the input topic, `device/+/data`.

   a. In the MQTT client, under **Subscriptions**, choose **Subscribe to a topic**.

   b. In **Subscription topic**, enter the topic of the input topic filter, **`device/+/data`**.
   c. Keep the rest of the fields at their default settings.
   d. Choose **Subscribe to topic**.

      In the **Subscriptions** column, under **Publish to a topic**, **`device/+/data`** appears.

   You should see the message that you sent in the **`device/+/data`** subscription.

2. Publish a message to the input topic with a specific device ID, **`device/22/data`**. You can't publish to MQTT topics that contain wildcard characters.

   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.
   b. In the **Publish** field, enter the input topic name, **`device/22/data`**.
   c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. To publish the MQTT message, choose **Publish to topic**.

3. Check to see the row in the DynamoDB table that your rule created.

   a. In the DynamoDB Tables hub in the AWS IoT console, choose **wx_data**, and then choose the **Items** tab.

      If you're already on the **Items** tab, you might need to refresh the display by choosing the refresh icon in the upper-right corner of the table's header.
   b. Notice that the **sample_time** values in the table are links and open one. If you just sent your first message, it will be the only one in the list.

      This link displays all the data in that row of the table.
   c. Expand the **device_data** entry to see the data that resulted from the rule query statement.
   d. Explore the different representations of the data that are available in this display. You can also edit the data in this display.
   e. After you have finished reviewing this row of data, to save any changes you made, choose **Save**, or to exit without saving any changes, choose **Cancel**.

If you don't see the correct behavior, check the troubleshooting tips.

Troubleshooting your DynamoDB rule

Here are some things to check in case you're not seeing the results you expect.

- **You got an error banner**

  If an error appeared when you published the input message, correct that error first. The following steps might help you correct that error.
- **You don't see the input message in the MQTT client**

Every time you publish your input message to the `device/22/data` topic, that message should appear in the MQTT client if you subscribed to the `device/+/data` topic filter as described in the procedure.

Things to check

- **Check the topic filter you subscribed to**

  If you subscribed to the input message topic as described in the procedure, you should see a copy of the input message every time you publish it.

  If you don't see the message, check the topic name you subscribed to and compare it to the topic to which you published. Topic names are case sensitive and the topic to which you subscribed must be identical to the topic to which you published the message payload.
- **Check the message publish function**

  In the MQTT client, under **Subscriptions**, choose **device/+/data**, check the topic of the publish message, and then choose **Publish to topic**. You should see the message payload from the edit box below the topic appear in the message list.

- **You don't see your data in the DynamoDB table**

  The first thing to do is to manually refresh the display by choosing the refresh icon in the upper-right corner of the table's header. If that doesn't display the data you're looking for, check the following.

  ### Things to check

  - **Check the AWS Region of your MQTT client and the rule that you created**

    The console in which you're running the MQTT client must be in the same AWS Region as the rule you created.
  - **Check the input message topic in the rule query statement**

    For the rule to work, it must receive a message with the topic name that matches the topic filter in the FROM clause of the rule query statement.

    Check the spelling of the topic filter in the rule query statement with that of the topic in the MQTT client. Topic names are case sensitive and the message's topic must match the topic filter in the rule query statement.
  - **Check the contents of the input message payload**

    For the rule to work, it must find the data field in the message payload that is declared in the SELECT statement.

    Check the spelling of the `temperature` field in the rule query statement with that of the message payload in the MQTT client. Field names are case sensitive and the `temperature` field in the rule query statement must be identical to the `temperature` field in the message payload.

    Make sure that the JSON document in the message payload is correctly formatted. If the JSON has any errors, such as a missing comma, the rule will not be able to read it.
  - **Check the key and field names used in the rule action**

    The field names used in the topic rule must match those found in the JSON message payload of the published message.

    Open the rule you created in the console and check the field names in the rule action configuration with those used in the MQTT client.
  - **Check the role being used by the rule**

The rule action must have permission to receive the original topic and publish the new topic.

The policies that authorize the rule to receive message data and update the DynamoDB table are specific to the topics used. If you change the topic or DynamoDB table name used by the rule, you must update the rule action's role to update its policy to match.

If you suspect this is the problem, edit the rule action and create a new role. New roles created by the rule action receive the authorizations necessary to perform these actions.

Review the results and next steps

After you send a few messages to the DynamoDB table with this rule, try experimenting with it to see how changing some aspects from the tutorial affect the data written to the table. Here are some ideas to get you started.

- Change the *device_id* in the input message's topic and observe the effect on the data. You could use this to simulate receiving data from multiple weather sensors.
- Change the fields selected in the rule query statement and observe the effect on the data. You could use this to filter the data stored in the table.
- Add a republish rule action to send an MQTT message for each row added to the table. You could use this for debugging.

After you have completed this tutorial, check out the next one!

Format a notification by using an AWS Lambda function

This tutorial demonstrates how to send MQTT message data to an AWS Lambda action for formatting and sending to another AWS service. In this tutorial, the AWS Lambda action uses the AWS SDK to send the formatted message to the Amazon SNS topic you created in the tutorial about how to the section called “Send an Amazon SNS notification” (p. 141).

In the tutorial about how to the section called “Send an Amazon SNS notification” (p. 141), the JSON document that resulted from the rule's query statement was sent as the body of the text message. The result was a text message that looked something like this example:

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

In this tutorial, you'll use an AWS Lambda rule action to call an AWS Lambda function that formats the data from the rule query statement into a friendlier format, such as this example:

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

The AWS Lambda function you'll create in this tutorial formats the message string by using the data from the rule query statement and calls the SNS publish function of the AWS SDK to create the notification.

What you'll learn in this tutorial

- How to create and test an AWS Lambda function
- How to use the AWS SDK in an AWS Lambda function to publish an Amazon SNS notification
- How to use simple SQL queries and functions in a rule query statement
- How to use the MQTT client to test an AWS IoT rule

This tutorial takes about 45 minutes to complete.

In this tutorial, you'll:

- Create an AWS Lambda function that sends a text message (p. 155)
- Create an AWS IoT rule with an AWS Lambda rule action (p. 157)
- Test the AWS IoT rule and AWS Lambda rule action (p. 159)
- Review the results and next steps (p. 162)

Before you start this tutorial, make sure that you have:

- **Set up your AWS account (p. 18)**

  You'll need your AWS account and AWS IoT console to complete this tutorial.
- **Reviewed View MQTT messages with the AWS IoT MQTT client (p. 60)**

  Be sure you can use the MQTT client to subscribe and publish to a topic. You'll use the MQTT client to test your new rule in this procedure.
- **Completed the other rules tutorials in this section**

  This tutorial requires the SNS notification topic you created in the tutorial about how to the section called "Send an Amazon SNS notification" (p. 141). It also assumes that you've completed the other rules-related tutorials in this section.
- **Reviewed the AWS Lambda overview**

  If you haven't used AWS Lambda before, review AWS Lambda and Getting started with Lambda to learn its terms and concepts.

Create an AWS Lambda function that sends a text message

The AWS Lambda function in this tutorial receives the result of the rule query statement, inserts the elements into a text string, and sends the resulting string to Amazon SNS as the message in a notification.

Unlike the tutorial about how to the section called "Send an Amazon SNS notification" (p. 141), which used an AWS IoT rule action to send the notification, this tutorial sends the notification from the Lambda function by using a function of the AWS SDK. The actual Amazon SNS notification topic used in this tutorial, however, is the same one that you used in the tutorial about how to the section called "Send an Amazon SNS notification" (p. 141).

To create an AWS Lambda function that sends a text message

1. Create a new AWS Lambda function.

   a. In the AWS Lambda console, choose **Create function**.

   b. In **Create function**, select **Use a blueprint**.

      Search for and select the `hello-world-python` blueprint, and then choose **Configure**.

   c. In **Basic information**:

      i. In **Function name**, enter the name of this function, `format-high-temp-notification`.

      ii. In **Execution role**, choose **Create a new role from AWS policy templates**.

      iii. In Role name, enter the name of the new role, `format-high-temp-notification-role`.

      iv. In **Policy templates - *optional***, search for and select **Amazon SNS publish policy**.

v. Choose **Create function**.

2. Modify the blueprint code to format and send an Amazon SNS notification.

   a. After you created your function, you should see the **format-high-temp-notification** details page. If you don't, open it from the Lambda **Functions** page.

   b. In the **format-high-temp-notification** details page, choose the **Configuration** tab and scroll to the **Function code** panel.

   c. In the **Function code** window, in the **Environment** pane, choose the Python file, `lambda_function.py`.

   d. In the **Function code** window, delete all of the original program code from the blueprint and replace it with this code.

```
import boto3
#
#   expects event parameter to contain:
#   {
#       "device_id": "32",
#       "reported_temperature": 38,
#       "max_temperature": 30,
#       "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
#   }
#
#   sends a plain text string to be used in a text message
#
#      "Device {0} reports a temperature of {1}, which exceeds the limit of {2}."
#
#   where:
#       {0} is the device_id value
#       {1} is the reported_temperature value
#       {2} is the max_temperature value
#
def lambda_handler(event, context):

    # Create an SNS client to send notification
    sns = boto3.client('sns')

    # Format text message from data
    message_text = "Device {0} reports a temperature of {1}, which exceeds the
 limit of {2}.".format(
            str(event['device_id']),
            str(event['reported_temperature']),
            str(event['max_temperature'])
        )

    # Publish the formatted message
    response = sns.publish(
            TopicArn = event['notify_topic_arn'],
            Message = message_text
        )

    return response
```

   e. Choose **Deploy**.

3. In a new window, look up the Amazon Resource Name (ARN) of your Amazon SNS topic from the tutorial about how to the section called “Send an Amazon SNS notification” (p. 141).

   a. In a new window, open the Topics page of the Amazon SNS console.

   b. In the **Topics** page, find the **high_temp_notice** notification topic in the list of Amazon SNS topics.

   c. Find the **ARN** of the **high_temp_notice** notification topic to use in the next step.

4. Create a test case for your Lambda function.

   a. In the Lambda **Functions** page of the console, on the **format-high-temp-notification** details page, choose **Select a test event** in the upper right corner of the page (even though it looks disabled), and then choose **Configure test events**.
   b. In **Configure test event**, choose **Create new test event**.
   c. In **Event name**, enter **`SampleRuleOutput`**.
   d. In the JSON editor below **Event name**, paste this sample JSON document. This is an example of what your AWS IoT rule will send to the Lambda function.

      ```
      {
        "device_id": "32",
        "reported_temperature": 38,
        "max_temperature": 30,
        "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
      }
      ```
   e. Refer to the window that has the **ARN** of the **high_temp_notice** notification topic and copy the ARN value.
   f. Replace the `notify_topic_arn` value in the JSON editor with the ARN from your notification topic.

      Keep this window open so you can use this ARN value again when you create the AWS IoT rule.
   g. Choose **Create**.
5. Test the function with sample data.

   a. In the **format-high-temp-notification** details page, in the upper-right corner of the page, confirm that **SampleRuleOutput** appears next to the **Test** button. If it doesn't, choose it from the list of available test events.
   b. To send the sample rule output message to your function, choose **Test**.

If the function and the notification both worked, you will get a text message on the phone that subscribed to the notification.

If you didn't get a text message on the phone, check the result of the operation. In the **Function code** panel, in the **Execution result** tab, review the response to find any errors that occurred. Don't continue to the next step until your function can send the notification to your phone.

Create an AWS IoT rule with an AWS Lambda rule action

In this step, you'll use the rule query statement to format the data from the imaginary weather sensor device to send to a Lambda function, which will format and send a text message.

A sample message payload received from the weather devices looks like this:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

In this rule, you'll use the rule query statement to create a message payload for the Lambda function that looks like this:

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

This contains all the information the Lambda function needs to format and send the correct text message.

To create the AWS IoT rule to call a Lambda function

1. Open the **Rules** hub of the AWS IoT console.
2. To start creating your new rule in **Rules**, choose **Create**.
3. In the top part of **Create a rule**:
    a. In **Name**, enter the rule's name, **`wx_friendly_text`**.

       Remember that a rule name must be unique within your AWS Account and Region, and it can't have any spaces. We've used an underscore character in this name to separate the two words in the rule's name.
    b. In **Description**, describe the rule.

       A meaningful description makes it easier to remember what this rule does and why you created it. The description can be as long as needed, so be as detailed as possible.
4. In **Rule query statement** of **Create a rule**:
    a. In **Using SQL version**, select **`2016-03-23`**.
    b. In the **Rule query statement** edit box, enter the statement:

```
SELECT
  cast(topic(2) AS DECIMAL) as device_id,
  temperature as reported_temperature,
  30 as max_temperature,
  'arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice' as notify_topic_arn
FROM 'device/+/data' WHERE temperature > 30
```

       This statement:
       - Listens for MQTT messages with a topic that matches the `device/+/data` topic filter and that have a `temperature` value greater than 75.
       - Selects the second element from the topic string, converts it to a decimal number, and then assigns it to the `device_id` field.
       - Selects the value of the `temperature` field from the message payload and assigns it to the `reported_temperature` field.
       - Creates a constant value, `30`, to represent the limit value and assigns it to the `max_temperature` field.
       - Creates a constant value for the `notify_topic_arn` field.
    c. Refer to the window that has the **ARN** of the **high_temp_notice** notification topic and copy the ARN value.
    d. Replace the ARN value (*`arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice`*) in the rule query statement editor with the ARN of your notification topic.
5. In **Set one or more actions**:
    a. To open up the list of rule actions for this rule, choose **Add action**.

   b. In **Select an action**, choose **Send a message to a Lambda function**.
   c. To open the selected action's configuration page, at the bottom of the action list, choose **Configure action**.

6. In **Configure action**:

   a. In **Function name**, choose **Select**.
   b. Choose **format-high-temp-notification**.
   c. At the bottom of **Configure action**, choose **Add action**.
   d. To create the rule, at the bottom of **Create a rule**, choose **Create rule**.

Test the AWS IoT rule and AWS Lambda rule action

To test your new rule, you'll use the MQTT client to publish and subscribe to the MQTT messages used by this rule.

Open the MQTT client in the AWS IoT console in a new window. Now you can edit the rule without losing the configuration of your MQTT client. If you leave the MQTT client to go to another page in the console, you'll lose your subscriptions or message logs.

To use the MQTT client to test your rule

1. In the MQTT client in the AWS IoT console, subscribe to the input topics, in this case, `device/+/data`.

   a. In the MQTT client, under **Subscriptions**, choose **Subscribe to a topic**.
   b. In **Subscription topic**, enter the topic of the input topic filter, **`device/+/data`**.
   c. Keep the rest of the fields at their default settings.
   d. Choose **Subscribe to topic**.

      In the **Subscriptions** column, under **Publish to a topic**, **`device/+/data`** appears.

2. Publish a message to the input topic with a specific device ID, **`device/32/data`**. You can't publish to MQTT topics that contain wildcard characters.

   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.
   b. In the **Publish** field, enter the input topic name, **`device/32/data`**.
   c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. To publish your MQTT message, choose **Publish to topic**.

3. Confirm that the text message was sent.

   a. In the MQTT client, under **Subscriptions**, there is a green dot next to the topic to which you subscribed earlier.

The green dot indicates that one or more new messages have been received since the last time you looked at them.

b. Under **Subscriptions**, choose **device/+/data** to check that the message payload matches what you just published and looks like this:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. Check the phone that you used to subscribe to the SNS topic and confirm the contents of the message payload look like this:

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

If you change the topic ID element in the message topic, remember that casting the `topic(2)` value to a numeric value will only work if that element in the message topic contains only numeric characters.

4. Try sending an MQTT message in which the temperature does not exceed the limit.

   a. In the MQTT client, under **Subscriptions**, choose **Publish to topic**.

   b. In the **Publish** field, enter the input topic name, **`device/33/data`**.

   c. Copy the sample data shown here and, in the edit box below the topic name, paste the sample data.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. To send your MQTT message, choose **Publish to topic**.

You should see the message that you sent in the **`device/+/data`** subscription; however, because the temperature value is below the max temperature in the rule query statement, you shouldn't receive a text message.

If you don't see the correct behavior, check the troubleshooting tips.

Troubleshooting your AWS Lambda rule and notification

Here are some things to check, in case you're not seeing the results you expect.

- **You got an error banner**

  If an error appeared when you published the input message, correct that error first. The following steps might help you correct that error.

- **You don't see the input message in the MQTT client**

  Every time you publish your input message to the `device/32/data` topic, that message should appear in the MQTT client, if you subscribed to the `device/+/data` topic filter as described in the procedure.

  Things to check

  - **Check the topic filter you subscribed to**

    If you subscribed to the input message topic as described in the procedure, you should see a copy of the input message every time you publish it.

    If you don't see the message, check the topic name you subscribed to and compare it to the topic to which you published. Topic names are case sensitive and the topic to which you subscribed must be identical to the topic to which you published the message payload.
  - **Check the message publish function**

    In the MQTT client, under **Subscriptions**, choose **device/+/data**, check the topic of the publish message, and then choose **Publish to topic**. You should see the message payload from the edit box below the topic appear in the message list.
- **You don't receive an SMS message**

  For your rule to work, it must have the correct policy that authorizes it to receive a message and send an SNS notification, and it must receive the message.

  Things to check

  - **Check the AWS Region of your MQTT client and the rule that you created**

    The console in which you're running the MQTT client must be in the same AWS Region as the rule you created.
  - **Check that the temperature value in the message payload exceeds the test threshold**

    If the temperature value is less than or equal to 30, as defined in the rule query statement, the rule will not perform any of its actions.
  - **Check the input message topic in the rule query statement**

    For the rule to work, it must receive a message with the topic name that matches the topic filter in the FROM clause of the rule query statement.

    Check the spelling of the topic filter in the rule query statement with that of the topic in the MQTT client. Topic names are case sensitive and the message's topic must match the topic filter in the rule query statement.
  - **Check the contents of the input message payload**

    For the rule to work, it must find the data field in the message payload that is declared in the SELECT statement.

    Check the spelling of the `temperature` field in the rule query statement with that of the message payload in the MQTT client. Field names are case sensitive and the `temperature` field in the rule query statement must be identical to the `temperature` field in the message payload.

    Make sure that the JSON document in the message payload is correctly formatted. If the JSON has any errors, such as a missing comma, the rule will not be able to read it.
  - **Check the Amazon SNS notification**

In Create an Amazon SNS topic that sends an SMS text message (p. 142), refer to step 3 that describes how to test the Amazon SNS notification and test the notification to make sure the notification works.
- **Check the Lambda function**

  In Create an AWS Lambda function that sends a text message (p. 155), refer to step 5 that describes how to test the Lambda function using test data and test the Lambda function.
- **Check the role being used by the rule**

  The rule action must have permission to receive the original topic and publish the new topic.

  The policies that authorize the rule to receive message data and republish it are specific to the topics used. If you change the topic used to republish the message data, you must update the rule action's role to update its policy to match the current topic.

  If you suspect this is the problem, edit the Republish rule action and create a new role. New roles created by the rule action receive the authorizations necessary to perform these actions.

Review the results and next steps

In this tutorial:

- You created an AWS IoT rule to call a Lambda function that sent an Amazon SNS notification that used your customized message payload.
- You used a simple SQL query and functions in a rule query statement to create a new message payload for your Lambda function.
- You used the MQTT client to test your AWS IoT rule.

Next steps

After you send a few text messages with this rule, try experimenting with it to see how changing some aspects of the tutorial affect the message and when it's sent. Here are some ideas to get you started.

- Change the *`device_id`* in the input message's topic and observe the effect in the text message contents.
- Change the fields selected in the rule query statement, update the Lambda function to use them in a new message, and observe the effect in the text message contents.
- Change the test in the rule query statement to test for a minimum temperature instead of a maximum temperature. Update the Lambda function to format a new message and remember to change the name of `max_temperature`.
- To learn more about how to find errors that might occur while you're developing and using AWS IoT rules, see Monitoring AWS IoT (p. 310).

Other AWS IoT tutorials

The tutorials in this section show you how to use multiple AWS IoT services together to accomplish a task. These tutorials focus more on the integration of the services rather than a thorough walkthrough of AWS IoT features. The tutorials in this section might build on one another to show how you can start small and evolve your solutions as your business needs change or grow.

Each tutorial includes a list of prerequisites, including specific hardware. Where possible, the tutorials provide alternatives if you don't have all of the required hardware.

Monitoring soil moisture with AWS IoT and Raspberry Pi

This tutorial shows you how to use a Raspberry Pi, a moisture sensor, and AWS IoT to monitor the soil moisture level for a house plant or garden. The Raspberry Pi runs code that reads the moisture level and temperature from the sensor and then sends the data to AWS IoT. You create a rule in AWS IoT that sends an email to an address subscribed to an Amazon SNS topic when the moisture level falls below a threshold.

Contents

Prerequisites

To complete this tutorial, you need:

- An AWS account.
- An IAM user with administrator permissions.
- A development computer running Windows, macOS, Linux, or Unix to access the AWS IoT console.
- A Raspberry Pi 3B or 4B running the latest Raspbian OS. For installation instructions, see Installing operating system images on the Rasberry Pi website.
- A monitor, keyboard, mouse, and Wi-Fi network or Ethernet connection for your Raspberry Pi.
- A Raspberry Pi-compatible moisture sensor. The sensor used in this tutorial is an Adafruit STEMMA I2C Capacitive Moisture Sensor with a JST 4-pin to female socket cable header.

Setting up AWS IoT

To complete this tutorial, you need to create the following resources. To connect a device to AWS IoT, you create an IoT thing, a device certificate, and an AWS IoT policy.

- An AWS IoT thing.

  A thing represents a physical device (in this case, your Rasberry Pi) and contains static metadata about the device.
- A device certificate.

  All devices must have a device certificate to connect to and authenticate with AWS IoT.
- An AWS IoT policy.

  Each device certificate has one or more AWS IoT policies associated with it. These policies determine which AWS IoT resources the device can access.
- An AWS IoT root CA certificate.

  Devices and other clients use an AWS IoT root CA certificate to authenticate the AWS IoT server with which they are communicating. For more information, see Server authentication (p. 201).

- An AWS IoT rule.

  A rule contains a query and one or more rule actions. The query extracts data from device messages to determine if the message data should be processed. The rule action specifies what to do if the data matches the query.
- An Amazon SNS topic and topic subscription.

  The rule listens for moisture data from your Raspberry Pi. If the value is below a threshold, it sends a message to the Amazon SNS topic. Amazon SNS sends that message to all email addresses subscribed to the topic.

Create the AWS IoT policy

Create an AWS IoT policy that allows your Raspberry Pi to connect and send messages to AWS IoT.

1. In the AWS IoT console, if a **Get started** button appears, choose it. Otherwise, in the navigation pane, expand **Secure**, and then choose **Policies**.
2. If a **You don't have any policies yet** dialog box appears, choose **Create a policy**. Otherwise, choose **Create**.
3. Enter a name for the AWS IoT policy (for example, `MoistureSensorPolicy`).
4. In the **Add statements** section, replace the existing policy with the following JSON. Replace *region* and *account* with your AWS Region and AWS account number.

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "iot:Connect",
         "Resource": "arn:aws:iot:region:account:client/RaspberryPi"
      },
      {
         "Effect": "Allow",
         "Action": "iot:Publish",
         "Resource": [
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/update",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/delete",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/get"
         ]
      },
      {
         "Effect": "Allow",
         "Action": "iot:Receive",
         "Resource": [
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/update/
accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/delete/
accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/get/
accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/update/
rejected",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/delete/
rejected"
         ]
      },
      {
         "Effect": "Allow",
         "Action": "iot:Subscribe",
         "Resource": [
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
update/accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
delete/accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/get/
accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
update/rejected",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
delete/rejected"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "iot:DeleteThingShadow"
         ],
         "Resource": "arn:aws:iot:region:account:thing/RaspberryPi"

      }
   ]
}
```

5. Choose **Create**.

Create the AWS IoT thing, certificate, and private key

Create a thing in the AWS IoT registry to represent your Raspberry Pi.

1. In the AWS IoT console, in the navigation pane, choose **Manage**, and then choose **Things**.
2. If a **You don't have any things yet** dialog box is displayed, choose **Register a thing**. Otherwise, choose **Create**.
3. On the **Creating AWS IoT things** page, choose **Create a single thing**.
4. On the **Add your device to the device registry** page, enter a name for your IoT thing (for example, **`RaspberryPi`**), and then choose **Next**. You can't change the name of a thing after you create it. To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.
5. On the **Add a certificate for your thing** page, choose **Create certificate**.
6. Choose the **Download** links to download the certificate, private key, and root CA certificate.

   Important
   This is the only time you can download your certificate and private key.
7. Choose **Activate**.
8. Choose **Attach a policy**.
9. For **Add a policy for your thing**, choose **MoistureSensorPolicy**, and then choose **Register Thing**.

Create an Amazon SNS topic and subscription

Create an Amazon SNS topic and subscription.

1. From the AWS SNS console, in the navigation pane, choose **Topics**, and then choose **Create topic**.
2. Enter a name for the topic (for example, **`MoistureSensorTopic`**).
3. Enter a display name for the topic (for example, **`Moisture Sensor Topic`**). This is the name displayed for your topic in the Amazon SNS console.
4. Choose **Create topic**.

5. In the Amazon SNS topic detail page, choose **Create subscription**.
6. For **Protocol**, choose **Email**.
7. For **Endpoint**, enter your email address.
8. Choose **Create subscription**.
9. Open your email client and look for a message with the subject **`MoistureSensorTopic`**. Open the email and click the **Confirm subscription** link.

   Important
   You won't receive any email alerts from this Amazon SNS topic until you confirm the subscription.

You should receive an email message with the text you typed.

Create an AWS IoT rule to send an email

An AWS IoT rule defines a query and one or more actions to take when a message is received from a device. The AWS IoT rules engine listens for messages sent by devices and uses the data in the messages to determine if some action should be taken. For more information, see Rules for AWS IoT (p. 352).

In this tutorial, your Raspberry Pi publishes messages on `aws/things/RaspberryPi/shadow/update`. This is an internal MQTT topic used by devices and the Thing Shadow service. The Raspberry Pi publishes messages that have the following form:

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
    }
}
```

You create a query that extracts the moisture and temperature data from the incoming message. You also create an Amazon SNS action that takes the data and sends it to Amazon SNS topic subscribers if the moisture reading is below a threshold value.

Create an Amazon SNS rule

1. In the AWS IoT console, in the navigation pane, choose **Act**. If a **You don't have any rules yet** dialog box appears, choose **Create a rule**. Otherwise, choose **Create**.
2. In the **Create a rule** page, enter a name for your rule (for example, **`MoistureSensorRule`**).
3. For **Description**, provide a short description for this rule (for example, **`Sends an alert when soil moisture level readings are too low`**).
4. Under **Rule query statement**, choose SQL version **2016-03-23**, and enter the following AWS IoT SQL query statement:

   ```
   SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
    state.reported.moisture < 400
   ```

   This statement triggers the rule action when the `moisture` reading is less than `400`.

   Note
   You might have to use a different value. After you have the code running on your Raspberry Pi, you can see the values you get from your sensor by touching the sensor, placing it in water, or placing it in a planter.
5. Under **Set one or more actions**, choose **Add action**.

6. On the **Select an action** page, choose **Send a message as an SNS push notification**.
7. Scroll to the bottom of the page, and then choose **Configure action**.
8. On the **Configure action** page, for **SNS target**, choose **Select**, and then choose **LowMoistureTopic**.
9. For **Message format**, choose **RAW**.
10. Under **Choose or create a role to grant AWS IoT access to perform this action**, choose **Create role**. Enter a name for the role (for example, **`LowMoistureTopicRole`**), and then choose **Create role**.
11. Choose **Add action**.
12. Choose **Create rule**.

Setting up your Raspberry Pi and moisture sensor

Insert your micro SD card into the Raspberry Pi, connect your monitor, keyboard, mouse, and, if you're not using Wi-Fi, Ethernet cable. Do not connect the power cable yet.

Connect the JST jumper cable to the moisture sensor. The other side of the jumper has four wires:

- Green: I2C SCL
- White: I2C SDA
- Red: power (3.5 V)
- Black: ground

Hold the Raspberry Pi with the Ethernet jack on the right. In this orientation, there are two rows of GPIO pins at the top. Connect the wires from the moisture sensor to the bottom row of pins in the following order. Starting at the left-most pin, connect red (power), white (SDA), and green (SCL). Skip one pin, and then connect the black (ground) wire. For more information, see Python Computer Wiring.

Attach the power cable to the Raspberry Pi and plug the other end into a wall socket to turn it on.

Configure your Raspberry Pi

1. On **Welcome to Raspberry Pi**, choose **Next**.
2. Choose your country, language, timezone, and keyboard layout. Choose **Next**.
3. Enter a password for your Raspberry Pi, and then choose **Next**.
4. Choose your Wi-Fi network, and then choose **Next**. If you aren't using a Wi-Fi network, choose **Skip**.
5. Choose **Next** to check for software updates. When the updates are complete, choose **Restart** to restart your Raspberry Pi.

After your Raspberry Pi starts up, enable the I2C interface.

1. In the upper left corner of the Raspbian desktop, click the Raspberry icon, choose **Preferences**, and then choose **Raspberry Pi Configuration**.
2. On the **Interfaces** tab, for **I2C**, choose **Enable**.
3. Choose **OK**.

The libraries for the Adafruit STEMMA moisture sensor are written for CircuitPython. To run them on a Raspberry Pi, you need to install the latest version of Python 3.

1. Run the following commands from a command prompt to update your Raspberry Pi software:

   ```
   sudo apt-get update
   ```

```
sudo apt-get upgrade
```

2. Run the following command to update your Python 3 installation:

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. Run the following command to install the Raspberry Pi GPIO libraries:

   ```
   pip3 install RPI.GPIO
   ```
4. Run the following command to install the Adafruit Blinka libraries:

   ```
   pip3 install adafruit-blinka
   ```

   For more information, see Installing CircuitPython Libraries on Raspberry Pi.
5. Run the following command to install the Adafruit Seesaw libraries:

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```
6. Run the following command to install the AWS IoT Device SDK for Python:

   ```
   pip3 install AWSIoTPythonSDK
   ```

Your Raspberry Pi now has all of the required libraries. Create a file called **`moistureSensor.py`** and copy the following Python code into the file:

```
from adafruit_seesaw.seesaw import Seesaw
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from board import SCL, SDA

import logging
import time
import json
import argparse
import busio

# Shadow JSON schema:
#
# {
#   "state": {
#       "desired":{
#           "moisture":<INT VALUE>,
#           "temp":<INT VALUE>
#       }
#   }
# }

# Function called when a shadow is updated
def customShadowCallback_Update(payload, responseStatus, token):

    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
```

```
        print("Update request " + token + " rejected!")

# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):

     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")

    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")


# Read in command-line parameters
def parseArgs():

    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
 help="Your AWS IoT custom endpoint")
    parser.add_argument("-r", "--rootCA", action="store", required=True, dest="rootCAPath",
 help="Root CA file path")
    parser.add_argument("-c", "--cert", action="store", dest="certificatePath",
 help="Certificate file path")
    parser.add_argument("-k", "--key", action="store", dest="privateKeyPath", help="Private
 key file path")
    parser.add_argument("-p", "--port", action="store", dest="port", type=int, help="Port
 number override")
    parser.add_argument("-n", "--thingName", action="store", dest="thingName",
 default="Bot", help="Targeted thing name")
    parser.add_argument("-id", "--clientId", action="store", dest="clientId",
 default="basicShadowUpdater", help="Targeted client id")

    args = parser.parse_args()
    return args


# Configure logging
# AWSIoTMQTTShadowClient writes data to the log
def configureLogging():

    logger = logging.getLogger("AWSIoTPythonSDK.core")
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)


# Parse command line arguments
args = parseArgs()

if not args.certificatePath or not args.privateKeyPath:
    parser.error("Missing credentials for authentication.")
    exit(2)

# If no --port argument is passed, default to 8883
if not args.port:
    args.port = 8883


# Init AWSIoTMQTTShadowClient
```

```
myAWSIoTMQTTShadowClient = None
myAWSIoTMQTTShadowClient = AWSIoTMQTTShadowClient(args.clientId)
myAWSIoTMQTTShadowClient.configureEndpoint(args.host, args.port)
myAWSIoTMQTTShadowClient.configureCredentials(args.rootCAPath, args.privateKeyPath,
 args.certificatePath)

# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTShadowClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTShadowClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTShadowClient.configureMQTTOperationTimeout(5) # 5 sec

# Initialize Raspberry Pi's I2C interface
i2c_bus = busio.I2C(SCL, SDA)

# Intialize SeeSaw, Adafruit's Circuit Python library
ss = Seesaw(i2c_bus, addr=0x36)

# Connect to AWS IoT
myAWSIoTMQTTShadowClient.connect()

# Create a device shadow handler, use this to update and delete shadow document
deviceShadowHandler = myAWSIoTMQTTShadowClient.createShadowHandlerWithName(args.thingName,
 True)

# Delete current shadow JSON doc
deviceShadowHandler.shadowDelete(customShadowCallback_Delete, 5)

# Read data from moisture sensor and update shadow
while True:

    # read moisture level through capacitive touch pad
    moistureLevel = ss.moisture_read()

    # read temperature from the temperature sensor
    temp = ss.get_temp()

    # Display moisture and temp readings
    print("Moisture Level: {}".format(moistureLevel))
    print("Temperature: {}".format(temp))

    # Create message payload
    payload = {"state":{"reported":{"moisture":str(moistureLevel),"temp":str(temp)}}}

    # Update shadow
    deviceShadowHandler.shadowUpdate(json.dumps(payload), customShadowCallback_Update, 5)
    time.sleep(1)
```

Save the file to a place you can find it. Run `moistureSensor.py` from the command line with the following parameters:

endpoint

Your custom AWS IoT endpoint. For more information, see Device Shadow REST API (p. 515).

rootCA

The full path the your AWS IoT root CA certificate.

cert

The full path to your AWS IoT device certificate.

key

The full path to your AWS IoT device certificate private key.

thingName

Your thing name (in this case, `RaspberryPi`).

clientId

The MQTT client ID. Use `RaspberryPi`.

The command line should look like this:

```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key ~/certs/
raspberrypi-private.pem.key --thingName RaspberryPi --clientId RaspberryPi
```

Try touching the sensor, putting it in a planter, or putting it in a glass of water to see how the sensor responds to various levels of moisture. If needed, you can change the threshold value in the `MoistureSensorRule`. When the moisture sensor reading goes below the value specified in your rule's SQL query statement, AWS IoT publishes a message to the Amazon SNS topic. You should receive an email message that contains the moisture and temperature data.

After you have verified receipt of email messages from Amazon SNS, press **CTRL+C** to stop the Python program. It is unlikely the Python program will send enough messages to incur charges, but it is a best practice to stop the program when you are done.

Managing devices with AWS IoT

AWS IoT provides a registry that helps you manage *things*. A thing is a representation of a specific device or logical entity. It can be a physical device or sensor (for example, a light bulb or a switch on a wall). It can also be a logical entity like an instance of an application or physical entity that does not connect to AWS IoT but is related to other devices that do (for example, a car that has engine sensors or a control panel).

Information about a thing is stored in the registry as JSON data. Here is an example thing:

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

Things are identified by a name. Things can also have attributes, which are name-value pairs you can use to store information about the thing, such as its serial number or manufacturer.

A typical device use case involves the use of the thing name as the default MQTT client ID. Although we do not enforce a mapping between a thing's registry name and its use of MQTT client IDs, certificates, or shadow state, we recommend you choose a thing name and use it as the MQTT client ID for both the registry and the Device Shadow service. This provides organization and convenience to your IoT fleet without removing the flexibility of the underlying device certificate model or shadows.

You do not need to create a thing in the registry to connect a device to AWS IoT. Adding things to the registry allows you to manage and search for devices more easily.

How to manage things with the registry

You use the AWS IoT console or the AWS CLI to interact with the registry. The following sections show how to use the CLI to work with the registry.

When naming your thing objects:

- You should not use personally identifiable information in your thing name. The thing name can appear in unencrypted communications and reports.
- You should not use a colon character (:) in a thing name. The colon character is used as a delimiter by other AWS IoT services and this can cause them to parse strings with thing names incorrectly.

Create a thing

The following command shows how to use the AWS IoT **CreateThing** command from the CLI to create a thing. You can't change a thing's name after you create it. To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

```
$ aws iot create-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

The **CreateThing** command displays the name and Amazon Resource Name (ARN) of your new thing:

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

Note
We do not recommend using personally identifiable information in your thing names.

List things

You can use the **ListThings** command to list all things in your account:

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
            "thingName": "MyWallSwitch"
        }
    ]
}
```

You can use the **ListThings** command to search for all things of a specific thing type:

```
$  aws iot list-things --thing-type-name "LightBulb"
```

```
{
    "things": [
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

You can use the **ListThings** command to search for all things that have an attribute with a specific value. This command searches only searchable attributes.

```
$  aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```

{
    "things": [
        {
            "thingTypeName": "StopLight",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3,
            "thingName": "MyLightBulb"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

If fleet indexing (p. 684) is enabled, you can use the **search-index** command to search on searchable and non-searchable thing attributes, device shadow values, and connectivity values. For more information about what you can query by using the **search-index** command, see Example thing queries (p. 701) and the CLI reference on the **search-index** command.

Describe things

You can use the **DescribeThing** command to display more detailed information about a thing:

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "version": 3,
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "StopLight",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

Update a thing

You can use the **UpdateThing** command to update a thing. Note that this command updates only the thing's attributes. You can't change a thing's name. To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

The **UpdateThing** command does not produce output. You can use the **DescribeThing** command to see the result:

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

Delete a thing

You can use the **DeleteThing** command to delete a thing:

```
$ aws iot delete-thing --thing-name "MyThing"
```

This command returns successfully with no error if the deletion is successful or you specify a thing that doesn't exist.

Attach a principal to a thing

A physical device must have an X.509 certificate to communicate with AWS IoT. You can associate the certificate on your device with the thing in the registry that represents your device. To attach a certificate to your thing, use the **AttachThingPrincipal** command:

```
$ aws iot attach-thing-principal --thing-name "MyLightBulb" --principal "arn:aws:iot:us-
east-1:123456789012:cert/a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

The **AttachThingPrincipal** command does not produce any output.

Detach a principal from a thing

You can use the **DetachThingPrincipal** command to detach a certificate from a thing:

```
$ aws iot detach-thing-principal --thing-name "MyLightBulb" --principal "arn:aws:iot:us-
east-1:123456789012:cert/a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

The **DetachThingPrincipal** command does not produce any output.

Thing types

Thing types allow you to store description and configuration information that is common to all things associated with the same thing type. This simplifies the management of things in the registry. For example, you can define a LightBulb thing type. All things associated with the LightBulb thing type share a set of attributes: serial number, manufacturer, and wattage. When you create a thing of type LightBulb (or change the type of an existing thing to LightBulb) you can specify values for each of the attributes defined in the LightBulb thing type.

Although thing types are optional, their use makes it easier to discover things.

- Things with a thing type can have up to 50 attributes.
- Things without a thing type can have up to three attributes.
- A thing can be associated with only one thing type.
- There is no limit on the number of thing types you can create in your account.

Thing types are immutable. You cannot change a thing type name after it has been created. You can deprecate a thing type at any time to prevent new things from being associated with it. You can also delete thing types that have no things associated with them.

Create a thing type

You can use the **CreateThingType** command to create a thing type:

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

The **CreateThingType** command returns a response that contains the thing type and its ARN:

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

List thing types

You can use the **ListThingTypes** command to list thing types:

```
$ aws iot list-thing-types
```

The **ListThingTypes** command returns a list of the thing types defined in your AWS account:

```
{
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
                "thingTypeDescription": "light bulb type"
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

Describe a thing type

You can use the **DescribeThingType** command to get information about a thing type:

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

The **DescribeThingType** command returns information about the specified type:

```
{
    "thingTypeProperties": {
        "searchableAttributes": [
            "model",
            "wattage"
        ],
        "thingTypeDescription": "light bulb type"
    },
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
    "thingTypeName": "LightBulb",
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1544466338.399
    }
}
```

Associate a thing type with a thing

You can use the **CreateThing** command to specify a thing type when you create a thing:

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

You can use the **UpdateThing** command at any time to change the thing type associated with a thing:

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

You can also use the **UpdateThing** command to disassociate a thing from a thing type.

Deprecate a thing type

Thing types are immutable. They cannot be changed after they are defined. You can, however, deprecate a thing type to prevent users from associating any new things with it. All existing things associated with the thing type are unchanged.

To deprecate a thing type, use the **DeprecateThingType** command:

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

You can use the **DescribeThingType** command to see the result:

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
    },
    "thingTypeMetadata": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "deprecationDate": 1468446026349
    }
}
```

Deprecating a thing type is a reversible operation. You can undo a deprecation by using the `--undo-deprecate` flag with the **DeprecateThingType** CLI command:

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

You can use the **DescribeThingType** CLI command to see the result:

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/StopLight",
    "thingTypeId": "12345678abcdefgh12345678ijklmnop12345678"
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type"
    },
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1468425854308,
    }
}
```

Delete a thing type

You can delete thing types only after they have been deprecated. To delete a thing type, use the **DeleteThingType** command:

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

Note
You must wait five minutes after you deprecate a thing type before you can delete it.

Static thing groups

Static thing groups allow you to manage several things at once by categorizing them into groups. Static thing groups contain a group of things that are managed by using the console, CLI, or the API. Dynamic thing groups (p. 188), on the other hand, contain things that match a specified query. Static thing groups can also contain other static thing groups — you can build a hierarchy of groups. You can attach a policy to a parent group and it is inherited by its child groups, and by all of the things in the group and in its child groups. This makes control of permissions easy for large numbers of things.

Here are the things you can do with static thing groups:

- Create, describe or delete a group.
- Add a thing to a group, or to more than one group.
- Remove a thing from a group.
- List the groups you have created.
- List all child groups of a group (its direct and indirect descendants.)
- List the things in a group, including all the things in its child groups.
- List all ancestor groups of a group (its direct and indirect parents.)
- Add, delete or update the attributes of a group. (Attributes are name-value pairs you can use to store information about a group.)
- Attach or detach a policy to or from a group.
- List the policies attached to a group.
- List the policies inherited by a thing (by virtue of the policies attached to its group, or one of its parent groups.)
- Configure logging options for things in a group. See Configure AWS IoT logging (p. 310).
- Create jobs that are sent to and executed on every thing in a group and its child groups. See Jobs (p. 537).

Here are some limitations of static thing groups:

- A group can have at most one direct parent.
- If a group is a child of another group, you must specify this at the time it is created.
- You can't change a group's parent later, so be sure to plan your group hierarchy and create a parent group before you create any child groups it contains.
- The number of groups to which a thing can belong is limited.

- You cannot add a thing to more than one group in the same hierarchy. (In other words, you cannot add a thing to two groups that share a common parent.)
- You cannot rename a group.
- Thing group names can't contain international characters, such as û, é and ñ.
- You should not use personally identifiable information in your thing group name. The thing group name can appear in unencrypted communications and reports.
- You should not use a colon character (:) in a thing group name. The colon character is used as a delimiter by other AWS IoT services and this can cause them to parse strings with thing group names incorrectly.

Attaching and detaching policies to groups can enhance the security of your AWS IoT operations in a number of significant ways. The per-device method of attaching a policy to a certificate, which is then attached to a thing, is time consuming and makes it difficult to quickly update or change policies across a fleet of devices. Having a policy attached to the thing's group saves steps when it is time to rotate the certificates on a thing. And policies are dynamically applied to things when they change group membership, so you aren't required to re-create a complex set of permissions each time a device changes membership in a group.

Create a static thing group

Use the **CreateThingGroup** command to create a static thing group:

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

The **CreateThingGroup** command returns a response that contains the static thing group's name, ID, and ARN:

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

Note
We do not recommend using personally identifiable information in your thing group names.

Here is an example that specifies a parent of the static thing group when it is created:

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name LightBulbs
```

As before, the **CreateThingGroup** command returns a response that contains the static thing group's name,, ID, and ARN:

```
{
    "thingGroupName": "RedLights",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

Important
Keep in mind the following limits when creating thing group hierarchies:

- A thing group can have only one direct parent.
- The number of direct child groups a thing group can have is limited.

- The maximum depth of a group hierarchy is limited.
- The number of attributes a thing group can have is limited. (Attributes are name-value pairs you can use to store information about a group.) The lengths of each attribute name and each value are also limited.

Describe a thing group

You can use the **DescribeThingGroup** command to get information about a thing group:

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

The **DescribeThingGroup** command returns information about the specified group:

```
{
    "thingGroupName": "RedLights",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
    "thingGroupId": "12345678abcdefgh12345678ijklmnop12345678",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1478299948.882
        "parentGroupName": "Lights",
        "rootToParentThingGroups": [
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/ShinyObjects",
                "groupName": "ShinyObjects"
            },
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs",
                "groupName": "LightBulbs"
            }
        ]
    },
    "thingGroupProperties": {
        "attributePayload": {
            "attributes": {
                "brightness": "3400_lumens"
            },
        },
        "thingGroupDescription": "string"
    },
}
```

Add a thing to a static thing group

You can use the **AddThingToThingGroup** command to add a thing to a static thing group:

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name RedLights
```

The **AddThingToThingGroup** command does not produce any output.

Important
You can add a thing to a maximum of 10 groups. But you cannot add a thing to more than one group in the same hierarchy. (In other words, you cannot add a thing to two groups which share a common parent.)
If a thing belongs to as many thing groups as possible, and one or more of those groups is a dynamic thing group, you can use the **overrideDynamicGroups** flag to make static groups take priority over dynamic groups.

Remove a thing from a static thing group

You can use the **RemoveThingFromThingGroup** command to remove a thing from a group:

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

The **RemoveThingFromThingGroup** command does not produce any output.

List things in a thing group

You can use the **ListThingsInThingGroup** command to list the things that belong to a group:

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

The **ListThingsInThingGroup** command returns a list of the things in the given group:

```
{
    "things":[
        "TestThingA"
    ]
}
```

With the `--recursive` parameter, you can list things belonging to a group and those in any of its child groups:

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

Note
This operation is eventually consistent. In other words, changes to the thing group might not be reflected immediately.

List thing groups

You can use the **ListThingGroups** command to list your account's thing groups:

```
$ aws iot list-thing-groups
```

The **ListThingGroups** command returns a list of the thing groups in your AWS account:

```
{
    "thingGroups": [
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/ReplaceableObjects"
        }
    ]
}
```

Use the optional filters to list those groups that have a given group as parent (`--parent-group`) or groups whose name begins with a given prefix (`--name-prefix-filter`.) The `--recursive` parameter allows you to list all children groups, not just direct child groups of a thing group:

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

In this case, the **ListThingGroups** command returns a list of the direct child groups of the thing group defined in your AWS account:

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

Use the `--recursive` parameter with the **ListThingGroups** command to list all child groups of a thing group, not just direct children:

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

The **ListThingGroups** command returns a list of all child groups of the thing group:

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
    ]
}
```

Note
This operation is eventually consistent. In other words, changes to the thing group might not be reflected immediately.

List groups for a thing

You can use the **ListThingGroupsForThing** command to list the groups a thing belongs to, including any parent groups:

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

The **ListThingGroupsForThing** command returns a list of the thing groups this thing belongs to, including any parent groups:

```
{
    "thingGroups":[
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/ReplaceableObjects"
        }
    ]
}
```

Update a static thing group

You can use the **UpdateThingGroup** command to update the attributes of a static thing group:

```
$ aws iot update-thing-group --thing-group-name "LightBulbs" --thing-group-properties
 "thingGroupDescription=\"this is a test group\", attributePayload=\"{\"attributes
\"={\"Owner\"=\"150\",\"modelNames\"=\"456\"}}"
```

The **UpdateThingGroup** command returns a response that contains the group's version number after the update:

```
{
    "version": 4
}
```

Note
The number of attributes that a thing can have is limited.

Delete a thing group

To delete a thing group, use the **DeleteThingGroup** command:

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

The **DeleteThingGroup** command does not produce any output.

> **Important**
> If you try to delete a thing group that has child thing groups, you receive an error:
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> operation: Cannot delete thing group : RedLights when there are still child groups
>  attached to it.
> ```
>
> You must delete any child groups first before you delete the group.

You can delete a group that has child things, but any permissions granted to the things by membership in the group no longer apply. Before deleting a group that has a policy attached, check carefully that removing those permissions would not stop the things in the group from being able to function properly. Also, note that commands that show which groups a thing belongs to (for example, **ListGroupsForThing**) might continue to show the group while records in the cloud are being updated.

Attach a policy to a static thing group

You can use the **AttachPolicy** command to attach a policy to a static thing group and so, by extension, to all things in that group and things in any of its child groups:

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

The **AttachPolicy** command does not produce any output

> **Important**
> You can attach a maximum number of two policies to a group.

> **Note**
> We do not recommend using personally identifiable information in your policy names.

The `--target` parameter can be a thing group ARN (as above), a certificate ARN, or an Amazon Cognito Identity. For more information about policies, certificates and authentication, see Authentication (p. 200).

Detach a policy from a static thing group

You can use the **DetachPolicy** command to detach a policy from a group and so, by extension, to all things in that group and things in any of its child groups:

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
 --policy-name "myLightBulbPolicy"
```

The **DetachPolicy** command does not produce any output.

List the policies attached to a static thing group

You can use the **ListAttachedPolicies** command to list the policies attached to a static thing group:

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedLights"
```

The `--target` parameter can be a thing group ARN (as above), a certificate ARN, or an Amazon Cognito identity.

Add the optional `--recursive` parameter to include all policies attached to the group's parent groups.

The **ListAttachedPolicies** command returns a list of policies:

```
{
    "policies": [
        "MyLightBulbPolicy"
        ...
    ]
}
```

List the groups for a policy

You can use the **ListTargetsForPolicy** command to list the targets, including any groups, that a policy is attached to:

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

Add the optional `--page-size` *`number`* parameter to specify the maximum number of results to be returned for each query, and the `--marker` *`string`* parameter on subsequent calls to retrieve the next set of results, if any.

The **ListTargetsForPolicy** command returns a list of targets and the token to use to retrieve more results:

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
}
```

Get effective policies for a thing

You can use the **GetEffectivePolicies** command to list the policies in effect for a thing, including the policies attached to any groups the thing belongs to (whether the group is a direct parent or indirect ancestor):

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

Use the `--principal` parameter to specify the ARN of the certificate attached to the thing. If you are using Amazon Cognito identity authentication, use the `--cognito-identity-pool-id` parameter and, optionally, add the `--principal` parameter to specify an Amazon Cognito identity. If you specify only the `--cognito-identity-pool-id`, the policies associated with that identity pool's role for unauthenticated users are returned. If you use both, the policies associated with that identity pool's role for authenticated users are returned.

The `--thing-name` parameter is optional and can be used instead of the `--principal` parameter. When used, the policies attached to any group the thing belongs to, and the policies attached to any parent groups of these groups (up to the root group in the hierarchy) are returned.

The **GetEffectivePolicies** command returns a list of policies:

```
{
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

Test authorization for MQTT actions

You can use the **TestAuthorization** command to test whether an MQTT action is allowed for a thing:

```
aws iot test-authorization \
    --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --auth-infos "{\"actionType\": \"PUBLISH\", \"resources\": [ \"arn:aws:iot:us-
east-1:123456789012:topic/my/topic\"]}"
```

Use the `--principal` parameter to specify the ARN of the certificate attached to the thing. If using Amazon Cognito Identity authentication, specify a Cognito Identity as the `--principal` or use the `--cognito-identity-pool-id` parameter, or both. (If you specify only the `--cognito-identity-pool-id` then the policies associated with that identity pool's role for unauthenticated users are considered. If you use both, the policies associated with that identity pool's role for authenticated users are considered.

Specify one or more MQTT actions you want to test by listing sets of resources and action types following the `--auth-infos` parameter. The `actionType` field should contain "PUBLISH", "SUBSCRIBE", "RECEIVE", or "CONNECT". The `resources` field should contain a list of resource ARNs. See AWS IoT Core policies (p. 235) for more information.

You can test the effects of adding policies by specifying them with the `--policy-names-to-add` parameter. Or you can test the effects of removing policies by them with the `--policy-names-to-skip` parameter.

You can use the optional `--client-id` parameter to further refine your results.

The **TestAuthorization** command returns details on actions that were allowed or denied for each set of `--auth-infos` queries you specified:

```
{
    "authResults": [
        {
            "allowed": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "authDecision": "string",
            "authInfo": {
                "actionType": "string",
                "resources": [ "string" ]
            },
            "denied": {
                "explicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                },
                "implicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                }
            },
            "missingContextValues": [ "string" ]
        }
    ]
}
```

Dynamic thing groups

Dynamic thing groups update group membership through search queries. Using dynamic thing groups, you can change the way you interact with things depending on their connectivity, registry, or shadow data.

Because dynamic thing groups are tied to your fleet index, you must enable the fleet indexing service to use them. You can preview the things in a dynamic thing group before you create the group with a fleet indexing search query. For more information, see Fleet indexing service (p. 684) and Query syntax (p. 700).

You can specify a dynamic thing group as a target for a job. Only things that meet the criteria that define the dynamic thing group perform the job.

For example, suppose that you want to update the firmware on your devices, but, to minimize the chance that the update is interrupted, you only want to update firmware on devices with battery life greater than 80%. You can create a dynamic thing group that only includes devices with a reported battery life above 80%, and you can use that dynamic thing group as the target for your firmware update job. Only devices that meet your battery life criteria receive the firmware update. As devices reach the 80% battery life criteria, they are added to the dynamic thing group and receive the firmware update.

For more information about specifying thing groups as job targets, see CreateJob (p. 569).

Dynamic thing groups differ from static thing groups in the following ways:

- Thing membership is not explicitly defined. To create a dynamic thing group, you must define a query string that defines group membership.
- Dynamic thing groups cannot be part of a hierarchy.
- Dynamic thing groups cannot have policies applied to them.
- You use a different set of commands to create, update, and delete dynamic thing groups. For all other operations, the same commands that you use to interact with static thing groups can be used to interact with dynamic thing groups.
- The number of dynamic groups that a single account can have is limited.

- You should not use personally identifiable information in your thing group name. The thing group name can appear in unencrypted communications and reports.
- You should not use a colon character (:) in a thing group name. The colon character is used as a delimiter by other AWS IoT services and this can cause them to parse strings with thing group names incorrectly.

For more information about static thing groups, see Static thing groups (p. 179).

As an example, suppose we create a dynamic group that contains all rooms in a warehouse whose temperature is greater than 60 degrees Fahrenheit. When a room's temperature is 61 degrees or higher, it is added to the RoomTooWarm dynamic thing group. All rooms in the RoomTooWarm dynamic thing group have cooling fans turned on. When a room's temperature falls to 60 degrees or lower, it is removed from the dynamic thing group and its fan would be turned off.

Create a dynamic thing group

Use the **CreateDynamicThingGroup** command to create a dynamic thing group. To create a dynamic thing group for the room too warm scenario you would use the **create-dynamic-thing-group** CLI command:

```
$ aws iot create-dynamic-thing-group --thing-group-name "RoomTooWarm" --query-string
 "attributes.temperature>60"
```

Note
We do not recommend using personally identifiable information in your dynamic thing group names.

The **CreateDynamicThingGroup** command returns a response that contains the index name, query string, query version, thing group name, thing group ID, and thing group ARN:

```
{
    "indexName": "AWS_Things",
    "queryVersion": "2017-09-30",
    "thingGroupName": "RoomTooWarm",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RoomTooWarm",
    "queryString": "attributes.temperature>60\n",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx"
}
```

Dynamic thing group creation is not instantaneous. The dynamic thing group backfill takes time to complete. When a dynamic thing group is created, the status of the group is set to `BUILDING`. When the backfill is complete, the status changes to `ACTIVE`. To check the status of your dynamic thing group, use the DescribeThingGroup command.

Describe a dynamic thing group

Use the **DescribeThingGroup** command to get information about a dynamic thing group:

```
$ aws iot describe-thing-group --thing-group-name "RoomTooWarm"
```

The **DescribeThingGroup** command returns information about the specified group:

```
{
    "status": "ACTIVE",
    "indexName": "AWS_Things",
    "thingGroupName": "RoomTooWarm",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RoomTooWarm",
    "queryString": "attributes.temperature>60\n",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1548716921.289
    },
    "thingGroupProperties": {},
    "queryVersion": "2017-09-30",
    "thingGroupId": "84dd9b5b-2b98-4c65-84e4-be0e1ecf4fd8"
}
```

Running **DescribeThingGroup** on a dynamic thing group returns attributes that are specific to dynamic thing groups, such as the queryString and the status.

The status of a dynamic thing group can take the following values:

`ACTIVE`

The dynamic thing group is ready for use.

`BUILDING`

The dynamic thing group is being created, and thing membership is being processed.

`REBUILDING`

The dynamic thing group's membership is being updated, following the adjustment of the group's search query.

Note

After you create a dynamic thing group, you can use the group, regardless of its status. Only dynamic thing groups with an `ACTIVE` status include all of the things that match the search query for that dynamic thing group. Dynamic thing groups with `BUILDING` and `REBUILDING` statuses might not include all of the things that match the search query.

Update a dynamic thing group

Use the **UpdateDynamicThingGroup** command to update the attributes of a dynamic thing group, including the group's search query. The following command updates the thing group description and the query string changing the membership criteria to temperature > 65:

```
$ aws iot update-dynamic-thing-group --thing-group-name "RoomTooWarm" --thing-group-
properties "thingGroupDescription=\"This thing group contains rooms warmer than 65F.\"" --
query-string "attributes.temperature>65"
```

The **UpdateDynamicThingGroup** command returns a response that contains the group's version number after the update:

```
{
    "version": 2
}
```

Dynamic thing group updates are not instantaneous. The dynamic thing group backfill takes time to complete. When a dynamic thing group is updated, the status of the group changes to `REBUILDING` while the group updates its membership. When the backfill is complete, the status changes to `ACTIVE`. To check the status of your dynamic thing group, use the DescribeThingGroup command.

Delete a dynamic thing group

Use the **DeleteDynamicThingGroup** command to delete a dynamic thing group:

```
$ aws iot delete-dynamic-thing-group --thing-group-name "RoomTooWarm"
```

The **DeleteDynamicThingGroup** command does not produce any output.

Commands that show which groups a thing belongs to (for example, **ListGroupsForThing**) might continue to show the group while records in the cloud are being updated.

Limitations and conflicts

Dynamic thing groups share these limitations with static thing groups:

- The number of attributes a thing group can have is limited.
- The number of groups to which a thing can belong is limited.
- Thing groups cannot be renamed.
- Thing group names cannot contain international characters, such as û, é, and ñ.

When using dynamic thing groups, keep the following in mind.

The fleet indexing service must be enabled

The fleet indexing service must be enabled and the fleet indexing backfill must be complete before you can create and use dynamic thing groups. Expect a delay after you enable the fleet indexing service. The backfill can take some time to complete. The more things that you have registered, the longer the backfill process takes. After you enable the fleet indexing service for dynamic thing groups, you cannot disable it until you delete all of your dynamic thing groups.

Note
If you have permissions to query the fleet index, you can access the data of things across the entire fleet.

The number of dynamic thing groups is limited

The number of dynamic groups is limited.

Successful commands can log errors

When creating or updating a dynamic thing group, it's possible that some things might be eligible to be in a dynamic thing group yet not be added to it. The command to create or update a dynamic thing group, however, still succeeds in those cases while logging an error and generating an `AddThingToDynamicThingGroupsFailed` metric (p. 322).

An error log entry in the CloudWatch log is created for each thing when an eligible thing cannot be added to a dynamic thing group or a thing is removed from a dynamic thing group to add it to another group. When a thing cannot be added to a dynamic group, an `AddThingToDynamicThingGroupsFailed` metric (p. 322) is also created; however, a single metric can represent multiple log entries.

When a thing becomes eligible to be added to a dynamic thing group, the following is considered:

- Is the thing already in as many groups as it can be? (See limits)

- **NO:** The thing is added to the dynamic thing group.
- **YES:** Is the thing a member of any dynamic thing groups?
  - **NO:** The thing can't be added to the dynamic thing group, an error is logged, and an `AddThingToDynamicThingGroupsFailed` metric (p. 322) is generated.
  - **YES:** Is the dynamic thing group to join older than any dynamic thing group that the thing is already a member of?
    - **NO:** The thing can't be added to the dynamic thing group, an error is logged, and an `AddThingToDynamicThingGroupsFailed` metric (p. 322) is generated.
    - **YES:** Remove the thing from the most recent dynamic thing group it is a member of, log an error, and add the thing to the dynamic thing group. This generates an error and an `AddThingToDynamicThingGroupsFailed` metric (p. 322) for the dynamic thing group from which the thing was removed.

When a thing in a dynamic thing group no longer meets the search query, it is removed from the dynamic thing group. Likewise, when a thing is updated to meet a dynamic thing group's search query, it is then added to the group as previously described. These additions and removals are normal and do not produce error log entries.

With `overrideDynamicGroups` enabled, static groups take priority over dynamic groups

The number of groups to which a thing can belong is limited. When you update thing membership by using the AddThingToThingGroup or UpdateThingGroupsForThing commands, adding the `--overrideDynamicGroups` parameter gives static thing groups priority over dynamic thing groups.

When adding a thing to a static thing group, the following is considered:

- Does the thing already belong to the maximum number of groups?
  - **NO:** The thing is added to the static thing group.
  - **YES:** Is the thing in any dynamic groups?
    - **NO:** The thing cannot be added to the thing group. The command raises an exception.
    - **YES:** Was **--overrideDynamicGroups** enabled?
      - **NO:** The thing cannot be added to the thing group. The command raises an exception.
      - **YES:** The thing is removed from the most recently created dynamic thing group, an error is logged, and an `AddThingToDynamicThingGroupsFailed` metric (p. 322) is generated for the dynamic thing group from which the thing was removed. Then, the thing is added to the static thing group.

Older dynamic thing groups take priority over newer ones

The number of groups to which a thing can belong is limited. When a thing becomes eligible to be added to a dynamic thing group beca a create or update operation, and the thing is already in as many groups as it can be, it can be removed from another dynamic thing group to enable this addition. For more information about how this occurs, see Successful commands can log errors (p. 191) and With `overrideDynamicGroups` enabled, static groups take priority over dynamic groups (p. 192) for examples.

When a thing is removed from a dynamic thing group, an error is logged, and an event is raised.

You cannot apply policies to dynamic thing groups

Attempting to apply a policy to a dynamic thing group generates an exception.

Dynamic thing group membership is eventually consistent

Only the final state of a thing is evaluated for the registry. Intermediary states can be skipped if states are updated rapidly. Avoid associating a rule or job, with a dynamic thing group whose membership depends on an intermediary state.

Tagging your AWS IoT resources

To help you manage and organize your thing groups, thing types, topic rules, jobs, scheduled audits and security profiles you can optionally assign your own metadata to each of these resources in the form of tags. This section describes tags and shows you how to create them.

To help you manage your costs related to things, you can create billing groups (p. 196) that contain things. You can then assign tags that contain your metadata to each of these billing groups. This section also discusses billing groups and the commands available to create and manage them.

Tag basics

You can use tags to categorize your AWS IoT resources in different ways (for example, by purpose, owner, or environment). This is useful when you have many resources of the same type — you can quickly identify a resource based on the tags you've assigned to it. Each tag consists of a key and optional value, both of which you define. For example, you can define a set of tags for your thing types that helps you track devices by type. We recommend that you create a set of tag keys that meets your needs for each kind of resource. Using a consistent set of tag keys makes it easier for you to manage your resources.

You can search for and filter resources based on the tags you add or apply. You can also use billing group tags to categorize and track your costs. You can also use tags to control access to your resources as described in Using tags with IAM policies (p. 195).

For ease of use, the Tag Editor in the AWS Management Console provides a central, unified way to create and manage your tags. For more information, see Working with Tag Editor in Working with the AWS Management Console.

You can also work with tags using the AWS CLI and the AWS IoT API. You can associate tags with thing groups, thing types, topic rules, jobs, security profiles, and billing groups when you create them by using the `Tags` field in the following commands:

- CreateBillingGroup
- CreateDestination
- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate
- CreateScheduledAudit
- CreateSecurityProfile
- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway

You can add, modify, or delete tags for existing resources that support tagging by using the following commands:

- TagResource
- ListTagsForResource
- UntagResource

You can edit tag keys and values, and you can remove tags from a resource at any time. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value. If you delete a resource, any tags associated with the resource are also deleted.

Tag restrictions and limitations

The following basic restrictions apply to tags:

- Maximum number of tags per resource — 50
- Maximum key length — 127 Unicode characters in UTF-8
- Maximum value length — 255 Unicode characters in UTF-8
- Tag keys and values are case sensitive.
- Do not use the "aws:" prefix in your tag names or values. It's reserved for AWS use. You can't edit or delete tag names or values with this prefix. Tags with this prefix don't count against your tags per resource limit.
- If your tagging schema is used across multiple services and resources, remember that other services might have restrictions on allowed characters. Allowed characters include letters, spaces, and numbers representable in UTF-8, and the following special characters: + - = . _ : / @.

Using tags with IAM policies

You can apply tag-based resource-level permissions in the IAM policies you use for AWS IoT API actions. This gives you better control over what resources a user can create, modify, or use. You use the `Condition` element (also called the `Condition` block) with the following condition context keys and values in an IAM policy to control user access (permissions) based on a resource's tags:

- Use `aws:ResourceTag/tag-key: tag-value` to allow or deny user actions on resources with specific tags.
- Use `aws:RequestTag/tag-key: tag-value` to require that a specific tag be used (or not used) when making an API request to create or modify a resource that allows tags.
- Use `aws:TagKeys: [tag-key, ...]` to require that a specific set of tag keys be used (or not used) when making an API request to create or modify a resource that allows tags.

Note
The condition context keys and values in an IAM policy apply only to those AWS IoT actions where an identifier for a resource capable of being tagged is a required parameter. For example, the use of DescribeEndpoint is not allowed or denied on the basis of condition context keys and values because no taggable resource (thing groups, thing types, topic rules, jobs, or security profile) is referenced in this request.

For more information about using tags, see Controlling Access Using Tags in the *AWS Identity and Access Management User Guide*. The IAM JSON Policy Reference section of that guide has detailed syntax, descriptions, and examples of the elements, variables, and evaluation logic of JSON policies in IAM.

The following example policy applies two tag-based restrictions. An IAM user restricted by this policy:

- Cannot give a resource the tag "env=prod" (in the example, see the line `"aws:RequestTag/env" : "prod"`
- Cannot modify or access a resource that has an existing tag "env=prod" (in the example, see the line `"aws:ResourceTag/env" : "prod"`).

```
{
    "Version" : "2012-10-17",
    "Statement" : [
        {
          "Effect" : "Deny",
          "Action" : "iot:*",
          "Resource" : "*",
          "Condition" : {
            "StringEquals" : {
              "aws:RequestTag/env" : "prod"
            }
          }
        },
        {
          "Effect" : "Deny",
          "Action" : "iot:*",
          "Resource" : "*",
          "Condition" : {
            "StringEquals" : {
              "aws:ResourceTag/env" : "prod"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "iot:*"
          ],
          "Resource": "*"
        }
    ]
}
```

You can also specify multiple tag values for a given tag key by enclosing them in a list, like this:

```
            "StringEquals" : {
              "aws:ResourceTag/env" : ["dev", "test"]
            }
```

Note
If you allow or deny users access to resources based on tags, you must consider explicitly denying users the ability to add those tags to or remove them from the same resources. Otherwise, it's possible for a user to circumvent your restrictions and gain access to a resource by modifying its tags.

Billing groups

AWS IoT doesn't allow you to directly apply tags to individual things, but it does allow you to place things in billing groups and to apply tags to these. For AWS IoT, allocation of cost and usage data based on tags is limited to billing groups.

AWS IoT Core for LoRaWAN resources, such as wireless devices and gateways, can't be added to billing groups. However, they can be associated with AWS IoT things, which can be added to billing groups.

The following commands are available:

- AddThingToBillingGroup adds a thing to a billing group.
- CreateBillingGroup creates a billing group.
- DeleteBillingGroup deletes the billing group.
- DescribeBillingGroup returns information about a billing group.
- ListBillingGroups lists the billing groups you have created.
- ListThingsInBillingGroup lists the things you have added to the given billing group.
- RemoveThingFromBillingGroup removes the given thing from the billing group.
- UpdateBillingGroup updates information about the billing group.
- CreateThing allows you to specify a billing group for the thing when you create it.
- DescribeThing returns the description of a thing including the billing group the thing belongs to, if any.

The AWS IoT Wireless API provides these actions to associate wireless devices and gateways with AWS IoT things.

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

Viewing cost allocation and usage data

You can use billing group tags to categorize and track your costs. When you apply tags to billing groups (and so to the things they include), AWS generates a cost allocation report as a comma-separated value (CSV) file with your usage and costs aggregated by your tags. You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. For more information about using tags for cost allocation, see Use Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Note
To accurately associate usage and cost data with those things you have placed in billing groups, each device or application must:

- Be registered as a thing in AWS IoT. For more information, see Managing devices with AWS IoT (p. 172).
- Connect to the AWS IoT message broker through MQTT using only the thing's name as the client ID. For more information, see the section called "Device communication protocols" (p. 76).
- Authenticate using a client certificate associated with the thing.

The following pricing dimensions are available for billing groups (based on the activity of things associated with the billing group):

- Connectivity (based on the thing name used as the client ID to connect).
- Messaging (based on messages inbound from, and outbound to, a thing; MQTT only).
- Shadow operations (based on the thing whose message triggered a shadow update).
- Rules triggered (based on the thing whose inbound message triggered the rule; does not apply to those rules triggered by MQTT lifecycle events).
- Thing index updates (based on the thing that was added to the index).

- Remote actions (based on the thing updated).
- Detect (p. 834) reports (based on the thing whose activity is reported).

Cost and usage data based on tags (and reported for a billing group) doesn't reflect the following activities:

- Device registry operations (including updates to things, thing groups, and thing types). For more information, see Managing devices with AWS IoT (p. 172)).
- Thing group index updates (when adding a thing group).
- Index search queries.
- Device provisioning (p. 658).
- Audit (p. 771) reports.

Security in AWS IoT

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

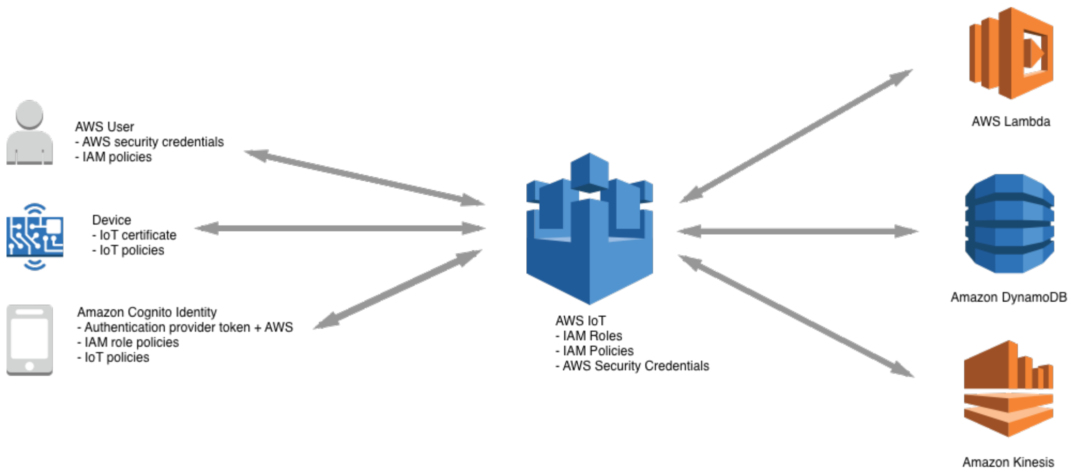
- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS IoT, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS IoT. The following topics show you how to configure AWS IoT to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS IoT resources.

Topics

AWS IoT security

Each connected device or client must have a credential to interact with AWS IoT. All traffic to and from AWS IoT is sent securely over Transport Layer Security (TLS). AWS cloud security mechanisms protect data as it moves between AWS IoT and other AWS services.



- You are responsible for managing device credentials (X.509 certificates, AWS credentials, Amazon Cognito identities, federated identities, or custom authentication tokens) and policies in AWS IoT. For more information, see Key management in AWS IoT (p. 277). You are responsible for assigning unique identities to each device and managing the permissions for each device or group of devices.
- Your devices connect to AWS IoT using X.509 certificates or Amazon Cognito identities over a secure TLS connection. During research and development, and for some applications that make API calls or use WebSockets, you can also authenticate using IAM users and groups or custom authentication tokens. For more information, see IAM users, groups, and roles (p. 221).
- When using AWS IoT authentication, the message broker is responsible for authenticating your devices, securely ingesting device data, and granting or denying access permissions you specify for your devices using AWS IoT policies.
- When using custom authentication, a custom authorizer is responsible for authenticating your devices and granting or denying access permissions you specify for your devices using AWS IoT or IAM policies.
- The AWS IoT rules engine forwards device data to other devices or other AWS services according to rules you define. It uses AWS Identity and Access Management to securely transfer data to its final destination. For more information, see Identity and access management for AWS IoT (p. 277).

Authentication

Authentication is a mechanism where you verify the identity of a client or a server. Server authentication is the process where devices or other clients ensure they are communicating with an actual AWS IoT endpoint. Client authentication is the process where devices or other clients authenticate themselves with AWS IoT.

AWS training and certification

Take the following course to learn about authentication in AWS IoT: Deep Dive into AWS IoT Authentication and Authorization.

X.509 Certificate overview

X.509 certificates are digital certificates that use the X.509 public key infrastructure standard to associate a public key with an identity contained in a certificate. X.509 certificates are issued by a trusted entity called a certification authority (CA). The CA maintains one or more special certificates called CA certificates that it uses to issue X.509 certificates. Only the certification authority has access to

CA certificates. X.509 certificate chains are used both for server authentication by clients and client authentication by the server.

Server authentication

When your device or other client attempts to connect to AWS IoT Core, the AWS IoT Core server will send an X.509 certificate that your device uses to authenticate the server. Authentication takes place at the TLS layer through validation of the X.509 certificate chain (p. 203) This is the same method used by your browser when you visit an HTTPS URL. If you want to use certificates from your own certificate authority, see Manage your CA certificates (p. 207).

When your devices or other clients establish a TLS connection to an AWS IoT Core endpoint, AWS IoT Core presents a certificate chain that the devices use to verify that they're communicating with AWS IoT Core and not another server impersonating AWS IoT Core. The chain that is presented depends on a combination of the type of endpoint the device is connecting to and the cipher suite (p. 275) that the client and AWS IoT Core negotiated during the TLS handshake.

Endpoint types

AWS IoT Core supports two different data endpoint types, `iot:Data` and `iot:Data-ATS`. `iot:Data` endpoints present a certificate signed by the VeriSign Class 3 Public Primary G5 root CA certificate. `iot:Data-ATS` endpoints present a server certificate signed by an Amazon Trust Services CA.

Certificates presented by ATS endpoints are cross signed by Starfield. Some TLS client implementations require validation of the root of trust and require that the Starfield CA certificates are installed in the client's trust stores.

> **Warning**
> Using a method of certificate pinning that hashes the whole certificate (including the issuer name, and so on) is not recommended because this will cause certificate verification to fail because the ATS certificates we provide are cross signed by Starfield and have a different issuer name.

Use `iot:Data-ATS` endpoints unless your device requires Symantec or Verisign CA certificates. Symantec and Verisign certificates have been deprecated and are no longer supported by most web browsers.

You can use the `describe-endpoint` command to create your ATS endpoint.

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

The `describe-endpoint` command returns an endpoint in the following format.

```
account-specific-prefix.iot.your-region.amazonaws.com
```

The first time `describe-endpoint` is called, an endpoint is created. All subsequent calls to `describe-endpoint` return the same endpoint.

For backward-compatibility, AWS IoT Core still supports Symantec endpoints. For more information, see How AWS IoT Core is Helping Customers Navigate the Upcoming Distrust of Symantec Certificate Authorities. Devices operating on ATS endpoints are fully interoperable with devices operating on Symantec endpoints in the same account and do not require any re-registration.

> **Note**
> To see your `iot:Data-ATS` endpoint in the AWS IoT Core console, choose **Settings**. The console displays only the `iot:Data-ATS` endpoint. By default, the `describe-endpoint` command displays the `iot:Data` endpoint for backward compatibility. To see the `iot:Data-ATS` endpoint, specify the `--endpointType` parameter, as in the previous example.

Creating an `IotDataPlaneClient` with the AWS SDK for Java

By default, the AWS SDK for Java - Version 2 creates an `IotDataPlaneClient` by using an `iot:Data` endpoint. To create a client that uses an `iot:Data-ATS` endpoint, you must do the following.

- Create an `iot:Data-ATS` endpoint by using the DescribeEndpoint API.
- Specify that endpoint when you create the `IotDataPlaneClient`.

The following example performs both of these operations.

```
public void setup() throws Exception {
        IotClient client =
 IotClient.builder().credentialsProvider(CREDENTIALS_PROVIDER_CHAIN).region(Region.US_EAST_1).build();
        String endpoint = client.describeEndpoint(r -> r.endpointType("iot:Data-
ATS")).endpointAddress();
        iot = IotDataPlaneClient.builder()
                                .credentialsProvider(CREDENTIALS_PROVIDER_CHAIN)
                                .endpointOverride(URI.create("https://" + endpoint))
                                .region(Region.US_EAST_1)
                                .build();
}
```

CA certificates for server authentication

Depending on which type of data endpoint you are using and which cipher suite you have negotiated, AWS IoT Core server authentication certificates are signed by one of the following root CA certificates:

VeriSign Endpoints (legacy)

- RSA 2048 bit key: VeriSign Class 3 Public Primary G5 root CA certificate

Amazon Trust Services Endpoints (preferred)

> **Note**
> You might need to right click these links and select **Save link as...** to save these certificates as files.

- RSA 2048 bit key: Amazon Root CA 1.
- RSA 4096 bit key: Amazon Root CA 2. Reserved for future use.
- ECC 256 bit key: Amazon Root CA 3.
- ECC 384 bit key: Amazon Root CA 4. Reserved for future use.

These certificates are all cross-signed by the Starfield Root CA Certificate. All new AWS IoT Core regions, beginning with the May 9, 2018 launch of AWS IoT Core in the Asia Pacific (Mumbai) Region, serve only ATS certificates.

Server authentication guidelines

There are many variables that can affect a device's ability to validate the AWS IoT Core server authentication certificate. For example, devices may be too memory constrained to hold all possible root CA certificates, or devices may implement a non-standard method of certificate validation. For these reasons we suggest following these guidelines:

- We recommend that you use your ATS endpoint and install all supported Amazon Root CA certificates.
- If you cannot store all of these certificates on your device and if your devices do not use ECC-based validation, you can omit the Amazon Root CA 3 and Amazon Root CA 4 ECC certificates. If your devices

do not implement RSA-based certificate validation, you can omit the Amazon Root CA 1 and Amazon Root CA 2 RSA certificates. You might need to right click these links and select **Save link as...** to save these certificates as files.

- If you are experiencing server certificate validation issues when connecting to your ATS endpoint, try adding the relevant cross-signed Amazon Root CA certificate to your trust store. You might need to right click these links and select **Save link as...** to save these certificates as files.
  - Cross-signed Amazon Root CA 1
  - Cross-signed Amazon Root CA 2 - Reserved for future use.
  - Cross-signed Amazon Root CA 3
  - Cross-signed Amazon Root CA 4 - Reserved for future use.
- If you are experiencing server certificate validation issues, your device may need to explicitly trust the root CA. Try adding the Starfield Root CA Certificate to your trust store.
- If you still experience issues after executing the steps above, please contact AWS Developer Support.

Note
CA certificates have an expiration date after which they cannot be used to validate a server's certificate. CA certificates might have to be replaced before their expiration date. Make sure that you can update the root CA certificates on all of your devices or clients to help ensure ongoing connectivity and to keep up to date with security best practices.

Note
When connecting to AWS IoT Core in your device code, pass the certificate into the API you are using to connect. The API you use will vary by SDK. For more information, see the AWS IoT Core Device SDKs (p. 990).

Client authentication

AWS IoT supports three types of identity principals for device or client authentication:

- X.509 client certificates (p. 203)
- IAM users, groups, and roles (p. 221)
- Amazon Cognito identities (p. 222)

These identities can be used with devices, mobile, web, or desktop applications. They can even be used by a user typing AWS IoT command line interface (CLI) commands. Typically, AWS IoT devices use X.509 certificates, while mobile applications use Amazon Cognito identities. Web and desktop applications use IAM or federated identities. AWS CLI commands use IAM. For more information about IAM identities, see Identity and access management for AWS IoT (p. 277).

X.509 client certificates

X.509 certificates provide AWS IoT with the ability to authenticate client and device connections. Client certificates must be registered with AWS IoT before a client can communicate with AWS IoT. A client certificate can be registered in multiple AWS accounts in the same AWS Region to facilitate moving devices between your AWS accounts in the same region. See Using X.509 client certificates in multiple AWS accounts with multi-account registration (p. 204) for more information.

We recommend that each device or client be given a unique certificate to enable fine-grained client management actions, including certificate revocation. Devices and clients must also support rotation and replacement of certificates to help ensure smooth operation as certificates expire.

For information about using X.509 certificates to support more than a few devices, see Device provisioning (p. 658) to review the different certificate management and provisioning options that AWS IoT supports.

AWS IoT supports these types of X.509 client certificates:

- X.509 certificates generated by AWS IoT
- X.509 certificates signed by a CA registered with AWS IoT.
- X.509 certificates signed by a CA that is not registered with AWS IoT.

This section describes how to manage X.509 certificates in AWS IoT. You can use the AWS IoT console or AWS CLI to perform these certificate operations:

- Create AWS IoT client certificates (p. 205)
- Create your own client certificates (p. 206)
- Register a client certificate (p. 211)
- Activate or deactivate a client certificate (p. 215)
- Revoke a client certificate (p. 217)

For more information about the AWS CLI commands that perform these operations, see AWS IoT CLI Reference.

Using X.509 client certificates

X.509 certificates authenticate client and device connections to AWS IoT. X.509 certificates provide several benefits over other identification and authentication mechanisms. X.509 certificates enable asymmetric keys to be used with devices. For example, you could burn private keys into secure storage on a device so that sensitive cryptographic material never leaves the device. X.509 certificates provide stronger client authentication over other schemes, such as user name and password or bearer tokens, because the private key never leaves the device.

AWS IoT authenticates client certificates using the TLS protocol's client authentication mode. TLS support is available in many programming languages and operating systems and is commonly used for encrypting data. In TLS client authentication, AWS IoT requests an X.509 client certificate and validates the certificate's status and AWS account against a registry of certificates. It then challenges the client for proof of ownership of the private key that corresponds to the public key contained in the certificate. AWS IoT requires clients to send the Server Name Indication (SNI) extension to the Transport Layer Security (TLS) protocol. For more information on configuring the SNI extension, see Transport security in AWS IoT (p. 275).

X.509 certificates can be verified against a trusted certificate authority (CA). You can create client certificates that use the Amazon Root CA and you can use your own client certificates signed by another CA. For more information about using your own X.509 certificates, see Create your own client certificates (p. 206).

The date and time when certificates signed by a CA certificate expire are set when the certificate is created. X.509 certificates generated by AWS IoT expire at midnight UTC on December 31, 2049 (2049-12-31T23:59:59Z). For more information about using the AWS IoT console to create certificates that use the Amazon Root CA, see Create AWS IoT client certificates (p. 205).

Using X.509 client certificates in multiple AWS accounts with multi-account registration

Multi-Account Registration makes it possible to move devices between your AWS accounts in the same Region or in different Regions. With this, you can register, test, and configure a device in a pre-production account, and then register and use the same device and device certificate in a production account. You can also register the client certificate on the device (the device certificates) without a CA (p. 213) that is registered with AWS IoT.

Note
Certificates used for Multi-Account Registration are supported on the `iot:Data-ATS`, `iot:Data` (legacy), and `iot:Jobs` endpoint types. Certificates used for Multi-Account Registration cannot be used on the `iot:CredentialProvider` endpoint type. For more information about AWS IoT device endpoints, see AWS IoT device data and service endpoints (p. 73).

Devices that use Multi-Account Registration must send the Server Name Indication (SNI) extension to the Transport Layer Security (TLS) protocol and provide the complete endpoint address in the `host_name` field, when they connect to AWS IoT. AWS IoT uses the endpoint address in `host_name` to route the connection to the correct AWS IoT account. Existing devices that don't send a valid endpoint address in `host_name` will continue to work, but they will not be able to use the features that require this information. For more information about the SNI extension and to learn how to identify the endpoint address for the `host_name` field, see Transport security in AWS IoT (p. 275).

To use multi-account registration

1. Do not register the CA that signed the device certificates with AWS IoT.
2. Register the device certificates without a CA. See Register a client certificate without a registered CA (CLI) (p. 213).
3. Use the correct `host_name` in the SNI extension to TLS when the device connects to AWS IoT. See Transport security in AWS IoT (p. 275).

Certificate signing algorithms supported by AWS IoT

AWS IoT supports the following certificate-signing algorithms:

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

Create AWS IoT client certificates

AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA).

This topic describes how to create a client certificate signed by the Amazon Root certificate authority and download the certificate files. After you create the client certificate files, you must install them on the client.

Note
Each X.509 client certificate provided by AWS IoT holds issuer and subject attributes that are set at the time of certificate creation. The certificate attributes are immutable only after the certificate is created.

You can use the AWS IoT console or the AWS CLI to create an AWS IoT certificate signed by the Amazon Root certificate authority.

Create an AWS IoT certificate (console)

To create an AWS IoT certificate using the AWS IoT console

1. Sign in to the AWS Management Console, and open the AWS IoT console.

2. In the left navigation pane, choose **Secure**, choose **Certificates**, and then choose **Create**.
3. Choose **One-click certificate creation (recommended)** - **Create certificate**.
4. From the **Certificate created!** page, download the client certificate files for the thing, public key, and private key to a secure location.

   If you also need the Amazon Root CA certificate file, this page also has the link to the page from where you can download it.
5. A client certificate has now been created and registered with AWS IoT. You must activate the certificate before you use it in a client.

   Choose **Activate** to activate the client certificate now. If you don't want to activate the certificate now, Activate a client certificate (console) (p. 215) describes how to activate the certificate later.
6. If you want to attach a policy to the certificate, choose **Attach a policy**.

   If you don't want to attach a policy now, choose **Done** to finish. You can attach a policy later.

After you complete the procedure, install the certificate files on the client.

Create an AWS IoT certificate (CLI)

The AWS CLI provides the **create-keys-and-certificate** command to create client certificates signed by the Amazon Root certificate authority. This command, however, does not download the Amazon Root CA certificate file. You can download the Amazon Root CA certificate file from CA certificates for server authentication (p. 202).

This command creates private key, public key, and X.509 certificate files and registers and activates the certificate with AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile certificate_filename \
    --public-key-outfile public_key_filename \
    --private-key-outfile private_key_filename
```

If you don't want to activate the certificate when you create and register it, this command creates private key, public key, and X.509 certificate files and registers the certificate, but it does not activate it. Activate a client certificate (CLI) (p. 216) describes how to activate the certificate later.

```
aws iot create-keys-and-certificate \
    --no-set-as-active \
    --certificate-pem-outfile certificate_filename \
    --public-key-outfile public_key_filename \
    --private-key-outfile private_key_filename
```

Install the certificate files on the client.

Create your own client certificates

AWS IoT supports client certificates signed by other root certificate authorities (CA). You can register client certificates signed by another root CA; however, if you want the device or client to register its client certificate when it first connects to AWS IoT, the root CA must be registered with AWS IoT.

Note
A CA certificate can be registered by only one account in a Region.

For more information about using X.509 certificates to support more than a few devices, see Device provisioning (p. 658) to review the different certificate management and provisioning options that AWS IoT supports.

Topics

- Manage your CA certificates (p. 207)
- Create a client certificate using your CA certificate (p. 210)

Manage your CA certificates

This section describes common tasks for managing your own certificate authority (CA) certificates.

- Create a CA certificate (p. 207), if you need one.
- Register your CA certificate (p. 207)
- Deactivate a CA certificate (p. 209)

Create a CA certificate

If you do not have a CA certificate, you can use OpenSSL v1.1.1i tools to create one.

Note
You can't perform this procedure in the AWS IoT console.

To create a CA certificate using OpenSSL v1.1.1i tools

1. Generate a key pair.

```
openssl genrsa -out root_CA_key_filename 2048
```

2. Use the private key from the key pair to generate a CA certificate.

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename \
    -sha256 -days 1024 \
    -out root_CA_pem_filename
```

Register your CA certificate

You might need to register your certificate authority (CA) with AWS IoT if you are using client certificates signed by a CA that AWS IoT doesn't recognize.

If you want clients to automatically register their client certificates with AWS IoT when they first connect, the CA that signed the client certificates must be registered with AWS IoT. Otherwise, you don't need to register the CA certificate that signed the client certificates.

Note
A CA certificate can be registered by only one account in a Region.

Register a CA certificate (console)

Note
Make sure you have the root CA's certificate file and private key file before you begin.

This procedure also requires using the command line interface to run **OpenSSL v1.1.1i** commands.

To register a CA certificate using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **CAs**, and then **Register**.
3. In **Select a CA**, choose **Register CA**.
4. In **Register a CA certificate**, follow the steps displayed.

   Steps 1 - 4 are performed in the command line interface.

   Steps 5 and 6 require the files created in Steps 3 and 4.
5. If you want to activate this certificate when you register it, check **Activate CA certificate**.

   The CA certificate must be active before you can register any client certificates that are signed by it.
6. If you want to enable this certificate to automatically register client certificates signed by this certificate, select **Enable auto-registration of device certificates**.
7. Choose **Register CA certificate** to complete the registration.

The CA certificate appears in the list of certificate authorities with its current status.

Register a CA certificate (CLI)

Note
Make sure you have the root CA's certificate file and private key file before you begin.

To register a CA certificate using the AWS CLI

1. Use **get-registration-code** to get a registration code from AWS IoT. Save the `registrationCode` returned to use as the `Common Name` of the private key verification certificate.

```
aws iot get-registration-code
```

2. Generate a key pair for the private key verification certificate:

```
openssl genrsa -out verification_cert_key_filename 2048
```

3. Create a certificate signing request (CSR) for the private key verification certificate. Set the `Common Name` field of the certificate to the `registrationCode` returned by **get-registration-code**.

```
openssl req -new \
    -key verification_cert_key_filename \
    -out verification_cert_csr_filename
```

   You are prompted for some information, including the `Common Name` for the certificate.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
```

```
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

4. Use the CSR to create a private key verification certificate:

```
openssl x509 -req \
                                        -in verification_cert_csr_filename \
    -CA root_CA_pem_filename \
    -CAkey root_CA_key_filename \
    -CAcreateserial \
    -out verification_cert_pem_filename \
    -days 500 -sha256
```

5. Register the CA certificate with AWS IoT. Pass in the CA certificate filename and the private key verification certificate filename to the **register-ca-certificate** command:

```
aws iot register-ca-certificate \
    --ca-certificate file://root_CA_pem_filename \
    --verification-cert file://verification_cert_pem_filename
```

   This command returns the `certificateId`, if successful.

6. At this point, the CA certificate has been registered with AWS IoT, but is not active. The CA certificate must be active before you can register any client certificates that are signed by it.

   This step activates the CA certificate.

   Use the **update-certificate** CLI command to activate the CA certificate:

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

Use the **describe-ca-certificate** command to see the status of the CA certificate.

Deactivate a CA certificate

When a certificate authority (CA) certificate is enabled for automatic client certificate registration, AWS IoT checks the CA certificate used to sign the client certificate to make sure the CA is `ACTIVE`. If the CA certificate is `INACTIVE`, AWS IoT doesn't allow the client certificate to be registered.

By setting the CA certificate as `INACTIVE`, you prevent any new client certificates issued by the CA from being registered automatically.

Note
Any registered client certificates that were signed by the compromised CA certificate continue to work until you explicitly revoke each one of them.

Deactivate a CA certificate (console)

To deactivate a CA certificate using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.

2. In the left navigation pane, choose **Secure**, choose **CAs**.
3. In the list of certificate authorities, find the one that you want to deactivate, and open the option menu by using the ellipsis icon.
4. On the option menu, choose **Deactivate**.

The certificate authority should show as **Inactive** in the list.

> **Note**
> The AWS IoT console does not provide a way to list the certificates that were signed by the CA you deactivated. For an AWS CLI option to list those certificates, see Deactivate a CA certificate (CLI) (p. 210).

Deactivate a CA certificate (CLI)

The AWS CLI provides the **update-ca-certificate** command to deactivate a CA certificate.

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

Use the **list-certificates-by-ca** command to get a list of all registered client certificates that were signed by the specified CA. For each client certificate signed by the specified CA certificate, use the **update-certificate** command to revoke the client certificate to prevent it from being used.

Use the **describe-ca-certificate** command to see the status of the CA certificate.

Create a client certificate using your CA certificate

You can use your own certificate authority (CA) to create client certificates. The client certificate must be registered with AWS IoT before use. For information about the registration options for your client certificates, see Register a client certificate (p. 211).

Create a client certificate (CLI)

> **Note**
> You can't perform this procedure in the AWS IoT console.

To create a client certificate using the AWS CLI

1. Generate a key pair.

```
openssl genrsa -out device_cert_key_filename 2048
```

2. Create a CSR for the client certificate.

```
openssl req -new \
    -key device_cert_key_filename \
    -out device_cert_csr_filename
```

You are prompted for some information, as shown here:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

3. Create a client certificate from the CSR.

```
openssl x509 -req \
    -in device_cert_csr_filename \
    -CA root_CA_pem_filename \
    -CAkey root_CA_key_filename \
    -CAcreateserial \
    -out device_cert_pem_filename \
    -days 500 -sha256
```

At this point, the client certificate has been created, but it has not yet been registered with AWS IoT. For information about how and when to register the client certificate, see Register a client certificate (p. 211).

Register a client certificate

Client certificates must be registered with AWS IoT to enable communications between the client and AWS IoT. You can register each client certificate manually, or you can configure the client certificates to register automatically when the client connects to AWS IoT for the first time.

If you want your clients and devices to register their client certificates when they first connect, you must Register your CA certificate (p. 207) used to sign the client certificate with AWS IoT in the Regions in which you want to use it. The Amazon Root CA is automatically registered with AWS IoT.

Client certificates can be shared by AWS accounts and Regions. The procedures in these topics must be performed in each account and Region in which you want to use the client certificate. The registration of a client certificate in one account or Region is not automatically recognized by another.

Note
Clients that use the Transport Layer Security (TLS) protocol to connect to AWS IoT must support the Server Name Indication (SNI) extension to TLS. For more information, see Transport security in AWS IoT (p. 275).

Topics

Register a client certificate manually

You can register a client certificate manually by using the AWS IoT console and AWS CLI.

The procedures in this topic must be performed in each account and Region in which you want to use the client certificate. Client certificates can be shared by AWS accounts and Regions, but only if the client certificate is signed by a certificate authority (CA) that is NOT registered with AWS IoT. The registration of a client certificate in one account or Region is not automatically recognized by another.

Register a client certificate manually (console)

Note
Before you perform this procedure, make sure that you have the client certificate's .pem file and that the client certificate was signed by a CA that you have registered with AWS IoT (p. 207).

To register an existing certificate with AWS IoT using the console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**, and then choose **Create**.
3. On **Create a certificate**, locate the **Use my certificate** entry, and choose **Get started**.
4. On **Select a CA**:

   - **if the client certificates are signed by a CA that is registered with AWS IoT**

     Choose that CA from the list, and then choose **Next**.
   - **If the client certificates are not signed by a CA that is registered with AWS IoT**

     See Register a client certificate without a registered CA manually (console) (p. 212).
   - **If the client certificates are signed by Amazon's CA**

     Don't select any CA, just choose **Next**.

   If the client certificates are not signed by a CA that is registered with AWS IoT, see Register a client certificate without a registered CA manually (console) (p. 212).
5. On **Register existing device certificates**, choose **Select certificates**, and select up to 10 certificate files to register.
6. After closing the file dialog box, select whether you want to activate or revoke the client certificates when you register them.

   If you don't activate a certificate when it is registered, Activate a client certificate (console) (p. 215) describes how to activate it later.

   If a certificate is revoked when it is registered, it can't be activated later.

   After you choose the certificate files to register, and select the actions to take after registration, select **Register certificates**.

The client certificates that are registered successfully appear in the list of certificates.

Register a client certificate without a registered CA manually (console)

Note
Before you perform this procedure, make sure that you have the client certificate's .pem file.

To register an existing certificate with AWS IoT using the console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**, and then choose **Create**.
3. On **Create a certificate**, locate the **Use my certificate** entry, and choose **Get started**.
4. On **Select a CA**, choose **Next**.
5. On **Register existing device certificates**, choose **Select certificates**, and select up to 10 certificate files to register.

6. After closing the file dialog box, select whether you want to activate or revoke the client certificates when you register them.

   If you don't activate a certificate when it is registered, Activate a client certificate (console) (p. 215) describes how to activate it later.

   If a certificate is revoked when it is registered, it can't be activated later.

   After you choose the certificate files to register, and select the actions to take after registration, select **Register certificates**.

The client certificates that are registered successfully appear in the list of certificates.

Register a client certificate manually (CLI)

Note
Before you perform this procedure, make sure that you have the certificate authority (CA) .pem and the client certificate's .pem file. The client certificate must be signed by a certificate authority (CA) that you have registered with AWS IoT (p. 207).

Use the **register-certificate** command to register, but not activate, a client certificate.

```
aws iot register-certificate \
    --certificate-pem file://device_cert_pem_filename \
    --ca-certificate-pem file://ca_cert_pem_filename
```

The client certificate is registered with AWS IoT, but it is not active yet. See Activate a client certificate (CLI) (p. 216) for information on how to activate it later.

You can also activate the client certificate when you register it by using this command.

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_pem_filename \
    --ca-certificate-pem file://ca_cert_pem_filename
```

For more information about activating the certificate so that it can be used to connect to AWS IoT, see Activate or deactivate a client certificate (p. 215)

Register a client certificate without a registered CA (CLI)

Note
Before you perform this procedure, make sure that you have the certificate's .pem file.

Use the **register-certificate-without-ca** command to register, but not activate, a client certificate.

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_pem_filename
```

The client certificate is registered with AWS IoT, but it is not active yet. See Activate a client certificate (CLI) (p. 216) for information on how to activate it later.

You can also activate the client certificate when you register it by using this command.

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_pem_filename
```

For more information about activating the certificate so that it can be used to connect to AWS IoT, see Activate or deactivate a client certificate (p. 215)

Register a client certificate when the client connects to AWS IoT (just-in-time registration)

You can configure a CA certificate to enable client certificates it has signed to register with AWS IoT automatically the first time the client connects to AWS IoT.

To register client certificates when a client connects to AWS IoT for the first time, you must enable the CA certificate for automatic registration and configure the first connection by the client to provide the required certificates.

Configure a CA certificate to support automatic registration (console)

To configure a CA certificate to support automatic client certificate registration using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **CAs**.
3. In the list of certificate authorities, find the one for which you want to enable automatic registration, and open the option menu by using the ellipsis icon.
4. On the option menu, choose **Enable auto-registration**.

   Note
   The auto-registration status is not shown in the list of certificate authorities. To see the auto-registration status of a certificate authority, you must open the **Details** page of the certificate authority.

Configure a CA certificate to support automatic registration (CLI)

If you have already registered your CA certificate with AWS IoT, use the **update-ca-certificate** command to set `autoRegistrationStatus` of the CA certificate to `ENABLE`.

```
aws iot update-ca-certificate \
--certificate-idcaCertificateId \
--new-auto-registration-status ENABLE
```

If you want to enable `autoRegistrationStatus` when you register the CA certificate, use the **register-ca-certificate** command.

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_pem_filename \
--verification-cert file://verification_cert_pem_filename
```

Use the **describe-ca-certificate** command to see the status of the CA certificate.

Configure the first connection by a client for automatic registration

When a client attempts to connect to AWS IoT for the first time, it must present a file that contains both your registered CA certificate and the client certificate signed by your CA certificate as part of the TLS handshake. You can combine the two files using a command, such as the following:

```
cat device_cert_filename ca_certificate_pem_filename > combined_filename
```

When the client connects to AWS IoT, use the *combined_filename* file as the certificate file. AWS IoT recognizes the CA certificate as a registered CA certificate, registers the client certificate, and sets its status to `PENDING_ACTIVATION`. This means that the client certificate was automatically registered and is awaiting activation. The client certificate's state must be `ACTIVE` before it can be used to connect to AWS IoT.

When AWS IoT automatically registers a certificate or when a client presents a certificate in the `PENDING_ACTIVATION` status, AWS IoT publishes a message to the following MQTT topic:

`$aws/events/certificates/registered/caCertificateId`

Where *caCertificateId* is the ID of the CA certificate that issued the client certificate.

The message published to this topic has the following structure:

```
{
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

You can create a rule that listens on this topic and performs some actions. We recommend that you create a Lambda rule that verifies the client certificate is not on a certificate revocation list (CRL), activates the certificate, and creates and attaches a policy to the certificate. The policy determines which resources the client can access. For more information about how to create a Lambda rule that listens on the `$aws/events/certificates/registered/caCertificateID` topic and performs these actions, see Just-in-Time Registration of Client Certificates on AWS IoT.

If any error or exception occurs during the auto-registration of the client certificates, AWS IoT sends events or messages to your logs in CloudWatch Logs. For more information about setting up the logs for your account, see the Amazon CloudWatch documentation.

Activate or deactivate a client certificate

AWS IoT verifies that a client certificate is active when it authenticates a connection.

You can create and register client certificates without activating them so they can't be used until you want to use them. You can also deactivate active client certificates to disable them temporarily. Finally, you can revoke client certificates to prevent them from any future use.

Activate a client certificate (console)

To activate a client certificate using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.
3. In the list of certificates, locate the certificate that you want to activate, and open the option menu by using the ellipsis icon.
4. In the option menu, choose **Activate**.

The certificate should show as **Active** in the list of certificates.

Deactivate a client certificate (console)

To deactivate a client certificate using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.
3. In the list of certificates, locate the certificate that you want to deactivate, and open the option menu by using the ellipsis icon.
4. In the option menu, choose **Deactivate**.

The certificate should show as **Inactive** in the list of certificates.

Activate a client certificate (CLI)

The AWS CLI provides the **update-certificate** command to activate a certificate.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

If the command was successful, the certificate's status will be `ACTIVE`. Run **describe-certificate** to see the certificate's status.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Deactivate a client certificate (CLI)

The AWS CLI provides the **update-certificate** command to deactivate a certificate.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

If the command was successful, the certificate's status will be `INACTIVE`. Run **describe-certificate** to see the certificate's status.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Attach a thing or policy to a client certificate

When you create and register a certificate separate from an AWS IoT thing, it will not have any policies that authorize any AWS IoT operations, nor will it be associated with any AWS IoT thing object. This section describes how to add these relationships to a registered certificate.

> **Important**
> To complete these procedures, you must have already created the thing or policy that you want to attach to the certificate.

The certificate authenticates a device with AWS IoT so that it can connect. Attaching the certificate to a thing resource establishes the relationship between the device (by way of the certificate) and the thing resource. To authorize the device to perform AWS IoT actions, such as to allow the device to connect and publish messages, an appropriate policy must be attached to the device's certificate.

Attach a thing to a client certificate (console)

You will need the name of the thing object to complete this procedure.

To attach a thing object to a registered certificate

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.
3. In the list of certificates, locate the certificate to which you want to attach a policy, open the certificate's option menu by choosing the ellipsis icon, and choose **Attach thing**.
4. In the pop-up, locate the name of the thing you want to attach to the certificate, choose its check box, and choose **Attach**.

The thing object should now appear in the list of things on the certificate's details page.

Attach a policy to a client certificate (console)

You will need the name of the policy object to complete this procedure.

To attach a policy object to a registered certificate

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.
3. In the list of certificates, locate the certificate to which you want to attach a policy, open the certificate's option menu by choosing the ellipsis icon, and choose **Attach policy**.
4. In the pop-up, locate the name of the policy you want to attach to the certificate, choose its check box, and choose **Attach**.

The policy object should now appear in the list of policies on the certificate's details page.

Attach a thing to a client certificate (CLI)

The AWS CLI provides the **attach-thing-principal** command to attach a thing object to a certificate.

```
aws iot attach-thing-principal \
    --principal certificateArn \
    --thing-name thingName
```

Attach a policy to a client certificate (CLI)

The AWS CLI provides the **attach-policy** command to attach a policy object to a certificate.

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

Revoke a client certificate

If you detect suspicious activity on a registered client certificate, you can revoke it so that it can't be used again.

Revoke a client certificate (console)

To revoke a client certificate using the AWS IoT console

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.
3. In the list of certificates, locate the certificate that you want to revoke, and open the option menu by using the ellipsis icon.
4. In the option menu, choose **Revoke**.

If the certificate was successfully revoked, it will show as **Revoked** in the list of certificates.

Revoke a client certificate (CLI)

The AWS CLI provides the **update-certificate** command to revoke a certificate.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

If the command was successful, the certificate's status will be `REVOKED`. Run **describe-certificate** to see the certificate's status.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Transfer a certificate to another account

X.509 certificates that belong to one AWS account can be transferred to another AWS account.

To transfer an X.509 certificate from one AWS account to another

1. the section called "Begin a certificate transfer" (p. 218)

   The certificate must be deactivated and detached from all policies and things before initiating the transfer.
2. the section called "Accept or reject a certificate transfer" (p. 220)

   The receiving account must explicitly accept or reject the transferred certificate. After the receiving account accepts the certificate, the certificate must be activated before use.
3. the section called "Cancel a certificate transfer" (p. 221)

   The originating account can cancel a transfer, if the certificate has not been accepted.

Begin a certificate transfer

You can begin to transfer a certificate to another AWS account by using the AWS IoT console or the AWS CLI.

Begin a certificate transfer (console)

To complete this procedure, you'll need the ID of the certificate that you want to transfer.

Do this procedure from the account with the certificate to transfer.

To begin to transfer a certificate to another AWS account

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.

   Choose the certificate with an **Active** or **Inactive** status that you want to transfer and open its details page.
3. On the certificate's **Details** page, in the **Actions** menu, if the **Deactivate** option is available, choose the **Deactivate** option to deactivate the certificate.
4. On the certificate's **Details** page, in the left menu, choose **Policies**.
5. On the certificate's **Policies** page, if there are any policies attached to the certificate, detach each one by opening the policy's options menu and choosing **Detach**.

   The certificate must not have any attached policies before you continue.
6. On the certificate's **Policies** page, in the left menu, choose **Things**.
7. On the certificate's **Things** page, if there are any things attached to the certificate, detach each one by opening the thing's options menu and choosing **Detach**.

   The certificate must not have any attached things before you continue.
8. On the certificate's **Things** page, in the left menu, choose **Details**.
9. On the certificate's **Details** page, in the **Actions** menu, choose **Start transfer** to open the **Start transfer** dialog box.
10. In the **Start transfer** dialog box, enter the AWS account number of the account to receive the certificate and an optional short message.
11. Choose **Start transfer** to transfer the certificate.

The console should display a message that indicates the success or failure of the transfer. If the transfer was started, the certificate's status is updated to **Transferred**.

Begin a certificate transfer (CLI)

To complete this procedure, you'll need the *`certificateId`* and the *`certificateArn`* of the certificate that you want to transfer.

Do this procedure from the account with the certificate to transfer.

To begin to transfer a certificate to another AWS account

1. Use the **update-certificate** command to deactivate the certificate.

   ```
   aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
   ```
2. Detach all policies.

   1. Use the **list-attached-policies** command to list the policies attached to the certificate.

      ```
      aws iot list-attached-policies --target certificateArn
      ```
   2. For each attached policy, use the **detach-policy** command to detach the policy.

      ```
      aws iot detach-policy --target certificateArn --policy-name policy-name
      ```

3. Detach all things.

   1. Use the **list-principal-things** command to list the things attached to the certificate.

   ```
   aws iot list-principal-things --principal certificateArn
   ```

   2. For each attached thing, use the **detach-thing-principal** command to detach the thing.

   ```
   aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
   ```

4. Use the **transfer-certificate** command to start the certificate transfer.

   ```
   aws iot transfer-certificate --certificate-id certificateId --target-aws-
   account account-id
   ```

Accept or reject a certificate transfer

You can accept or reject a certificate transferred to you AWS account from another AWS account by using the AWS IoT console or the AWS CLI.

Accept or reject a certificate transfer (console)

To complete this procedure, you'll need the ID of the certificate that was transferred to your account.

Do this procedure from the account receiving the certificate that was transferred.

To accept or reject a certificate that was transferred to your AWS account

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.

   Choose the certificate with a status of **Pending transfer** that you want to accept or reject and open its details page.
3. On the certificate's **Details** page, in the **Actions** menu,

   - To accept the certificate, choose **Accept transfer**.
   - To not accept the certificate, choose **Reject transfer**.

Accept or reject a certificate transfer (CLI)

To complete this procedure, you'll need the *certificateId* of the certificate transfer that you want to accept or reject.

Do this procedure from the account receiving the certificate that was transferred.

To accept or reject a certificate that was transferred to your AWS account

1. Use the **accept-certificate-transfer** command to accept the certificate.

   ```
   aws iot accept-certificate-transfer --certificate-id certificateId
   ```

2. Use the **reject-certificate-transfer** command to reject the certificate.

   ```
   aws iot reject-certificate-transfer --certificate-id certificateId
   ```

Cancel a certificate transfer

You can cancel a certificate transfer before it has been accepted by using the AWS IoT console or the AWS CLI.

Cancel a certificate transfer (console)

To complete this procedure, you'll need the ID of the certificate transfer that you want to cancel.

Do this procedure from the account that initiated the certificate transfer.

To cancel a certificate transfer

1. Sign in to the AWS Management Console and open the AWS IoT console.
2. In the left navigation pane, choose **Secure**, choose **Certificates**.

   Choose the certificate with **Transferred** status whose transfer you want to cancel and open its options menu.
3. On the certificate's options menu, choose the **Revoke transfer** option to cancel the certificate transfer.

   Important
   Be careful not to mistake the **Revoke transfer** option with the **Revoke** option.
   The **Revoke transfer** option cancels the certificate transfer, while the **Revoke** option makes the certificate irreversibly unusable by AWS IoT.

Cancel a certificate transfer (CLI)

To complete this procedure, you'll need the *certificateId* of the certificate transfer that you want to cancel.

Do this procedure from the account that initiated the certificate transfer.

Use the **cancel-certificate-transfer** command to cancel the certificate transfer.

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

IAM users, groups, and roles

IAM users, groups, and roles are the standard mechanisms for managing identity and authentication in AWS. You can use them to connect to AWS IoT HTTP interfaces using the AWS SDK and AWS CLI.

IAM roles also allow AWS IoT to access other AWS resources in your account on your behalf. For example, if you want to have a device publish its state to a DynamoDB table, IAM roles allow AWS IoT to interact with Amazon DynamoDB. For more information, see IAM Roles.

For message broker connections over HTTP, AWS IoT authenticates IAM users, groups, and roles using the Signature Version 4 signing process. For information, see Signing AWS API Requests.

When using AWS Signature Version 4 with AWS IoT, clients must support the following in their TLS implementation:

- TLS 1.2, TLS 1.1, TLS 1.0
- SHA-256 RSA certificate signature validation
- One of the cipher suites from the TLS cipher suite support section

For information, see Identity and access management for AWS IoT (p. 277).

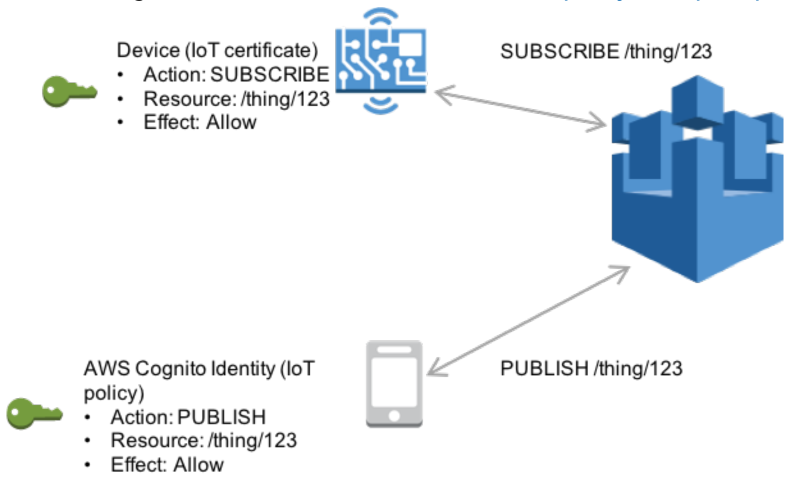
Amazon Cognito identities

Amazon Cognito Identity enables you to create temporary, limited-priviledge AWS credentials for use in mobile and web applications. When you use Amazon Cognito Identity, you create identity pools that create unique identities for your users and authenticate them with identity providers like Login with Amazon, Facebook, and Google. You can also use Amazon Cognito identities with your own developer authenticated identities. For more information, see Amazon Cognito Identity.

To use Amazon Cognito Identity, you define a Amazon Cognito identity pool that is associated with an IAM role. The IAM role is associated with an IAM policy that grants identities from your identity pool permission to access AWS resources like calling AWS services.

Amazon Cognito Identity creates unauthenticated and authenticated identities. Unauthenticated identities are used for guest users in a mobile or web application who want to use the app without signing in. Unauthenticated users are granted only those permissions specified in the IAM policy associated with the identity pool.

When you use authenticated identities, in addition to the IAM policy attached to the identity pool, you can attach an AWS IoT policy to an Amazon Cognito Identity using the AttachPolicy API and give fine-grained permissions to an individual user of your AWS IoT application. In this way, you can assign permissions for specific customers and their devices. For more information about creating policies for Amazon Cognito identities, see Publish/Subscribe policy examples (p. 252).



Custom authentication

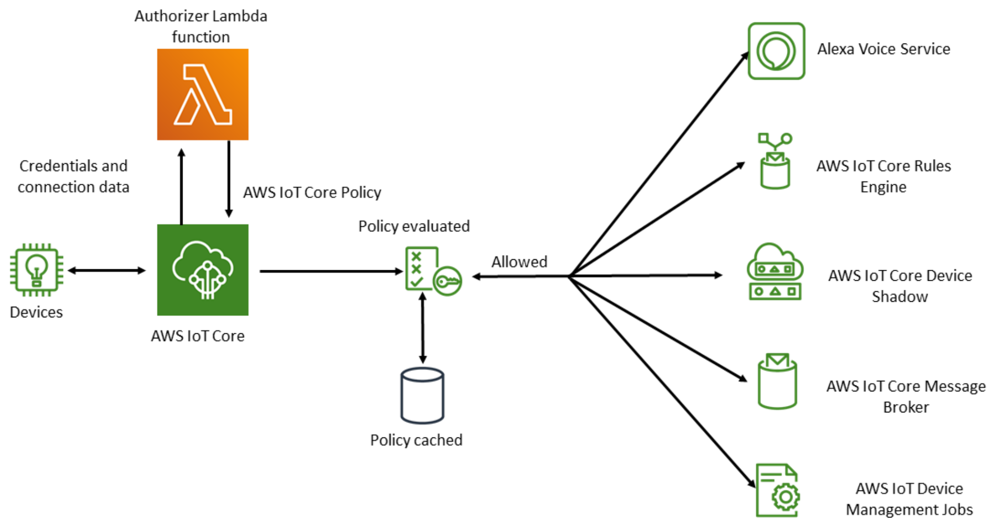
AWS IoT Core lets you define custom authorizers so that you can manage your own client authentication and authorization. This is useful when you need to use authentication mechanisms other than the ones that AWS IoT Core natively supports. (For more information about the natively supported mechanisms, see the section called “Client authentication” (p. 203)).

For example, if you are migrating existing devices in the field to AWS IoT Core and these devices use a custom bearer token or MQTT user name and password to authenticate, you can migrate them to AWS IoT Core without having to provision new identities for them. You can use custom authentication with any of the communication protocols that AWS IoT Core supports. For more information about the protocols that AWS IoT Core supports, see the section called “Device communication protocols” (p. 76).

Topics

Understanding the custom authentication workflow

Custom authentication enables you to define how to authenticate and authorize clients by using authorizer resources. Each authorizer contains of a reference to a customer-managed Lambda function, an optional public key for validating device credentials, and additional configuration information. The following diagram illustrates the authorization workflow for custom authentication in AWS IoT Core.


Authorizer Lambda function
Credentials and connection data
AWS IoT Core Policy
Policy evaluated
Allowed
Devices
AWS IoT Core
Policy cached
Alexa Voice Service
AWS IoT Core Rules Engine
AWS IoT Core Device Shadow
AWS IoT Core Message Broker
AWS IoT Device Management Jobs

AWS IoT Core custom authentication and authorization workflow

The following list explains each step in the custom authentication and authorization workflow.

1. A device connects to a customer's AWS IoT Core data endpoint by using one of the supported the section called "Device communication protocols" (p. 76). The device passes credentials in either the request's header fields or query parameters (for the HTTP Publish or MQTT over WebSockets protocols) or in the user name and password field of the MQTT CONNECT message (for the MQTT and MQTT over WebSockets protocols).
2. AWS IoT Core checks for one of two conditions:
   - The incoming request specifies an authorizer.
   - The AWS IoT Core data endpoint receiving the request has a default authorizer configured for it.

   If AWS IoT Core finds an authorizer in either of these ways, AWS IoT Core triggers the Lambda function associated with the authorizer.
3. (Optional) If you've enabled token signing, AWS IoT Core validates the request signature by using the public key stored in the authorizer before triggering the Lambda function. If validation fails, AWS IoT Core stops the request without invoking the Lambda function.
4. The Lambda function receives the credentials and connection metadata in the request and makes an authentication decision.
5. The Lambda function returns the results of the authentication decision and an AWS IoT Core policy document that specifies what actions are allowed in the connection. The Lambda function also returns

information that specifies how often AWS IoT Core revalidates the credentials in the request by invoking the Lambda function.

6. AWS IoT Core evaluates activity on the connection against the policy it has received from the Lambda function.

Scaling considerations

Because a Lambda function handles authentication and authorization for your authorizer, the function is subject to Lambda pricing and service limits, such as concurrent execution rate. For more information about Lambda pricing, see Lambda Pricing. You can manage the load on your Lambda function by adjusting the `refreshAfterInSecs` and `disconnectAfterInSeconds` parameters in your Lambda function response. For more information about the contents of your Lambda function response, see the section called "Defining your Lambda function" (p. 224).

> **Note**
> If you leave signing enabled, you can prevent excessive triggering of your Lambda by unrecognized clients. Consider this before you disable signing in your authorizer.

Creating and managing custom authorizers

AWS IoT Core implements custom authentication and authorization schemes by using authorizer resources. Each authorizer consists of the following components:

- *Name*: A unique user-defined string that identifies the authorizer.
- *Lambda function ARN*: The Amazon Resource Name (ARN) of the Lambda function that implements the authorization and authentication logic.
- *Token key name*: The key name used to extract the token from the HTTP headers, query parameters, or MQTT CONNECT user name in order to perform signature validation. This value is required if signing is enabled in your authorizer.
- *Signing disabled flag (optional)*: A Boolean value that specifies whether to disable the signing requirement on credentials. This is useful for scenarios where signing the credentials doesn't make sense, such as authentication schemes that use MQTT user name and password. The default value is `false`, so signing is enabled by default.
- *Token signing public key*: The public key that AWS IoT Core uses to validate the token signature. Its minimum length is 2,048 bits. This value is required if signing is enabled in your authorizer.

Lambda charges you for the number of times your Lambda function runs and for the amount of time it takes for the code in your function to execute. For more information about Lambda pricing, see Lambda Pricing. For more information about creating Lambda functions, see the Lambda Developer Guide.

> **Note**
> If you leave signing enabled, you can prevent excessive triggering of your Lambda by unrecognized clients. Consider this before you disable signing in your authorizer.

Defining your Lambda function

When AWS IoT Core invokes your authorizer, it triggers the associated Lambda associated with the authorizer with an event that contains the following JSON object. The example JSON object contains all of the possible fields. Any fields that aren't relevant to the connection request aren't included.

```
{
    "token" :"aToken",
    "signatureVerified": Boolean, // Indicates whether the device gateway has validated the
 signature.
    "protocols": ["tls", "http", "mqtt"], // Indicates which protocols to expect for the
 request.
```

```
    "protocolData": {
        "tls" : {
            "serverName": "serverName" // The server name indication (SNI) host_name
 string.
        },
        "http": {
            "headers": {
                "#{name}": "#{value}"
            },
            "queryString": "?#{name}=#{value}"
        },
        "mqtt": {
            "username": "myUserName",
            "password": "myPassword", // A base64-encoded string.
            "clientId": "myClientId" // Included in the event only when the device sends
 the value.
        }
    },
    "connectionMetadata": {
        "id": UUID // The connection ID. You can use this for logging.
    },
}
```

The Lambda function should use this information to authenticate the incoming connection and decide what actions are permitted in the connection. The function should send a response that contains the following values.

- `IsAuthenticated`: A Boolean value that indicates whether the request is authenticated.
- `PrincipalId`: An alphanumeric string that acts as an identifier for the token sent by the custom authorization request. Its minimum length is 1 character. Its maximum length is 128 characters.
- `PolicyDocuments`: A list of JSON-formatted AWS IoT Core policy documents For more information about creating AWS IoT Core policies, see the section called "AWS IoT Core policies" (p. 235). The maximum number of policy documents is 10 policy documents. Each policy document can contain a maximum of 2,048 characters.
- `DisconnectAfterInSeconds`: An integer that specifies the maximum duration (in seconds) of the connection to the AWS IoT Core gateway. The minimum value is 300 seconds, and the maximum value is 86,400 seconds.
- `RefreshAfterInSeconds`: An integer that specifies the interval between policy refreshes. When this interval passes, AWS IoT Core invokes the Lambda function to allow for policy refreshes. The minimum value is 300 seconds, and the maximum value is 86,400 seconds.

The following JSON object contains an example of a response that your Lambda function can send.

```

{
"isAuthenticated":true, //A Boolean that determines whether client can connect.
"principalId": "xxxxxxxx",  //A string that identifies the connection in logs.
"disconnectAfterInSeconds": 86400,
"refreshAfterInSeconds": 300,
 "policyDocuments": [
      {
        "Version": "2012-10-17",
        "Statement": [
           {
              "Action": "iot:Publish",
              "Effect": "Allow",
              "Resource": "arn:aws:iot:us-east-1:<your_aws_account_id>:topic/
customauthtesting"
            }
         ]
        }
    ]
}
```

The `policyDocument` value must contain a valid AWS IoT Core policy document. For more information about AWS IoT Core policies, see the section called “AWS IoT Core policies” (p. 235). In MQTT over TLS and MQTT over WebSockets connections, AWS IoT Core caches this policy for the interval specified in the value of the `refreshAfterInSeconds` field. During this interval, AWS IoT Core authorizes actions in an established connection against this cached policy without triggering your Lambda function again. If failures occur during custom authentication, AWS IoT Core terminates the connection. AWS IoT Core also terminates the connection if it has been open for longer than the value specified in the `disconnectAfterInSeconds`parameter.

The following JavaScript contains a sample Node.js Lambda function that looks for a password in the MQTT Connect message with a value of `test` and returns a policy that grants permission to connect to AWS IoT Core with a client named `myClientName` and publish to a topic that contains the same client name. If it doesn't find the expected password, it returns a policy that denies those two actions.

```
// A simple Lambda function for an authorizer. It demonstrates
// how to parse an MQTT password and generate a response.

exports.handler = function(event, context, callback) {
    var uname = event.protocolData.mqtt.username;
    var pwd = event.protocolData.mqtt.password;
    var buff = new Buffer(pwd, 'base64');
    var passwd = buff.toString('ascii');
    switch (passwd) {
        case 'test':
            callback(null, generateAuthResponse(passwd, 'Allow'));
        default:
            callback(null, generateAuthResponse(passwd, 'Deny'));
    }
};

// Helper function to generate the authorization response.
var generateAuthResponse = function(token, effect) {
    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'TEST123';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var publishStatement = {};
    var connectStatement = {};
    connectStatement.Action = '["iot:Connect"]';
    connectStatement.Effect = effect;
    connectStatement.Resource = '["arn:aws:iot:us-east-1:123456789012:client/
myClientName"]';
    publishStatement.Action = '["iot:Publish"]';
    publishStatement.Effect = effect;
    publishStatement.Resource = '["arn:aws:iot:us-east-1:123456789012:topic/telemetry/
myClientName"]';
    policyDocument.Statement[0] = connectStatement;
    policyDocument.Statement[1] = publishStatement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 600;

    return authResponse;
}
```

This Lambda function returns the following values when it receives the expected value of `test` in the MQTT Connect password field.

```
{
  "password": "password",
  "isAuthenticated": true,
  "principalId": "principalId",
  "policyDocuments": [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "iot:Connect",
          "Effect": "Allow",
          "Resource": "*"
        },
        {
          "Action": "iot:Publish",
          "Effect": "Allow",
          "Resource": "arn:aws:region:accountId:topic/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Subscribe",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topicfilter/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Receive",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        }
      ]
    }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 600
}
```

Creating an authorizer

You can create an authorizer by using the CreateAuthorizer API. The following example shows how to do this.

```
 aws iot create-authorizer
--authorizer-name MyAuthorizer
--authorizer-function-arn arn:aws:lambda:us-
west-2:<account_id>:function:MyAuthorizerFunction  //The ARN of the Lambda function.
[--token-key-name MyAuthorizerToken //The key used to extract the token from headers.
[--token-signing-public-keys FirstKey=
 "-----BEGIN PUBLIC KEY-----
  [...insert your public key here...]
  -----END PUBLIC KEY-----"
[--status ACTIVE]
[--tags <value>]
[--signing-disabled | --no-signing-disabled]
```

You can use the `signing-disabled` parameter to opt out of signature validation for each invocation of your authorizer. We strongly recommend that you do not disable signing unless you have to. Signature validation protects you against excessive invocations of your Lambda function from unknown devices.

You can't update the `signing-disabled` status of an authorizer after you create it. To change this behavior, you must create another custom authorizer with a different value for the `signing-disabled` parameter.

Values for the `tokenKeyName` and `tokenSigningPublicKeys` parameters are optional if you have disabled signing. They are required values if signing is enabled.

After you create your Lambda function and the custom authorizer, you must explicitly grant the AWS IoT Core service permission to invoke the function on your behalf. You can do this with the following command.

```
aws lambda add-permission --function-name <lambda_function_name> --principal
iot.amazonaws.com --source-arn <authorizer_arn> --statement-id Id-123 --action
"lambda:InvokeFunction"
```

Testing your authorizers

You can use the TestInvokeAuthorizer API to test the invocation and return values of your authorizer. This API enables you to specify protocol metadata and test the signature validation in your authorizer.

The following tabs show how to use the AWS CLI to test your authorizer.

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER \
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER ^
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER `
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

The value of the `token-signature` parameter is the signed token. To learn how to obtain this value, see the section called “Signing the token” (p. 231).

If your authorizer takes a user name and password, you can pass this information by using the `--mqtt-context` parameter. The following tabs show how to use the `TestInvokeAuthorizer` API to send a JSON object that contains a user name, password, and client name to your custom authorizer.

Unix-like

```

aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  \
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'

```

Windows CMD

```

aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  ^
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  `
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

The password must be base64-encoded. The following example shows how to encode a password in a Unix-like environment.

```
echo -n PASSWORD | base64
```

Managing custom authorizers

You can manage your authorizers by using the following APIs.

- ListAuthorizers: Show all authorizers in your account.
- DescribeAuthorizer: Displays properties of the specified authorizer. These values include creation date, last modified date, and other attributes.
- SetDefaultAuthorizer: Specifies the default authorizer for your AWS IoT Core data endpoints. AWS IoT Core uses this authorizer if a device doesn't pass AWS IoT Core credentials and doesn't specify an authorizer. For more information about using AWS IoT Core credentials, see the section called “Client authentication” (p. 203).
- UpdateAuthorizer: Changes the status, token key name, or public keys for the specified authorizer.
- DeleteAuthorizer: Deletes the specified authorizer.

Note
You can't update an authorizer's signing requirement. This means that you can't disable signing in an existing authorizer that requires it. You also can't require signing in an existing authorizer that doesn't require it.

Connecting to AWS IoT Core by using custom authentication

Devices can connect to AWS IoT Core by using custom authentication with any protocol that AWS IoT Core supports for device messaging. For more information about supported communication protocols, see the section called “Device communication protocols” (p. 76). The connection data that you pass to your authorizer Lambda function depends on the protocol you use. For more information about creating your authorizer Lambda function, see the section called “Defining your Lambda function” (p. 224). The following sections explain how to connect to authenticate by using each supported protocol.

HTTP

Devices sending data to AWS IoT Core by using the HTTP Publish API can pass credentials either through request headers or query parameters in their HTTP POST requests. Devices can specify an authorizer to invoke by using the `x-amz-customauthorizer-name` header or query parameter. If you have token signing enabled in your authorizer, you must pass the `token-key-name` and `x-amz-customauthorizer-signature` in either request headers or query parameters. The following example requests show how you pass these parameters in both request headers and query parameters.

```
//Passing credentials via headers
POST /topics/topic?qos=qos HTTP/1.1
Host: your-endpoint
x-amz-customauthorizer-signature: token-signature
token-key-name: some-token
x-amz-customauthorizer-name: <authorizer-name>

//Passing credentials via query parameters
POST /topics/topic?qos=qos&x-amz-customauthorizer-signature=${sign}&token-name=${token-
value} HTTP/1.1
```

MQTT

Devices connecting to AWS IoT Core by using an MQTT connection can pass credentials through the `username` and `password` fields of MQTT messages. The `username` value can also optionally contain a query string that passes additional values (including a token, signature, and authorizer name) to your authorizer. You can use this query string if you want to use a token-based authentication scheme instead of `username` and `password` values.

> **Note**
> Data in the password field is base64-encoded by AWS IoT Core. Your Lambda function must decode it.

The following example contains a `username` string that contains extra parameters that specify a token and signature.

```
username?x-amz-customauthorizer-name=${name}&x-amz-customauthorizer-signature=
${sign}&token-name=${token-value}
```

In order to invoke an authorizer, devices connecting to AWS IoT Core by using MQTT and custom authentication must connect on port 443. They also must pass the Application Layer Protocol Negotiation (ALPN) TLS extension with a value of `mqtt` and the Server Name Indication (SNI) extension with the host name of their AWS IoT Core data endpoint. For more information about these values, see the section called "Device communication protocols" (p. 76). The V2 *AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client* (p. 990) can configure both of these extensions.

MQTT over WebSockets

Devices connecting to AWS IoT Core by using MQTT over WebSockets can pass credentials in one of the two following ways.

- Through request headers or query parameters in the HTTP UPGRADE request to establish the WebSockets connection.
- Through the `username` and `password` fields in the MQTT CONNECT message.

If you pass credentials through the MQTT connect message, the ALPN and SNI TLS extensions are required. For more information about these extensions, see the section called "MQTT" (p. 230). The following example demonstrates how to pass credentials through the HTTP Upgrade request.

```

GET /mqtt HTTP/1.1
 Host: your-endpoint
Upgrade: WebSocket
Connection: Upgrades
x-amz-customauthorizer-signature: token-signature
token-key-name: some-token
sec-WebSocket-Key: any random base64 value
sec-websocket-protocol: mqtt
sec-WebSocket-Version: websocket version
```

Signing the token

You must sign the token with the private key of the public-private key pair that you used in the `create-authorizer` call. The following examples show how to create the token signature by using a UNIX-like command and JavaScript. They use the SHA-256 hash algorithm to encode the signature.

Command line

```
echo -n TOKEN_VALUE | openssl dgst -sha256 -sign PEM encoded RSA private key | openssl
 base64
```

JavaScript

```

const crypto = require('crypto')

const key = "PEM encoded RSA private key"

const k = crypto.createPrivateKey(key)
let sign = crypto.createSign('SHA256')
sign.write(t)
sign.end()
const s = sign.sign(k, 'base64')

```

Troubleshooting your authorizers

This topic walks through common issues that can cause problems in custom authentication workflows and steps for resolving them. To troubleshoot issues most effectively, enable CloudWatch logs for AWS IoT Core and set the log level to **DEBUG**. You can enable CloudWatch logs in the AWS IoT Core console (https://console.aws.amazon.com/iot/). For more information about enabling and configuring logs for AWS IoT Core, see the section called “Configure AWS IoT logging” (p. 310).

> **Note**
> If you leave the log level at **DEBUG** for long periods of time, CloudWatch might store large amounts of logging data. This can increase your CloudWatch charges. Consider using resource-based logging to increase the verbosity for only devices in a particular thing group. For more information about resource-based logging, see the section called “Configure AWS IoT logging” (p. 310). Also, when you're done troubleshooting, reduce the log level to a less verbose level.

Before you start troubleshooting, review the section called “Understanding the custom authentication workflow” (p. 223) for a high-level view of the custom authentication process. This helps you understand where to look for the source of a problem.

This topic discusses the following two areas for you to investigate.

- Issues related to your authorizer's Lambda function.
- Issues related to your device.

Check for issues in your authorizer’s Lambda function

Perform the following steps to make sure that your devices’ connection attempts are invoking your Lambda function.

1. Verify which Lambda function is associated with your authorizer.

You can do this by calling the DescribeAuthorizer API or by clicking on the desired authorizer in the **Secure** section of the AWS IoT Core console.

2. Check the invocation metrics for the Lambda function. Perform the following steps to do this.

   a. Open the AWS Lambda console (https://console.aws.amazon.com/lambda/) and select the function that is associated with your authorizer.
   b. Choose the **Monitor** tab and view metrics for the time frame that is relevant to your problem.

3. If you see no invocations, verify that AWS IoT Core has permission to invoke your Lambda function. If you see invocations, skip to the next step. Perform the following steps to verify that your Lambda function has the required permissions.

   a. Choose the **Permissions** tab for your function in the AWS Lambda console.
   b. Find the **Resource-based Policy** section at the bottom of the page. If your Lambda function has the required permissions, the policy looks like the following example.

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Id-123",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:Region:AccountID:function:FunctionName",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:iot:Region:AccountID:authorizer/AuthorizerName"
        }
      }
    }
  ]
}
```

   c. This policy grants the `InvokeFunction` permission on your function to the AWS IoT Core principal. If you don't see it, you'll have to add it by using the AddPermission API. The following example shows you how to do this by using the AWS CLI.

```
 aws lambda add-permission --function-name FunctionName --principal
iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
"lambda:InvokeFunction"
```

4. If you see invocations, verify that there are no errors. An error might indicate that the Lambda function isn't properly handling the connection event that AWS IoT Core sends to it.

   For information about handling the event in your Lambda function, see the section called "Defining your Lambda function" (p. 224). You can use the test feature in the AWS Lambda console (https://console.aws.amazon.com/lambda/) to hard-code test values in the function to make sure that the function is handling events correctly.

5. If you see invocations with no errors, but your devices are not able to connect (or publish, subscribe, and receive messages), the issue might be that the policy that your Lambda function returns doesn't give permissions for the actions that your devices are trying to take. Perform the following steps to determine whether anything is wrong with the policy that the function returns.

a. Use an Amazon CloudWatch Logs Insights query to scan logs over a short period of time to check for failures. The following example query sorts events by timestamp and looks for failures.

```
display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
 status = "Failure"
```

b. Update your Lambda function to log the data that it's returning to AWS IoT Core and the event that triggers the function. You can use these logs to inspect the policy that the function creates.

Investigating device issues

If you find no issues with invoking your Lambda function or with the policy that the function returns, look for problems with your devices' connection attempts. Malformed connection requests can cause AWS IoT Core not to trigger your authorizer. Connection problems can occur at both the TLS and application layers.

Possible TLS layer issues:

- Customers must pass either a hostname header (HTTP, MQTT over WebSockets) or the Server Name Indication TLS extension (HTTP, MQTT over WebSockets, MQTT) in all custom authentication requests. In both cases, the value passed must match one of your account's AWS IoT Core data endpoints. These are the endpoints that are returned when you perform the following CLI commands.
  - `aws iot describe-endpoint —endpoint type iot:data-ats`
  - aws iot describe-endpoint —endpoint type (for legacy VeriSign endpoints)
- Devices that use custom authentication for MQTT connections must also pass the Application Layer Protocol Negotiation (ALPN) TLS extension with a value of `mqtt`.
- Custom authentication is currently available only on port 443.

Possible application layer issues:

- If signing is enabled (the `signingDisabled` field is false in your authorizer), look for the following signature issues.
  - Make sure that you're passing the token signature in either the `x-amz-customauthorizer-signature`header or in a query string parameter.
  - Make sure that the service isn't signing a value other than the token.
  - Make sure that you pass the token in the header or query parameter that you specified in the `token-key-name` field in your authorizer.
- Make sure that the authorizer name you pass in the `x-amz-customauthorizer-name` header or query string parameter is valid or that you have a default authorizer defined for your account.

Authorization

Authorization is the process of granting permissions to an authenticated identity. You grant permissions in AWS IoT Core using AWS IoT Core and IAM policies. This topic covers AWS IoT Core policies. For more information about IAM policies, see Identity and access management for AWS IoT (p. 277) and IAM policies (p. 282).

AWS IoT Core policies determine what an authenticated identity can do. An authenticated identity is used by devices, mobile applications, web applications, and desktop applications. An authenticated identity can even be a user typing AWS IoT Core CLI commands. An identity can execute AWS IoT Core operations only if it has a policy that grants it permission for those operations.

Both AWS IoT Core policies and IAM policies are used with AWS IoT Core to control the operations an identity (also called a *principal*) can perform. The policy type you use depends on the type of identity you are using to authenticate with AWS IoT Core.

AWS IoT Core operations are divided into two groups:

- Control plane API allows you to perform administrative tasks like creating or updating certificates, things, rules, and so on.
- Data plane API allows you send data to and receive data from AWS IoT Core.

The type of policy you use depends on whether you are using control plane or data plane API.

The following table shows the identity types, the protocols they use, and the policy types that can be used for authorization.

AWS IoT Core data plane API and policy types

Protocol and authentication mechanism	SDK	Identity type	Policy type		
MQTT over TLS/TCP, TLS mutual authentication (port 8883 or 443)† (p. 77)	AWS IoT Core Device SDK	X.509 certificates	AWS IoT Core policy		
MQTT over HTTPS/ WebSocket, AWS SigV4 authentication (port 443)	AWS Mobile SDK	Authenticated Amazon Cognito identity	IAM and AWS IoT Core policies		
		Unauthenticated Amazon Cognito identity	IAM policy		
		IAM, or federated identity	IAM policy		
HTTPS, AWS Signature Version 4 authentication (port 443)	AWS CLI	Amazon Cognito, IAM, or federated identity	IAM policy		
HTTPS, TLS mutual authentication (port 8443)	No SDK support	X.509 certificates	AWS IoT Core policy		
HTTPS over custom	AWS IoT Core Device SDK	Custom authorizer	Custom authorizer policy		

Protocol and authentication mechanism	SDK	Identity type	Policy type		
authentication (Port 443)					

AWS IoT Core control plane API and policy types

Protocol and authentication mechanism	SDK	Identity type	Policy type		
HTTPS AWS Signature Version 4 authentication (port 443)	AWS CLI	Amazon Cognito identity	IAM policy		
		IAM, or federated identity	IAM policy		

AWS IoT Core policies are attached to X.509 certificates or Amazon Cognito identities. IAM policies are attached to an IAM user, group, or role. If you use the AWS IoT console or the AWS IoT Core CLI to attach the policy (to a certificate or Amazon Cognito Identity), you use an AWS IoT Core policy. Otherwise, you use an IAM policy.

Policy-based authorization is a powerful tool. It gives you complete control over what a device, user, or application can do in AWS IoT Core. For example, consider a device connecting to AWS IoT Core with a certificate. You can allow the device to access all MQTT topics, or you can restrict its access to a single topic. In another example, consider a user typing CLI commands at the command line. By using a policy, you can allow or deny access to any command or AWS IoT Core resource for the user. You can also control an application's access to AWS IoT Core resources.

Changes made to a policy can take a few minutes to become effective because of how AWS IoT caches the policy documents. That is, it may take a few minutes to access a resource that has recently been granted access, and a resource may be accessible for several minutes after its access has been revoked.

AWS training and certification

For information about authorization in AWS IoT Core, take the Deep Dive into AWS IoT Core Authentication and Authorization course on the AWS Training and Certification website.

AWS IoT Core policies

AWS IoT Core policies are JSON documents. They follow the same conventions as IAM policies. AWS IoT Core supports named policies so many identities can reference the same policy document. Named policies are versioned so they can be easily rolled back.

AWS IoT Core policies allow you to control access to the AWS IoT Core data plane. The AWS IoT Core data plane consists of operations that allow you to connect to the AWS IoT Core message broker, send and receive MQTT messages, and get or update a device's shadow.

An AWS IoT Core policy is a JSON document that contains one or more policy statements. Each statement contains:

- `Effect`, which specifies whether the action is allowed or denied.

- `Action`, which specifies the action the policy is allowing or denying.
- `Resource`, which specifies the resource or resources on which the action is allowed or denied.

Changes made to a policy can take a few minutes to become effective because of how AWS IoT caches the policy documents. That is, it may take a few minutes to access a resource that has recently been granted access, and a resource may be accessisble for several minutes after its access has been revoked.

Topics

AWS IoT Core policy actions

The following policy actions are defined by AWS IoT Core:

MQTT Policy Actions

iot:Connect

Represents the permission to connect to the AWS IoT Core message broker. The `iot:Connect` permission is checked every time a `CONNECT` request is sent to the broker. The message broker does not allow two clients with the same client ID to stay connected at the same time. After the second client connects, the broker closes the existing connection. The `iot:Connect` permission can be used to ensure only authorized clients using a specific client ID can connect.

iot:Publish

Represents the permission to publish on an MQTT topic. This permission is checked every time a PUBLISH request is sent to the broker. This can be used to allow clients to publish to specific topic patterns.

Note
To grant `iot:Publish` permission, you must also grant `iot:Connect` permission.

iot:Receive

Represents the permission to receive a message from AWS IoT Core. The `iot:Receive` permission is checked every time a message is delivered to a client. Because this permission is checked on every delivery, it can be used to revoke permissions to clients that are currently subscribed to a topic.

iot:Subscribe

Represents the permission to subscribe to a topic filter. This permission is checked every time a SUBSCRIBE request is sent to the broker. This can be used to allow clients to subscribe to topics that match specific topic patterns.

Note
To grant `iot:Subscribe` permission, you must also grant `iot:Connect` permission.

Shadow Policy Actions

iot:DeleteThingShadow

Represents the permission to delete a device's shadow. The `iot:DeleteThingShadow` permission is checked every time a request is made to delete the shadow's contents.

iot:GetThingShadow

Represents the permission to retrieve a device's shadow. The `iot:GetThingShadow` permission is checked every time a request is made to retrieve the shadow's contents.

iot:UpdateThingShadow

Represents the permission to update a device's shadow. The `iot:UpdateThingShadow` permission is checked every time a request is made to update the shadow's contents.

Note
The job execution policy actions apply only for the HTTP TLS endpoint. If you use the MQTT endpoint, you must use MQTT policy actions defined in this topic.

Job Executions AWS IoT Core Policy Actions

iot:DescribeJobExecution

Represents the permission to retrieve a job execution for a given thing. The `iot:DescribeJobExecution` permission is checked every time a request is made to get a job execution.

iot:GetPendingJobExecutions

Represents the permission to retrieve the list of jobs that are not in a terminal status for a thing. The `iot:GetPendingJobExecutions` permission is checked every time a request is made to retrieve the list.

iot:UpdateJobExecution

Represents the permission to update a job execution. The `iot:UpdateJobExecution` permission is checked every time a request is made to update the state of a job execution.

iot:StartNextPendingJobExecution

Represents the permission to get and start the next pending job execution for a thing. (That is, to update a job execution with status QUEUED to IN_PROGRESS.) The `iot:StartNextPendingJobExecution` permission is checked every time a request is made to start the next pending job execution.

AWS IoT Core action resources

To specify a resource for an AWS IoT Core policy action, you must use the ARN of the resource. All resource ARNs are of the following form:

```
arn:aws:iot:region:AWS-account-ID:Resource-type/Resource-name
```

The following table shows the resource to specify for each action type:

Action	Resource type	Resource name	ARN example
`iot:DeleteThingShadow`	`thing`	The thing's name	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
`iot:Connect`	`client`	The client's client ID	`arn:aws:iot:us-east-1:123456789012:client/myClientId`

Action	Resource type	Resource name	ARN example
iot:Publish	topic	A topic string	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:Subscribe	topicfilter	A topic filter string	arn:aws:iot:us-east-1:123456789012:topicfilt myTopicFilter
iot:Receive	topic	A topic string	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:UpdateThingShadow	thing	The thing's name, and the shadow's name, if applicable	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:GetThingShadow	thing	The thing's name, and the shadow's name, if applicable	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:DescribeJobExecution	thing	The thing's name	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
iot:GetPendingJobExecutions	thing	The thing's name	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
iot:UpdateJobExecution	thing	The thing's name	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
iot:StartNextPendingJobExecution	thing	The thing's name	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`

AWS IoT Core policy variables

AWS IoT Core defines policy variables that can be used in AWS IoT Core policies in the `Resource` or `Condition` block. When a policy is evaluated, the policy variables are replaced by actual values. For example, if a device is connected to the AWS IoT Core message broker with a client ID of 100-234-3456, the `iot:ClientId` policy variable is replaced in the policy document by 100-234-3456. For more information about policy variables, see IAM Policy Variables and Multi-Value Conditions.

Basic AWS IoT Core policy variables

AWS IoT Core defines the following basic policy variables:

- `iot:ClientId`: The client ID used to connect to the AWS IoT Core message broker.
- `aws:SourceIp`: The IP address of the client connected to the AWS IoT Core message broker.

The following AWS IoT Core policy shows a policy that uses policy variables:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": [
          "arn:aws:iot:us-east-1:123451234510:client/${iot:ClientId}"
        ]
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Publish"],
        "Resource": [
          "arn:aws:iot:us-east-1:123451234510:topic/my/topic/${iot:ClientId}"
        ]
      }
    ]
}
```

In these examples, `${iot:ClientId}` is replaced by the ID of the client connected to the AWS IoT Core message broker when the policy is evaluated. When you use policy variables like `${iot:ClientId}`, you can inadvertently open access to unintended topics. For example, if you use a policy that uses `${iot:ClientId}` to specify a topic filter:

```
{
    "Effect": "Allow",
    "Action": ["iot:Subscribe"],
    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
    ]
}
```

A client can connect using + as the client ID. This would allow the user to subscribe to any topic that matches the topic filter `my/+/topic`. To protect against such security gaps, use the `iot:Connect` policy action to control which client IDs can connect. For example, this policy allows only those clients whose client ID is `clientid1` to connect:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/clientid1"
        ]
      }
    ]
}
```

Thing policy variables

Thing policy variables allow you to write AWS IoT Core policies that grant or deny permissions based on thing properties like thing names, thing types, and thing attribute values. You can use thing policy variables to apply the same policy to control many AWS IoT Core devices. For more information about device provisioning, see Device Provisioning. The thing name is obtained from the client ID in the MQTT `Connect` message sent when a thing connects to AWS IoT Core.

Keep the following in mind when using thing policy variables in AWS IoT Core policies.

- Use the AttachThingPrincipal API to attach certificates or principals (authenticated Amazon Cognito identities) to a thing.
- When you're replacing thing names with thing policy variables, the value of `clientId` in the MQTT connect message or the TLS connection must exactly match the thing name.

The following thing policy variables are available:

- `iot:Connection.Thing.ThingName`

  This resolves to the name of the thing in the AWS IoT Core registry for which the policy is being evaluated. AWS IoT Core uses the certificate the device presents when it authenticates to determine which thing to use to verify the connection. This policy variable is only available when a device connects over MQTT or MQTT over the WebSocket protocol.
- `iot:Connection.Thing.ThingTypeName`

  This resolves to the thing type associated with the thing for which the policy is being evaluated. The thing name is set to the client ID of the MQTT/WebSocket connection. The thing type name is obtained by a call to the `DescribeThing` API. This policy variable is available only when connecting over MQTT or MQTT over the WebSocket protocol.
- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  This resolves to the value of the specified attribute associated with the thing for which the policy is being evaluated. A thing can have up to 50 attributes. Each attribute is available as a policy variable: `iot:Connection.Thing.Attributes[`*`attributeName`*`]` where *`attributeName`* is the name of the attribute. The thing name is set to the client ID of the MQTT/WebSocket connection. This policy variable is only available when connecting over MQTT or MQTT over the WebSocket protocol.
- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]` enforces that only the devices that are both registered in AWS IoT and attached to principal can access the permissions inside the policy. You can use this variable to prevent a device from connecting to AWS IoT Core if it presents a certificate that is not attached to an IoT thing in the AWS IoT Core registry.This variable has values `true` or `false` indicating that the connecting thing is attached to the certificate or Amazon Cognito identity in the registry using AttachThingPrincipal API. Thing name is taken as client Id.

X.509 Certificate AWS IoT Core policy variables

X.509 certificate policy variables allow you to write AWS IoT Core policies that grant permissions based on X.509 certificate attributes. The following sections describe how you can use these certificate policy variables.

CertificateId

In the RegisterCertificate API, the `certificateId` appears in the response body. To get information about your certificate, you can use the `certificateId` in DescribeCertificate.

Issuer attributes

The following AWS IoT Core policy variables allow you to allow or deny permissions based on certificate attributes set by the certificate issuer.

- `iot:Certificate.Issuer.DistinguishedNameQualifier`
- `iot:Certificate.Issuer.Country`
- `iot:Certificate.Issuer.Organization`
- `iot:Certificate.Issuer.OrganizationalUnit`
- `iot:Certificate.Issuer.State`
- `iot:Certificate.Issuer.CommonName`
- `iot:Certificate.Issuer.SerialNumber`
- `iot:Certificate.Issuer.Title`
- `iot:Certificate.Issuer.Surname`
- `iot:Certificate.Issuer.GivenName`
- `iot:Certificate.Issuer.Initials`
- `iot:Certificate.Issuer.Pseudonym`
- `iot:Certificate.Issuer.GenerationQualifier`

Subject attributes

The following AWS IoT Core policy variables allow you to grant or deny permissions based on certificate subject attributes set by the certificate issuer.

- `iot:Certificate.Subject.DistinguishedNameQualifier`
- `iot:Certificate.Subject.Country`
- `iot:Certificate.Subject.Organization`
- `iot:Certificate.Subject.OrganizationalUnit`
- `iot:Certificate.Subject.State`
- `iot:Certificate.Subject.CommonName`
- `iot:Certificate.Subject.SerialNumber`
- `iot:Certificate.Subject.Title`
- `iot:Certificate.Subject.Surname`
- `iot:Certificate.Subject.GivenName`
- `iot:Certificate.Subject.Initials`
- `iot:Certificate.Subject.Pseudonym`
- `iot:Certificate.Subject.GenerationQualifier`

X.509 certificates allow these attributes to contain one or more values. By default, the policy variables for each multi-value attribute return the first value. For example, the `Certificate.Subject.Country` attribute might contain a list of country names, but when evaluated in a policy, `iot:Certificate.Subject.Country` is replaced by the first country name. You can request a specific attribute value other than the first value by using a one-based index. For example, `iot:Certificate.Subject.Country.1` is replaced by the second country name in the `Certificate.Subject.Country` attribute. If you specify an index value that does not exist (for example, if you ask for a third value when there are only two values assigned to the attribute), no substitution is made and authorization fails. You can use the `.List` suffix on the policy variable name to specify all values of the attribute.

Registered devices (2)

For devices registered as things in the AWS IoT Core registry, the following policy allows clients with a thing name registered in the AWS IoT Core registry to connect, but restricts the right to publish to a thing name specific topic to those clients with certificates whose `Certificate.Subject.Organization` attribute is set to `"Example Corp"` or `"AnyCompany"`.

This restriction is accomplished by using a `"Condition"` field that specifies a condition that must be met to allow the preceding action. In this case the condition is that the `Certificate.Subject.Organization` attribute associated with the certificate must include one of the values listed:

```
{
    "Version":"2012-10-17",
    "Statement":[
      {
        "Effect":"Allow",
        "Action":[
          "iot:Connect"
        ],
        "Resource":[
          "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
        ]
      },
      {
        "Effect":"Allow",
        "Action":[
          "iot:Publish"
        ],
        "Resource":[
          "arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"
        ],
        "Condition":{
          "ForAllValues:StringEquals":{
            "iot:Certificate.Subject.Organization.List":[
              "Example Corp",
              "AnyCompany"
            ]
          }
        }
      }
    ]
}
```

Unregistered devices (2)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client IDs `client1`, `client2`, and `client3`, but restricts the right to publish to a client-id specific topic to those clients with certificates whose `Certificate.Subject.Organization` attribute is set to `"Example Corp"` or `"AnyCompany"`. This restriction is accomplished by using a `"Condition"` field that specifies a condition that must be met to allow the preceding action. In this case the condition is that the `Certificate.Subject.Organization` attribute associated with the certificate must include one of the values listed:

```
{
    "Version":"2012-10-17",
    "Statement":[
      {
        "Effect":"Allow",
        "Action":[
          "iot:Connect"
        ],
        "Resource":[
          "arn:aws:iot:us-east-1:123456789012:client/client1",
          "arn:aws:iot:us-east-1:123456789012:client/client2",
          "arn:aws:iot:us-east-1:123456789012:client/client3"
        ]
      },
      {
        "Effect":"Allow",
        "Action":[
          "iot:Publish"
        ],
        "Resource":[
          "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
        ],
        "Condition":{
          "ForAllValues:StringEquals":{
            "iot:Certificate.Subject.Organization.List":[
              "Example Corp",
              "AnyCompany"
            ]
          }
        }
      }
    ]
}
```

Issuer alternate name attributes

The following AWS IoT Core policy variables allow you to grant or deny permissions based on issuer alternate name attributes set by the certificate issuer.

- `iot:Certificate.Issuer.AlternativeName.RFC822Name`
- `iot:Certificate.Issuer.AlternativeName.DNSName`
- `iot:Certificate.Issuer.AlternativeName.DirectoryName`
- `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Issuer.AlternativeName.IPAddress`

Subject alternate name attributes

The following AWS IoT Core policy variables allow you to grant or deny permissions based on subject alternate name attributes set by the certificate issuer.

- `iot:Certificate.Subject.AlternativeName.RFC822Name`
- `iot:Certificate.Subject.AlternativeName.DNSName`
- `iot:Certificate.Subject.AlternativeName.DirectoryName`
- `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Subject.AlternativeName.IPAddress`

Other attributes

You can use `iot:Certificate.SerialNumber` to allow or deny access to AWS IoT Core resources based on the serial number of a certificate. The `iot:Certificate.AvailableKeys` policy variable contains the name of all certificate policy variables that contain values.

X.509 Certificate policy variable limitations

The following limitations apply to X.509 certificate policy variables:

Wildcards

If wildcard characters are present in certificate attributes, the policy variable is not replaced by the certificate attribute value, leaving the `${policy-variable}` text in the policy document. This might cause authorization failure. The following wildcard characters can be used: `*`, `$`, `+`, `?`, and `#`.

Array fields

Certificate attributes that contain arrays are limited to five items. Additional items are ignored.

String length

All string values are limited to 1024 characters. If a certificate attribute contains a string longer than 1024 characters, the policy variable is not replaced by the certificate attribute value, leaving the `${policy-variable}` in the policy document. This might cause authorization failure.

Special Characters

Any special character, such as `,`, `"`, `\`, `+`, `=`, `<`, `>` and `;` must be prefixed with a backslash (`\`) when used in a policy variable. For example, `Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US` becomes `Amazon Web Service O=\Amazon.com Inc. L\=Seattle ST\=Washington C\=US`.

Example AWS IoT policies

See the following topics to learn about common elements in an AWS IoT policy. You can use the example policies to complete common tasks in AWS IoT.

Topics

Example policies

For additional policy examples, see the following topics in other sections of this guide:

- the section called “Identity-based policy examples” (p. 298)
- the section called “Basic AWS IoT Core policy variables” (p. 238)
- the section called “X.509 Certificate AWS IoT Core policy variables” (p. 240)

AWS IoT policy elements

AWS IoT policies are specified in a JSON document. An AWS IoT policy is composed of the following items:

Version

Must be set to `"2012-10-17"`.

Effect

Must be set to `"Allow"` or `"Deny"`.

Action

Must be set to `"iot:`*`operation-name`*`"` where *`operation-name`* is one of the following:

`"iot:Connect"`: Connect to AWS IoT.

`"iot:Receive"`: Receive messages from AWS IoT.

"iot:Publish": MQTT publish.

"iot:Subscribe": MQTT subscribe.

"iot:UpdateThingShadow": Update a device's shadow.

"iot:GetThingShadow": Retrieve a device's shadow.

"iot:DeleteThingShadow": Delete a device's shadow.

Resource

Must be set to one of the following:

Client: `arn:aws:iot:`*`region`*`:`*`account-id`*`:client/`*`client-id`*

Topic ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:topic/`*`topic-name`*

Topic filter ARN: `arn:aws:iot:`*`region`*`:`*`account-id`*`:topicfilter/`*`topic-filter`*

Connect policy examples

The following policy grants permission to connect to AWS IoT Core with client ID `client1`:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "iot:Connect"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/client1"
        ]
      }
    ]
}
```

The following policy denies permission to client IDs `client1` and `client2` to connect to AWS IoT Core, while allowing devices to connect using a client ID that matches the name of a thing registered in the AWS IoT Core registry:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Deny",
        "Action": [
          "iot:Connect"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/client1",
          "arn:aws:iot:us-east-1:123456789012:client/client2"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "iot:Connect"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
        ]
      }
    ]
}
```

MQTT persistent sessions policy examples

`connectAttributes` allow you to specify what attributes you want to use in your connect message in your IAM policies such as `PersistentConnect` and `LastWill`. For more information, see Using connectAttributes (p. 81)

The following policy allows connect with `PersistentConnect` feature:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

The following policy disallows `PersistentConnect`, other features are allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringNotEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
```

The above policy can also be expressed using `StringEquals`, any other feature including new feature is allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

The following policy allows connect by both `PersistentConnect` and `LastWill`, any other new feature is not allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect",
                        "LastWill"
                    ]
                }
            }
        }
    ]
}
```

The following policy allows clean connect by clients with or without `LastWill`, no other features will be allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
                }
            }
        }
    ]
}
```

The following policy only allows connect using default features:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                    ]
                }
            }
        }
    ]
}
```

The following policy allows connect only with `PersistentConnect`, any new feature is allowed as long as the connection uses `PersistentConnect`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

The following policy states the connect must have both `PersistentConnect` and `LastWill` usage, no new feature is allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect",
                        "LastWill"
                    ]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                    ]
                }
            }
        }
    ]
}
```

The following policy must not have `PersistentConnect` but can have `LastWill`, any other new feature is not allowed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
                }
            }
        }
    ]
}
```

The following policy allows connect only by clients that have a `LastWill` with topic `"my/lastwill/topicName"`, any feature is allowed as long as it uses the `LastWill` topic:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "iot:LastWillTopic": "arn:aws:iot:*region*:*account-id*:topic/*my/
lastwill/topicName*"
                }
            }
        }
    ]
}
```

The following policy only allows clean connect using a specific `LastWillTopic`, any feature is allowed as long as it uses the `LastWillTopic`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "iot:LastWillTopic": "arn:aws:iot:*region*:*account-id*:topic/*my/
lastwill/topicName*"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

Registered devices (3)

The following policy grants permission for a device to connect using its thing name as the client ID and to subscribe to the topic filter `my/topic/filter`. The device must be registered with AWS IoT Core. When the device connects to AWS IoT Core, it must provide the certificate associated with the IoT thing in the AWS IoT Core registry:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "iot:Connect"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "iot:Subscribe"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic/filter"
        ]
      }
    ]
}
```

Unregistered devices (3)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect using client ID `client1` and to subscribe to topic filter `my/topic`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic"
            ]
        }
    ]
}
```

Publish/Subscribe policy examples

The policy you use depends on how you are connecting to AWS IoT Core. You can connect to AWS IoT Core using an MQTT client, HTTP, or WebSocket. When you connect with an MQTT client, you are authenticating with an X.509 certificate. When you connect over HTTP or the WebSocket protocol, you are authenticating with Signature Version 4 and Amazon Cognito.

Policies for MQTT clients

To specify wildcards in topic names, use * in the `resource` attribute of the policy when the device publishes and subscribes to multiple topics. The following policy enables a device to publish to all subtopics that start with the same thing name.

Registered devices (5)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using a client ID that matches the thing name and to publish to any topic prefixed by the thing name:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

Unregistered devices (5)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client ID `client1`, `client2`, or `client3` and to publish to any topic prefixed by the client ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}/*"
            ]
        }
    ]
}
```

You can also use the * wildcard at the end of a topic filter. Using wildcard characters might lead to granting unintended privileges, so they should only be used after careful consideration. One situation in which they might be useful is when devices must subscribe to messages with many different topics (for example, if a device must subscribe to reports from temperature sensors in multiple locations).

Registered devices (6)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using the device's thing name as the client ID, and to subscribe to a topic prefixed by the thing name, followed by `room`, followed by any string. (It is expected that these topics are, for example, `thing1/room1`, `thing1/room2`, and so on):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/
${iot:Connection.Thing.ThingName}/room*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/room*"
            ]
        }
    ]
}
```

Unregistered devices (6)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client IDs `client1`, `client2`, `client3`, and to subscribe to a topic prefixed by the client ID, followed by `room`, followed by any string. (It is expected that these topics are, for example, `client1/room1`, `client1/room2`, and so on):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/${iot:ClientId}/room*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}/room*"
            ]
        }
    ]
}
```

When you specify topic filters in AWS IoT Core policies for MQTT clients, MQTT wildcard characters "+" and "#" are treated as literal characters. Their use might result in unexpected behavior.

Registered devices (4)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with the client ID that matches the thing name, and to subscribe to the topic filter `some/+/topic` only:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/some/+/topic"
            ]
        }
    ]
}
```

Unregistered devices (4)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client ID `client1` and subscribe to the topic filter `some/+/topic` only:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "iot:Connect"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/client1"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "iot:Subscribe"
        ],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:topicfilter/some/+/topic"
        ]
      }
    ]
}
```

Note
In a policy, the MQTT wildcard character + is treated as a literal, not a wildcard. Attempts to subscribe to topic filters that match the pattern `some/+/topic` fail and cause the client to disconnect.

Registered devices (7)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using the device's thing name as the client ID, and to subscribe to the topics `my/topic` and `my/othertopic`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/othertopic"
            ]
        }
    ]
}
```

Unregistered devices (7)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client ID `client1`, and to subscribe to the topics `my/topic` and `my/othertopic`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/othertopic"
            ]
        }
    ]
}
```

Registered devices (8)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using the device's thing name as the client ID and to subscribe to a topic unique to that thing name/client ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Thing.ThingName}"
            ]
        }
    ]
}
```

Unregistered devices (8)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client ID `client1`, and to publish to a topic unique to that client ID:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

Registered devices (9)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using the device's thing name as the client ID and to publish to any topic prefixed by that thing name or client except for one topic ending with `bar`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:Thing.ThingName}/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:Thing.ThingName}/bar"
            ]
        }
    ]
}
```

Unregistered devices (9)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client IDs `client1` and `client1` and to publish to any topic prefixed by the client ID used to connect, except for one topic ending with `bar`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}/bar"
            ]
        }
    ]
}
```

Registered devices (10)

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using the device's thing name as the client ID. The device can subscribe to the topic `my/topic`, but cannot publish to the *`thing-name`* `/bar` where *`thing-name`* is the name of the IoT thing connecting to AWS IoT Core:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${iot:Thing.ThingName}/bar"
            ]
        }
    ]
}
```

Unregistered devices (10)

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core using client ID `client1` and to subscribe to the topic `my/topic`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic"
            ]
        }
    ]
}
```

Thing policy variables are also replaced when a certificate or authenticated Amazon Cognito Identity is attached to a thing. The following policy grants permission to connect to AWS IoT Core with client ID `client1` and to publish and receive topic `iotmonitor/provisioning/987654321098`. It also allows the certificate holder to subscribe to this topic.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/iotmonitor/
provisioning/987654321098"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/iotmonitor/
provisioning/987654321098"
            ]
        }
    ]
}
```

Policies for HTTP and WebSocket clients

For the following operations, AWS IoT Core uses AWS IoT Core policies attached to Amazon Cognito identities (through the `AttachPolicy` API) to scope down the permissions attached to the Amazon Cognito Identity pool with authenticated identities. That means an Amazon Cognito Identity needs permission from the IAM role policy attached to the pool and the AWS IoT Core policy attached to the Amazon Cognito Identity through the AWS IoT Core `AttachPolicy` API.

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Note
For other AWS IoT Core operations or for unauthenticated identities, AWS IoT Core does not scope down the permissions attached to the Amazon Cognito identity pool role. For both authenticated and unauthenticated identities, this is the most permissive policy that we recommend you attach to the Amazon Cognito pool role.

HTTP

To allow unauthenticated Amazon Cognito identities to publish messages over HTTP on a topic specific to the Amazon Cognito Identity, attach the following policy to the Amazon Cognito Identity pool role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

To allow authenticated users, attach the preceding policy to the Amazon Cognito Identity pool role and to the Amazon Cognito Identity using the AWS IoT Core AttachPolicy API.

Note
When authorizing Amazon Cognito identities, AWS IoT Core considers both policies and grants the least privileges specified. An action is allowed only if both policies allow the requested action. If either policy disallows an action, that action is unauthorized.

MQTT

To allow unauthenticated Amazon Cognito identities to publish MQTT messages over WebSocket on a topic specific to the Amazon Cognito Identity in your account, attach the following policy to the Amazon Cognito Identity pool role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

To allow authenticated users, attach the preceding policy to the Amazon Cognito Identity pool role and to the Amazon Cognito Identity using the AWS IoT Core AttachPolicy API.

Note
When authorizing Amazon Cognito identities, AWS IoT Core considers both and grants the least privileges specified. An action is allowed only if both policies allow the requested action. If either policy disallows an action, that action is unauthorized.

Receive policy examples

Registered devices (11)

For devices registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with a client ID that matches the thing name and to subscribe to and receive messages on the `my/topic` topic:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic"
            ]
        }
    ]
}
```

Unregistered devices (11)

For devices not registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client ID `client1` and to subscribe to and receive messages on one topic:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/client1"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/my/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic"
            ]
        }
    ]
}
```

Connect and publish policy examples

For devices registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with a client ID that matches the thing name and restricts the device to publishing on a client-ID or thing name-specific MQTT topic. For a connection to be successful, the thing name must be registered in the AWS IoT Core registry and be authenticated using an identity or principal attached to the thing:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action":["iot:Publish"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}"]
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
      }
    ]
}
```

For devices not registered as things in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client ID `client1` and restricts the device to publishing on a clientID-specific MQTT topic:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action":["iot:Publish"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}"]
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:client/client1"]
      }
    ]
}
```

Certificate policy examples

For devices registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with a client ID that matches a thing name, and to publish to a topic whose name is equal to the `certificateId` of the certificate the device used to authenticate itself:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

For devices not registered in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client IDs, `client1`, `client2`, and `client3` and to publish to a topic whose name is equal to the `certificateId` of the certificate the device used to authenticate itself:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

For devices registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with a client ID that matches the thing name, and to publish to a topic whose name is equal to the subject's `CommonName` field of the certificate the device used to authenticate itself:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

Note
In this example, the certificate's subject common name is used as the topic identifier, with the assumption that the subject common name is unique for each registered certificate. If the certificates are shared across multiple devices, the subject common name is the same for all the

devices that share this certificate, thereby allowing publish privileges to the same topic from multiple devices (not recommended).

For devices not registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client IDs, `client1`, `client2`, and `client3` and to publish to a topic whose name is equal to the subject's `CommonName` field of the certificate the device used to authenticate itself:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

Note
In this example, the certificate's subject common name is used as the topic identifier, with the assumption that the subject common name is unique for each registered certificate. If the certificates are shared across multiple devices, the subject common name is the same for all the devices that share this certificate, thereby allowing publish privileges to the same topic from multiple devices (not recommended).

For devices registered in the AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with a client ID that matches the thing name, and to publish to a topic whose name is prefixed with `admin/` when the certificate used to authenticate the device has its `Subject.CommonName.2` field set to `Administrator`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
                }
            }
        }
    ]
}
```

For devices not registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client IDs `client1`, `client2`, and `client3` and to publish to a topic whose name is prefixed with `admin/` when the certificate used to authenticate the device has its `Subject.CommonName.2` field set to `Administrator`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
                }
            }
        }
    ]
}
```

For devices registered in AWS IoT Core registry, the following policy allows a device to use its thing name to publish on a specific topic that consists of `admin/` followed by the `ThingName` when the certificate used to authenticate the device has any one of its `Subject.CommonName` fields set to `Administrator`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
                }
            }
        }
    ]
}
```

For devices not registered in AWS IoT Core registry, the following policy grants permission to connect to AWS IoT Core with client IDs `client1`, `client2`, and `client3` and to publish to the topic `admin` when the certificate used to authenticate the device has any one of its `Subject.CommonName` fields set to `Administrator`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
                }
            }
        }
    ]
}
```

Thing policy examples

The following policy allows a device to connect if the certificate used to authenticate with AWS IoT Core is attached to the thing for which the policy is being evaluated:

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
                }
            }
        }
    ]
}
```

Authorization with Amazon Cognito identities

There are two types of Amazon Cognito identities: unauthenticated and authenticated. When your app supports unauthenticated Amazon Cognito identities, no authentication is performed so you do not know who the user is. For unauthenticated users, you grant permission by attaching an IAM role to an unauthenticated identity pool. You should grant access to only those resources you want available to unknown users.

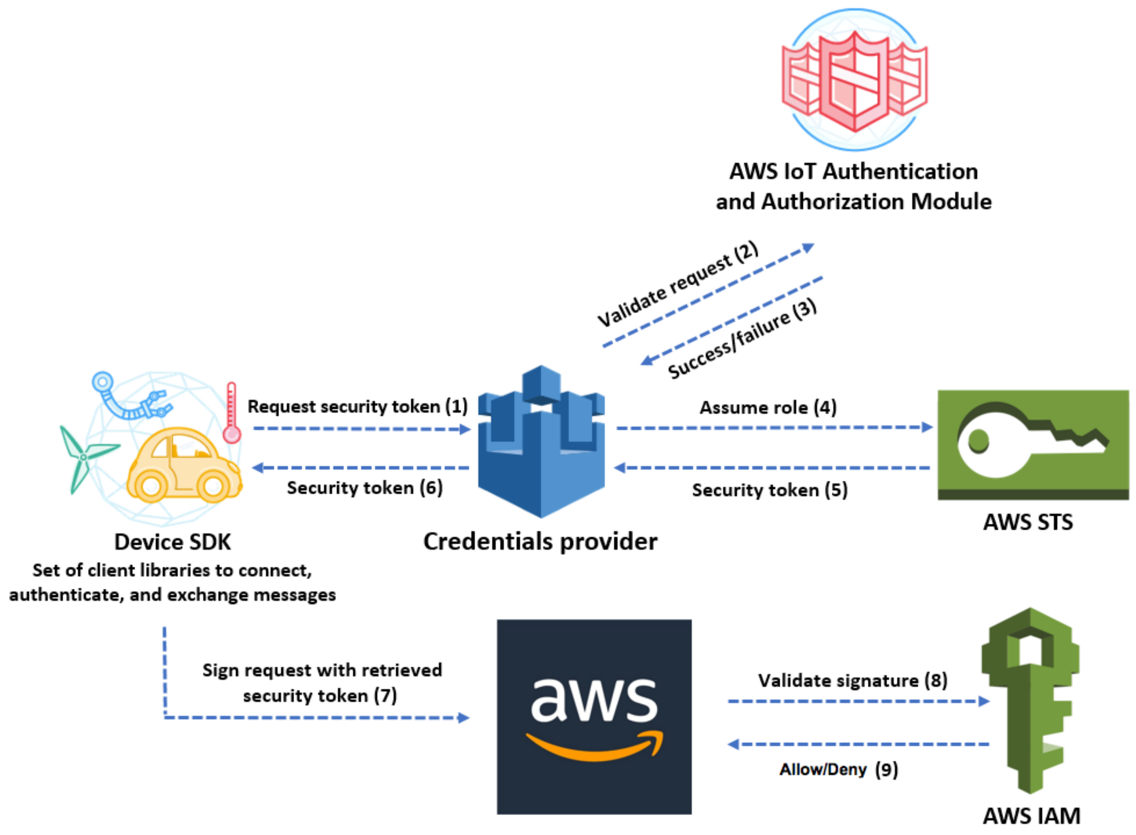
When your app supports authenticated Amazon Cognito identities, you specify a policy in two places. You attach an IAM policy to the authenticated Amazon Cognito Identity pool and you attach an AWS IoT Core policy to the Amazon Cognito Identity. You attach an IAM policy to a Amazon Cognito Identity pool using the Amazon Cognito Identity console when you create a Amazon Cognito Identity pool. To attach an AWS IoT Core policy to a Amazon Cognito Identity, you must define a Lambda function that calls `AttachPolicy`.

Authorizing direct calls to AWS services

Devices can use X.509 certificates to connect to AWS IoT Core using TLS mutual authentication protocols. Other AWS services do not support certificate-based authentication, but they can be called using AWS credentials in AWS Signature Version 4 format. The Signature Version 4 algorithm normally requires the caller to have an access key ID and a secret access key. AWS IoT Core has a credentials provider that allows you to use the built-in X.509 certificate as the unique device identity to authenticate AWS requests. This eliminates the need to store an access key ID and a secret access key on your device.

The credentials provider authenticates a caller using an X.509 certificate and issues a temporary, limited-privilege security token. The token can be used to sign and authenticate any AWS request. This way of authenticating your AWS requests requires you to create and configure an AWS Identity and Access Management (IAM) role and attach appropriate IAM policies to the role so that the credentials provider can assume the role on your behalf. For more information about AWS IoT Core and IAM, see Identity and access management for AWS IoT (p. 277).

The following diagram illustrates the credentials provider workflow.


AWS IoT Authentication and Authorization Module
Validate request (2)
Success/failure (3)
Request security token (1)
Security token (6)
Assume role (4)
Security token (5)
Device SDK
Set of client libraries to connect, authenticate, and exchange messages
Credentials provider
AWS STS
Sign request with retrieved security token (7)
aws
Validate signature (8)
Allow/Deny (9)
AWS IAM

1. The AWS IoT Core device makes an HTTPS request to the credentials provider for a security token. The request includes the device X.509 certificate for authentication.
2. The credentials provider forwards the request to the AWS IoT Core authentication and authorization module to validate the certificate and verify that the device has permission to request the security token.
3. If the certificate is valid and has permission to request a security token, the AWS IoT Core authentication and authorization module returns success. Otherwise, it sends an exception to the device.
4. After successfully validating the certificate, the credentials provider invokes the AWS Security Token Service (AWS STS) to assume the IAM role that you created for it.
5. AWS STS returns a temporary, limited-privilege security token to the credentials provider.
6. The credentials provider returns the security token to the device.
7. The device uses the security token to sign an AWS request with AWS Signature Version 4.
8. The requested service invokes IAM to validate the signature and authorize the request against access policies attached to the IAM role that you created for the credentials provider.
9. If IAM validates the signature successfully and authorizes the request, the request is successful. Otherwise, IAM sends an exception.

The following section describes how to use a certificate to get a security token. It is written with the assumption that you have already registered a device and created and activated your own certificate for it.

How to use a certificate to get a security token

1. Configure the IAM role that the credentials provider assumes on behalf of your device. Attach the following trust policy to the role.

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

   For each AWS service that you want to call, attach an access policy to the role. The credentials provider supports the following policy variables:

   - `credentials-iot:ThingName`
   - `credentials-iot:ThingTypeName`
   - `credentials-iot:AwsCertificateId`

   When the device provides the thing name in its request to an AWS service, the credentials provider adds `credentials-iot:ThingName` and `credentials-iot:ThingTypeName` as context variables to the security token. The credentials provider provides `credentials-iot:AwsCertificateId` as a context variable even if the device doesn't provide the thing name in the request. You pass the thing name as the value of the `x-amzn-iot-thingname` HTTP request header.

   These three variables work for IAM policies only, not AWS IoT Core policies.

2. Make sure that the user who performs the next step (creating a role alias) has permission to pass the newly created role to AWS IoT Core. The following policy gives both `iam:GetRole` and `iam:PassRole` permissions to an AWS user. The `iam:GetRole` permission allows the user to get information about the role that you've just created. The `iam:PassRole` permission allows the user to pass the role to another AWS service.

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

3. Create an AWS IoT Core role alias. The device that is going to make direct calls to AWS services must know which role ARN to use when connecting to AWS IoT Core. Hard-coding the role ARN is not a good solution because it requires you to update the device whenever the role ARN changes. A better solution is to use the `CreateRoleAlias` API to create a role alias that points to the role ARN. If the role ARN changes, you simply update the role alias. No change is required on the device. This API takes the following parameters:

`roleAlias`

Required. An arbitrary string that identifies the role alias. It serves as the primary key in the role alias data model. It contains 1-128 characters and must include only alphanumeric characters and the =, @, and - symbols. Uppercase and lowercase alphabetic characters are allowed.

`roleArn`

Required. The ARN of the role to which the role alias refers.

`credentialDurationInSeconds`

Optional. How long (in seconds) the credential is valid. The minimum value is 900 seconds (15 minutes). The maximum value is 3,600 seconds (60 minutes). The default value is 3,600 seconds.

Note
Although the credential lifetime specified in the IAM role can be longer, when AWS IoT Core Credential Provider issues the credential, its maximum lifetime is 3,600 seconds (60 minutes).

For more information about this API, see CreateRoleAlias.

4. Attach a policy to the device certificate. The policy attached to the device certificate must grant the device permission to assume the role. You do this by granting permission for the `iot:AssumeRoleWithCertificate` action to the role alias, as in the following example.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"iot:AssumeRoleWithCertificate",
         "Resource":"arn:aws:iot:your region:your_aws_account_id:rolealias/your role
 alias"
      }
   ]
}
```

5. Make an HTTPS request to the credentials provider to get a security token. Supply the following information:

   - *Certificate*: Because this is an HTTP request over TLS mutual authentication, you must provide the certificate and the private key to your client while making the request. Use the same certificate and private key you used when you registered your certificate with AWS IoT Core.

     To make sure your device is communicating with AWS IoT Core (and not a service impersonating it), see Server Authentication, follow the links to download the appropriate CA certificates, and then copy them to your device.
   - *RoleAlias*: The name of the role alias that you created for the credentials provider.
   - *ThingName*: The thing name that you created when you registered your AWS IoT Core thing. This is passed as the value of the `x-amzn-iot-thingname` HTTP header. This value is required only if you are using thing attributes as policy variables in AWS IoT Core or IAM policies.

   Run the following command in the AWS CLI to obtain the credentials provider endpoint for your AWS account. For more information about this API, see DescribeEndpoint.

   aws iot describe-endpoint --endpoint-type iot:CredentialProvider

The following JSON object is sample output of the **describe-endpoint** command. It contains the `endpointAddress` that you use to request a security token.

```
{
    "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
 region.amazonaws.com"
}
```

Use the endpoint to make an HTTPS request to the credentials provider to return a security token. The following example command uses `curl`, but you can use any HTTP client.

curl --cert *your certificate* --key *your device certificate key pair* -H "x-amzn-iot-thingname: *your thing name*" --cacert AmazonRootCA1.pem https://*your endpoint* /role-aliases/*your role alias*/credentials

This command returns a security token object that contains an `accessKeyId`, a `secretAccessKey`, a `sessionToken`, and an expiration. The following JSON object is sample output of the `curl` command.

```

    {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access
 key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

You can then use the `accessKeyId`, `secretAccessKey`, and `sessionToken` values to sign requests to AWS services. For an end-to-end demonstration, see How to Eliminate the Need for Hard-Coded AWS Credentials in Devices by Using the AWS IoT Credential Provider blog post on the *AWS Security Blog*.

Cross account access

AWS IoT Core allows you to enable a principal to publish or subscribe to a topic that is defined in an AWS account not owned by the principal. You configure cross account access by creating an IAM policy and IAM role and then attaching the policy to the role.

First, create a customer managed IAM policy as described in Creating IAM Policies, just like you would for other users and certificates in your AWS account.

For devices registered in AWS IoT Core registry, the following policy grants permission to devices connect to AWS IoT Core using a client ID that matches the device's thing name and to publish to the `my/topic/thing-name` where `thing-name` is the device's thing name:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"],
        }
    ]
}
```

For devices not registered in AWS IoT Core registry, the following policy grants permission to a device to use the thing name `client1` registered in your account's (123456789012) AWS IoT Core registry to connect to AWS IoT Core and to publish to a client ID-specific topic whose name is prefixed with `my/topic/`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

Next, follow the steps in Creating a Role to Delegate Permissions to an IAM User. Enter the account ID of the AWS account with which you want to share access. Then, in the final step, attach the policy you just created to the role. If, at a later time, you need to modify the AWS account ID to which you are granting access, you can use the following trust policy format to do so:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam:us-east-1:567890123456:user:MyUser"
            },
            "Action": "sts:AssumeRole",
        }
    ]
}
```

Data protection in AWS IoT Core

The AWS shared responsibility model applies to data protection in AWS IoT Core. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with AWS IoT or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into AWS IoT or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

AWS IoT devices gather data, perform some manipulation on that data, and then send that data to another web service. You might choose to store some data on your device for a short period of time. You're responsible for providing any data protection on that data at rest. When your device sends data to AWS IoT, it does so over a TLS connection as discussed later in this section. AWS IoT devices can send data to any AWS service. For more information about each service's data security, see the documentation for that service. AWS IoT can be configured to write logs to CloudWatch Logs and log AWS IoT API calls to AWS CloudTrail. For more information about data security for these services, see Authentication and Access Control for Amazon CloudWatch and Encrypting CloudTrail Log Files with AWS KMS-Managed Keys.

Data encryption in AWS IoT

By default, all AWS IoT data in transit and at rest is encrypted. Data in transit is encrypted using TLS (p. 275), and data at rest is encrypted using AWS owned keys. AWS IoT does not currently support customer-managed customer master keys (CMKs) from AWS Key Management Service (AWS KMS); however, Device Advisor and AWS IoT Wireless use only a KMS AWS Owned Customer Master Key (AOCMK) to encrypt customer data.

Transport security in AWS IoT

The AWS IoT message broker and Device Shadow service encrypt all communication while in-transit by using TLS version 1.2. TLS is used to ensure the confidentiality of the application protocols (MQTT, HTTP, and WebSocket) supported by AWS IoT. TLS support is available in a number of programming languages and operating systems. Data within AWS is encrypted by the specific AWS service. For more information about data encryption on other AWS services, see the security documentation for that service.

For MQTT, TLS encrypts the connection between the device and the broker. TLS client authentication is used by AWS IoT to identify devices. For HTTP, TLS encrypts the connection between the device and the broker. Authentication is delegated to AWS Signature Version 4.

AWS IoT requires devices to send the Server Name Indication (SNI) extension to the Transport Layer Security (TLS) protocol and provide the complete endpoint address in the `host_name` field. The `host_name` field must contain the endpoint you are calling, and it must be:

- The `endpointAddress` returned by **aws iot describe-endpoint --endpoint-type iot:Data-ATS**

  or
- The `domainName` returned by **aws iot describe-domain-configuration –-domain-configuration-name "*domain_configuration_name*"**

Connections attempted by devices without the correct `host_name` value will be refused and logged in CloudWatch.

AWS IoT does not support the SessionTicket TLS extension.

Transport security for LoRaWAN wireless devices

LoRaWAN devices follow the security practices described in LoRaWAN ™ SECURITY: A White Paper Prepared for the LoRa Alliance™ by Gemalto, Actility, and Semtech.

For more information about transport security with LoRaWAN devices, see Data security with AWS IoT Core for LoRaWAN (p. 115).

TLS cipher suite support

AWS IoT supports the following cipher suites:

- ECDHE-ECDSA-AES128-GCM-SHA256 (recommended)
- ECDHE-RSA-AES128-GCM-SHA256 (recommended)
- ECDHE-ECDSA-AES128-SHA256
- ECDHE-RSA-AES128-SHA256
- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-AES256-SHA384
- ECDHE-RSA-AES256-SHA384
- ECDHE-RSA-AES256-SHA
- ECDHE-ECDSA-AES256-SHA
- AES128-GCM-SHA256
- AES128-SHA256
- AES128-SHA
- AES256-GCM-SHA384
- AES256-SHA256
- AES256-SHA

Data encryption in AWS IoT

Data protection refers to protecting data while in-transit (as it travels to and from AWS IoT) and at rest (while it is stored on devices or by other AWS services). All data sent to AWS IoT is sent over an TLS connection using MQTT, HTTPS, and WebSocket protocols, making it secure by default while in transit. AWS IoT devices collect data and then send it to other AWS services for further processing. For more information about data encryption on other AWS services, see the security documentation for that service.

FreeRTOS provides a PKCS#11 library that abstracts key storage, accessing cryptographic objects and managing sessions. It is your responsibility to use this library to encrypt data at rest on your devices. For more information, see https://docs.aws.amazon.com/freertos/latest/userguide/security-pkcs.htmlFreeRTOS Public Key Cryptography Standard (PKCS) #11 Library

Device Advisor

Encryption in transit

Data sent to and from Device Advisor is encrypted in transit. All data sent to and from the service when using the Device Advisor APIs is encrypted using Signature Version 4. For more information about how AWS API requests are signed, see Signing AWS API requests. All data sent from your test devices to your Device Advisor test endpoint is sent over a TLS connection so it is secure by default in transit.

Key management in AWS IoT

All connections to AWS IoT are done using TLS, so no client-side encryption keys are necessary for the initial TLS connection.

Devices must authenticate using an X.509 certificate or an Amazon Cognito Identity. You can have AWS IoT generate a certificate for you, in which case it will generate a public/private key pair. If you are using the AWS IoT console you will be prompted to download the certificate and keys. If you are using the `create-keys-and-certificate` CLI command, the certificate and keys are returned by the CLI command. You are responsible for copying the certificate and private key onto your device and keeping it safe.

AWS IoT does not currently support customer-managed customer master keys (CMKs) from AWS Key Management Service (AWS KMS); however, Device Advisor and AWS IoT Wireless uses only a KMS AWS Owned Customer Master Key (AOCMK) to encrypt customer data.

Device Advisor

All data sent to Device Advisor when using the AWS APIs is encrypted at rest. Device Advisor encrypts all of your data at rest using AWS KMS customer master keys (CMKs) stored and managed in AWS Key Management Service. Device Advisor encrypts your data using AWS-owned customer master keys. For more information about AWS owned customer master keys, see AWS-owned CMKs.

Identity and access management for AWS IoT

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS IoT resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS IoT.

Service user – If you use the AWS IoT service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS IoT, see Troubleshooting AWS IoT identity and access (p. 301).

Service administrator – If you're in charge of AWS IoT resources at your company, you probably have full access to AWS IoT. It's your job to determine which AWS IoT features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT, see How AWS IoT works with IAM (p. 282).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS IoT. To view example AWS IoT identity-based policies that you can use in IAM, see AWS IoT identity-based policy examples (p. 298).

Authenticating with IAM identities

In AWS IoT identities can be device (X.509) certificates, Amazon Cognito identities, or IAM users or groups. This topic discusses IAM identities only. For more information about the other identities that AWS IoT supports, see Client authentication (p. 203).

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you

might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for AWS IoT in the *Service Authorization Reference*.

  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS IoT works with IAM

Before you use IAM to manage access to AWS IoT, you should understand which IAM features are available to use with AWS IoT. To get a high-level view of how AWS IoT and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

IAM policies

AWS IoT works with AWS IoT and IAM policies. This topic discusses IAM policies only. For more information, see AWS IoT Core policies (p. 235). AWS Identity and Access Management defines a policy action for each operation defined by AWS IoT, including control plane and data plane APIs.

IAM managed policies

AWS IoT provides a set of IAM managed policies you can either use as-is or as a starting point for creating custom IAM policies. These policies allow access to configuration and data operations. Configuration operations allow you to create things, certificates, policies, and rules. Data operations send data over MQTT or HTTP protocols. The following table describes these templates.

Policy template	Description
AWSIoTConfigAccess	Allows the associated identity access to all AWS IoT configuration operations. This policy can affect data processing and storage.
AWSIoTConfigReadOnlyAccess	Allows the associated identity to access read-only configuration operations.
AWSIoTDataAccess	Allows the associated identity full access to all AWS IoT data operations. Data operations send data over MQTT or HTTP protocols.
AWSIoTEventsFullAccess	Allows the associated identity full access to AWS IoT events.
AWSIoTEventsReadOnlyAccess	Allows the associated identity read only access to AWS IoT events.
AWSIoTFullAccess	Allows the associated identity full access to all AWS IoT configuration and messaging operations.
AWSIoTLogging	Allows the associated identity to create Amazon CloudWatch Logs groups and stream logs to the groups. This policy is attached to your CloudWatch logging role.
AWSIoTOTAUpdate	Allows the associated identity to create AWS IoT jobs, AWS IoT code signing jobs, and to describe AWS code signer jobs.

Policy template	Description
AWSIoTRuleActions	Allows the associated identity access to all AWS services supported in AWS IoT rule actions.
AWSIoTThingsRegistration	Allows the associated identity to register things in bulk using the StartThingRegistrationTask API. This policy can affect data processing and storage.
AWSIoTWirelessDataAccess	Allows the associated identity to send data to AWS IoT wireless devices.
AWSIoTWirelessFullAccess	Allows the associated identity full access to AWS IoT Wireless.
AWSIoTWirelessFullPublishAccess	Grants AWS IoT Wireless limited access to publish to AWS IoT rules on your behalf.
AWSIoTWirelessLogging	Allows the associated identity to create Amazon CloudWatch log groups and stream logs to the groups. This policy is attached to your CloudWatch logging role.
AWSIoTWirelessReadOnlyAccess	Allows the associated identity read only access to AWS IoT Wireless.
AWSIoTWirelessGatewayCertManager	Allows the associated identity access to create, list, and describe AWS IoT certificates.

AWS IoT identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. AWS IoT supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

The following table lists the IAM IoT actions, the associated AWS IoT API, and the resource the action manipulates.

Policy actions	AWS IoT API	Resources
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id*

Policy actions	AWS IoT API	Resources
		Note The AWS account specified in the ARN must be the account to which the certificate is being transferred.
iot:AddThingToThingGroup	AddThingToThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	None
iot:AttachPolicy	AttachPolicy	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* or arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:AttachSecurityProfile	AttachSecurityProfile	arn:aws:iot:*region*:*account-id*:securityprofile/*security-profile-name* arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:AttachThingPrincipal	AttachThingPrincipal	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id* **Note** The AWS account specified in the ARN must be the account to which the certificate is being transferred.
iot:CancelJob	CancelJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot:*region*:*account-id*:job/*job-id* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	None
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
iot:CreateDimension	CreateDimension	arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:CreateJob	CreateJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*
iot:CreatePolicy	CreatePolicy	*
iot:CreatePolicyVersion	CreatePolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name* **Note** This must be an AWS IoT policy, not an IAM policy.

Policy actions	AWS IoT API	Resources
iot:CreateRoleAlias	CreateRoleAlias	(parameter: roleAlias) arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:CreateSecurityProfile	CreateSecurityProfile	arn:aws:iot:*region*:*account-id*:securityprofile/*security-profile-name* arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:CreateThing	CreateThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:CreateThingGroup	CreateThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* for group being created and for parent group, if used
iot:CreateThingType	CreateThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:CreateTopicRule	CreateTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DeleteDimension	DeleteDimension	arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:DeleteJob	DeleteJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot:*region*:*account-id*:job/*job-id* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DeletePolicy	DeletePolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:DeleteSecurityProfile	DeleteSecurityProfile	arn:aws:iot:*region*:*account-id*:securityprofile/*security-profile-name* arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:DeleteThing	DeleteThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*

Policy actions	AWS IoT API	Resources
iot:DeleteThingType	DeleteThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name* (parameter: authorizerName) none
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	None
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	none
iot:DescribeIndex	DescribeIndex	arn:aws:iot:*region*:*account-id*:index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	None
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	None
iot:DescribeThingType	DescribeThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:DetachPolicy	DetachPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id* or arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DetachSecurityProfile	DetachSecurityProfile	arn:aws:iot:*region*:*account-id*:securityprofile/*security-profile-name* arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*

Policy actions	AWS IoT API	Resources
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:GetIndexingConfiguration	GetIndexingConfiguration	None
iot:GetJobDocument	GetJobDocument	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* or arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	None
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	None
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	None
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	None
iot:ListJobs	ListJobs	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* if thingGroupName parameter used
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListPolicyVersions	ListPolicyVersions	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListPrincipalPolicies	ListPrincipalPolicies	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListPrincipalThings	ListPrincipalThings	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListRoleAliases	ListRoleAliases	None

Policy actions	AWS IoT API	Resources
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListThingGroups	ListThingGroups	None
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	None
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	None
iot:ListThingTypes	ListThingTypes	*
iot:ListThings	ListThings	*
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:ListTopicRules	ListTopicRules	*
iot:ListV2LoggingLevels	ListV2LoggingLevels	None
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	None
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot:*region*:*account-id*:index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name*
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	arn:aws:iot:*region*:*account-id*:role/*role-name*
iot:SetV2LoggingLevel	SetV2LoggingLevel	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:SetV2LoggingOptions	SetV2LoggingOptions	arn:aws:iot:*region*:*account-id*:role/*role-name*
iot:StartThingRegistrationTask	StartThingRegistrationTask	None
iot:StopThingRegistrationTask	StopThingRegistrationTask	None
iot:TestAuthorization	TestAuthorization	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	None
iot:TransferCertificate	TransferCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*

Policy actions	AWS IoT API	Resources
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot:*region*:*account-id*:authorizerfunction/*authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:UpdateDimension	UpdateDimension	arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:UpdateEventConfigurations	UpdateEventConfigurations	None
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	None
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:UpdateSecurityProfile	UpdateSecurityProfile	arn:aws:iot:*region*:*account-id*:securityprofile/*security-profile-name* arn:aws:iot:*region*:*account-id*:dimension/*dimension-name*
iot:UpdateThing	UpdateThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*

Policy actions in AWS IoT use the following prefix before the action: `iot:`. For example, to grant someone permission to list all IoT things registered in their AWS account with the `ListThings` API, you include the `iot:ListThings` action in their policy. Policy statements must include either an `Action` or `NotAction` element. AWS IoT defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "iot:Describe*"
```

To see a list of AWS IoT actions, see Actions Defined by AWS IoT in the *IAM User Guide*.

Device Advisor actions

The following table lists the IAM IoT Device Advisor actions, the associated AWS IoT Device Advisor API, and the resource the action manipulates.

Policy actions	AWS IoT API	Resources
iotdeviceadvisor:CreateSuiteDefinition	CreateSuiteDefinition	None
iotdeviceadvisor:DeleteSuiteDefinition	DeleteSuiteDefinition	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:DescribeSuiteDefinition	DescribeSuiteDefinition	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:DescribeSuiteRun	DescribeSuiteRun	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:GetSuiteRunReport	GetSuiteRunReport	arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-definition-id*/*suite-run-id*
iotdeviceadvisor:ListSuiteDefinitions	ListSuiteDefinitions	None
iotdeviceadvisor:ListSuiteRuns	ListSuiteRuns	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
ListTagsForResource	ListTagsForResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-run-id*
iotdeviceadvisor:ListTestCases	ListTestCases	None
iotdeviceadvisor:StartSuiteRun	StartSuiteRun	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:TagResource	TagResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-run-id*
iotdeviceadvisor:UntagResource	UntagResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-run-id*
iotdeviceadvisor:UpdateSuiteDefinition	UpdateSuiteDefinition	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*

Policy actions in AWS IoT Device Advisor use the following prefix before the action: `iotdeviceadvisor:`. For example, to grant someone permission to list all suite definitions registered in their AWS account with the ListSuiteDefinitions API, you include the `iotdeviceadvisor:ListSuiteDefinitions` action in their policy.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

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

AWS IoT resources

Policy actions	AWS IoT API	Resources
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id* **Note** The AWS account specified in the ARN must be the account to which the certificate is being transferred.
iot:AddThingToThingGroup	AddThingToThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	None
iot:AttachPolicy	AttachPolicy	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* or arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:AttachThingPrincipal	AttachThingPrincipal	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id* **Note** The AWS account specified in the ARN must be the account to which the certificate is being transferred.
iot:CancelJob	CancelJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot:*region*:*account-id*:job/*job-id* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	None
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
iot:CreateJob	CreateJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*
iot:CreatePolicy	CreatePolicy	*
CreatePolicyVersion	iot:CreatePolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*

Policy actions	AWS IoT API	Resources
		Note This must be an AWS IoT policy, not an IAM policy.
iot:CreateRoleAlias	CreateRoleAlias	(parameter: roleAlias) arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:CreateThing	CreateThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:CreateThingGroup	CreateThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* for group being created and for parent group, if used
iot:CreateThingType	CreateThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:CreateTopicRule	CreateTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DeleteJob	DeleteJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot:*region*:*account-id*:job/*job-id* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DeletePolicy	DeletePolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:DeleteThing	DeleteThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DeleteThingType	DeleteThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*

Policy actions	AWS IoT API	Resources
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name* (parameter: authorizerName) none
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	None
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	none
iot:DescribeIndex	DescribeIndex	arn:aws:iot:*region*:*account-id*:index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	None
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	None
iot:DescribeThingType	DescribeThingType	arn:aws:iot:*region*:*account-id*:thingtype/*thing-type-name*
iot:DetachPolicy	DetachPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id* or arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:GetIndexingConfiguration	GetIndexingConfiguration	None
iot:GetJobDocument	GetJobDocument	arn:aws:iot:*region*:*account-id*:job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*

Policy actions	AWS IoT API	Resources
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* or arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	None
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	None
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	None
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	None
iot:ListJobs	ListJobs	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* if thingGroupName parameter used
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListPolicyVersions	ListPolicyVersions	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListPrincipalPolicies	ListPrincipalPolicies	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListPrincipalThings	ListPrincipalThings	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:ListRoleAliases	ListRoleAliases	None
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:ListThingGroups	ListThingGroups	None
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	None
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	None
iot:ListThingTypes	ListThingTypes	*
iot:ListThings	ListThings	*

Policy actions	AWS IoT API	Resources
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:ListTopicRules	ListTopicRules	*
iot:ListV2LoggingLevels	ListV2LoggingLevels	None
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	None
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name* arn:aws:iot:*region*:*account-id*:thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot:*region*:*account-id*:rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot:*region*:*account-id*:index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot:*region*:*account-id*:authorizer/*authorizer-function-name*
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot:*region*:*account-id*:policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	arn:aws:iot:*region*:*account-id*:role/*role-name*
iot:SetV2LoggingLevel	SetV2LoggingLevel	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:SetV2LoggingOptions	SetV2LoggingOptions	arn:aws:iot:*region*:*account-id*:role/*role-name*
iot:StartThingRegistrationTask	StartThingRegistrationTask	None
iot:StopThingRegistrationTask	StopThingRegistrationTask	None
iot:TestAuthorization	TestAuthorization	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	None
iot:TransferCertificate	TransferCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot:*region*:*account-id*:authorizerfunction/*authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot:*region*:*account-id*:cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot:*region*:*account-id*:cert/*cert-id*
iot:UpdateEventConfigurations	UpdateEventConfigurations	None
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	None
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot:*region*:*account-id*:rolealias/*role-alias-name*
iot:UpdateThing	UpdateThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*

Policy actions	AWS IoT API	Resources
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot:*region*:*account-id*:thinggroup/*thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot:*region*:*account-id*:thing/*thing-name*

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

Some AWS IoT actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

To see a list of AWS IoT resource types and their ARNs, see Resources Defined by AWS IoT in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS IoT.

Device Advisor resources

To define resource-level restrictions for AWS IoT Device Advisor IAM policies, use the following resource ARN formats for suite definitions and suite runs.

Suite definition resource ARN format

arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*

Suite run resource ARN format

arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-definition-id*/*suite-run-id*

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

AWS IoT defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

AWS IoT condition keys

AWS IoT condition keys	Description	Type
`aws:RequestTag/${tag-key}`	A tag key that is present in the request that the user makes to AWS IoT.	String
`aws:ResourceTag/${tag-key}`	The tag key component of a tag attached to an AWS IoT resource.	String
`aws:TagKeys`	The list of all the tag key names associated with the resource in the request.	String

To see a list of AWS IoT condition keys, see Condition Keys for AWS IoT in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS IoT.

Examples

To view examples of AWS IoT identity-based policies, see AWS IoT identity-based policy examples (p. 298).

AWS IoT resource-based policies

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on the AWS IoT resource and under what conditions.

AWS IoT does not support IAM resource-based policies. It does, however, support AWS IoT resource-based policies.

Authorization based on AWS IoT tags

You can attach tags to AWS IoT resources or pass tags in a request to AWS IoT. To control access based on tags, you provide tag information in the condition element of a policy using the `iot:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information, see Using tags with IAM policies (p. 195). For more information about tagging AWS IoT resources, see Tagging your AWS IoT resources (p. 194).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Viewing AWS IoT resources based on tags (p. 300).

AWS IoT IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with AWS IoT

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

AWS IoT supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

AWS IoT does not supports service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

AWS IoT identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS IoT resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS IoT resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using AWS IoT quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the AWS IoT console

To access the AWS IoT console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the AWS IoT console, also attach the following AWS managed policy to the entities: `AWSIoTFullAccess`. For more information, see Adding Permissions to a User in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

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

Viewing AWS IoT resources based on tags

You can use conditions in your identity-based policy to control access to AWS IoT resources based on tags. This example shows how you might create a policy that allows viewing a thing. However, permission is granted only if the thing tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListBillingGroupsInConsole",
            "Effect": "Allow",
            "Action": "iot:ListBillingGroups",
            "Resource": "*"
        },
        {
            "Sid": "ViewBillingGroupsIfOwner",
            "Effect": "Allow",
           "Action": "iot:DescribeBillingGroup",
            "Resource": "arn:aws:iot:*:*:billinggroup/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to view an AWS IoT billing group, the billing group must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise, he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Viewing AWS IoT Device Advisor resources based on tags

You can use conditions in your identity-based policy to control access to AWS IoT Device Advisor resources based on tags. The following example shows how you can create a policy that allows viewing a particular suite definition. However, permission is granted only if the suite definition tag has `SuiteType` set to the value of `MQTT`. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
            }
        }
    ]
}
```

Troubleshooting AWS IoT identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT and IAM.

Topics

I am not authorized to perform an action in AWS IoT

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a thing but does not have `iot:`*`DescribeThing`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing
           on resource: MyIoTThing
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`MyIoTThing`* resource using the `iot:`*`DescribeThing`* action.

Using AWS IoT Device Advisor

If you're using AWS IoT Device Advisor, the following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a suite definition but does not have `iotdeviceadvisor:DescribeSuiteDefinition` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:DescribeSuiteDefinition
           on resource: MySuiteDefinition
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`MySuiteDefinition`* resource using the `iotdeviceadvisor:`*`DescribeSuiteDefinition`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS IoT.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS IoT. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

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

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access AWS IoT

To allow others to access AWS IoT, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS IoT.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my AWS IoT resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS IoT supports these features, see How AWS IoT works with IAM (p. 282).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and Monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT and your AWS solutions. You should collect monitoring data from all parts of your AWS solution so that you can more easily debug a multi-point failure, if one occurs. For information on logging and monitoring procedures, see Monitoring AWS IoT (p. 310)

Monitoring Tools

AWS provides tools that you can use to monitor AWS IoT. You can configure some of these tools to do the monitoring for you. Some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use the following automated monitoring tools to watch AWS IoT and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods. For more information, see Monitor AWS IoT alarms and metrics using Amazon CloudWatch (p. 317).
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. Amazon CloudWatch Logs also allows you to see critical steps AWS IoT Device Advisor test cases take, generated events and MQTT messages sent from your devices or AWS IoT Core during test execution. These logs make it possible to debug and take corrective actions on your devices. For more information, see Monitor AWS IoT using CloudWatch Logs (p. 330) For more information about using Amazon CloudWatch, see Monitoring Log Files in the *Amazon CloudWatch User Guide*.
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see What Is Amazon CloudWatch Events in the *Amazon CloudWatch User Guide*.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Log AWS IoT API calls using AWS CloudTrail (p. 349) and also Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.

Manual Monitoring Tools

Another important part of monitoring AWS IoT involves manually monitoring those items that the CloudWatch alarms don't cover. The AWS IoT, CloudWatch, and other AWS service console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on AWS IoT.

- AWS IoT dashboard shows:
  - CA certificates
  - Certificates
  - Polices
  - Rules
  - Things

- CloudWatch home page shows:
  - Current alarms and status.
  - Graphs of alarms and resources.
  - Service health status.

  You can use CloudWatch to do the following:
  - Create customized dashboards to monitor the services you care about.
  - Graph metric data to troubleshoot issues and discover trends.
  - Search and browse all your AWS resource metrics.
  - Create and edit alarms to be notified of problems.

Compliance validation for AWS IoT Core

Third-party auditors assess the security and compliance of AWS IoT Core as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS IoT is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS IoT Core

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

AWS IoT Core stores information about your devices in the device registry. It also stores CA certificates, device certificates, and device shadow data. This data is not automatically replicated in the event of hardware or network failures. AWS IoT Core publishes MQTT events when the device registry is updated. You can use these messages to back up your registry data and save it somewhere, like a DynamoDB table. You are responsible for saving certificates that AWS IoT Core creates for you or those you create yourself. Device shadow stores state data about your devices and can be resent when a device comes back online. AWS IoT Device Advisor stores information about your test suite configuration. This data is automatically replicated in the event of hardware or network failures.

AWS IoT Core resources are Region-specific and aren't replicated across AWS Regions unless you specifically do so.

Infrastructure security in AWS IoT

As a collection of managed services, AWS IoT is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS IoT through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems, such as Java 7 and later, support these modes. For more information, see Transport security in AWS IoT (p. 275).

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Vulnerability analysis and management in AWS IoT Core

IoT fleets can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make fleet setup complex and error-prone. And because devices are often constrained in computational power, memory, and storage capabilities, this limits the use of encryption and other forms of security on the devices themselves. Also, devices often use software with known vulnerabilities. These factors make IoT fleets an attractive target for hackers and make it difficult to secure your device fleet on an ongoing basis.

AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. You can use AWS IoT Device Defender to analyze, audit, and monitor connected devices to detect abnormal behavior, and mitigate security risks. AWS IoT Device Defender can audit device fleets to ensure they adhere to security best practices and detect abnormal behavior on devices. This makes it possible to enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised. For more information, see AWS IoT Device Defender (p. 714).

AWS IoT Device Advisor pushes updates and patches as needed. AWS IoT Device Advisor update test cases automatically. The test cases you select are always with latest version.

Security best practices in AWS IoT Core

This section contains information about security best practices for AWS IoT Core. For more information, see Ten security golden rules for IoT solutions.

Protecting MQTT connections in AWS IoT

AWS IoT Core is a managed cloud service that makes it possible for connected devices to interact with cloud applications and other devices easily and securely. AWS IoT Core supports HTTP, WebSocket, and MQTT, a lightweight communication protocol specifically designed to tolerate intermittent connections. If you are connecting to AWS IoT using MQTT, each of your connections must be associated with an identifier known as a client ID. MQTT client IDs uniquely identify MQTT connections. If a new connection is established using a client ID that is already claimed for another connection, the AWS IoT message broker drops the old connection to allow the new connection. Client IDs must be unique within each AWS account and each AWS Region. This means that you don't need to enforce global uniqueness of client IDs outside of your AWS account or across Regions within your AWS account.

The impact and severity of dropping MQTT connections on your device fleet depends on many factors. These include:

- Your use case (for example, the data your devices send to AWS IoT, how much data, and the frequency that the data is sent).
- Your MQTT client configuration (for example, auto reconnect settings, associated back-off timings, and use of MQTT persistent sessions (p. 80)).
- Device resource constraints.
- The root cause of the disconnections, its aggressiveness, and persistence.

To avoid client ID conflicts and their potential negative impacts, make sure that each device or mobile application has an AWS IoT or IAM policy that restricts which client IDs can be used for MQTT connections to the AWS IoT message broker. For example, you can use an IAM policy to prevent a device from unintentionally closing another device's connection by using a client ID that is already in use. For more information, see Authorization (p. 233).

All devices in your fleet must have credentials with privileges that authorize intended actions only, which include (but not limited to) AWS IoT MQTT actions such as publishing messages or subscribing to topics with specific scope and context. The specific permission policies can vary for your use cases. Identify the permission policies that best meet your business and security requirements.

To simplify creation and management of permission policies, you can use AWS IoT Core policy variables (p. 238) and IAM policy variables. Policy variables can be placed in a policy and when the policy is evaluated, the variables are replaced by values that come from the device's request. Using policy variables, you can create a single policy for granting permissions to multiple devices. You can identify the relevant policy variables for your use case based on your AWS IoT account configuration, authentication mechanism, and network protocol used in connecting to AWS IoT message broker. However, to write the best permission policies, consider the specifics of your use case and your threat model.

For example, if you registered your devices in the AWS IoT registry, you can use thing policy variables (p. 239) in AWS IoT policies to grant or deny permissions based on thing properties like thing names, thing types, and thing attribute values. The thing name is obtained from the client ID in the MQTT connect message sent when a thing connects to AWS IoT. The thing policy variables are replaced when a thing connects to AWS IoT over MQTT using TLS mutual authentication or MQTT over the WebSocket protocol using authenticated Amazon Cognito identities. You can use the AttachThingPrincipal API to attach certificates and authenticated Amazon Cognito identities to a thing. `iot:Connection.Thing.ThingName` is a useful thing policy variable to enforce client ID restrictions. The following example AWS IoT policy requires a registered thing's name to be used as the client ID for MQTT connections to the AWS IoT message broker:

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
        "Effect":"Allow",
        "Action":"iot:Connect",
        "Resource":[
            "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
        ]
    }
    ]
}
```

If you want to identify ongoing client ID conflicts, you can enable and use CloudWatch Logs for AWS IoT (p. 330). For every MQTT connection that the AWS IoT message broker disconnects due to client ID conflicts, a log record similar to the following is generated:

```
{
    "timestamp": "2019-04-28 22:05:30.105",
    "logLevel": "ERROR",
    "traceId": "02a04a93-0b3a-b608-a27c-1ae8ebdb032a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "clientId01",
    "principalId": "1670fcf6de55adc1930169142405c4a2493d9eb5487127cd0091ca0193a3d3f6",
    "sourceIp": "203.0.113.1",
    "sourcePort": 21335,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID"
}
```

You can use a CloudWatch Logs filter such as `{$.reason= "DUPLICATE_CLIENT_ID" }` to search for instances of client ID conflicts or to set up CloudWatch metric filters and corresponding CloudWatch alarms for continuous monitoring and reporting.

You can use AWS IoT Device Defender to identify overly permissive AWS IoT and IAM policies. AWS IoT Device Defender also provides an audit check that notifies you if multiple devices in your fleet are connecting to the AWS IoT message broker using the same client ID.

You can use AWS IoT Device Advisor to validate that your devices can reliably connect to AWS IoT Core and follow security best practices.

See also

- AWS IoT Core
- AWS IoT's Security Features (p. 200)
- AWS IoT Core policy variables (p. 238)
- IAM Policy Variables
- Amazon Cognito Identity
- AWS IoT Device Defender
- CloudWatch Logs for AWS IoT (p. 330)

Keep your device's clock in sync

It's important to have an accurate time on your device. X.509 certificates have an expiry date and time. The clock on your device is used to verify that a server certificate is still valid. If you're building commercial IoT devices, remember that your products may be stored for extended periods before being sold. Real-time clocks can drift during this time and batteries can get discharged, so setting time in the factory is not sufficient.

For most systems, this means that the device's software must include a network time protocol (NTP) client. The device should wait until it synchronizes with an NTP server before it tries to connect to AWS IoT Core. If this isn't possible, the system should provide a way for a user to set the device's time so that subsequent connections succeed.

After the device synchronizes with an NTP server, it can open a connection with AWS IoT Core. How much clock skew that is allowed depends on what you're trying to do with the connection.

Validate the server certificate

The first thing a device does to interact with AWS IoT is to open a secure connection. When you connect your device to AWS IoT, ensure that you're talking to AWS IoT and not another server impersonating AWS IoT. Each of the AWS IoT servers is provisioned with a certificate issued for the `iot.amazonaws.com` domain. This certificate was issued to AWS IoT by a trusted certificate authority that verified our identity and ownership of the domain.

One of the first things AWS IoT Core does when a device connects is send the device a server certificate. Devices can verify that they were expecting to connect to `iot.amazonaws.com` and that the server on the end of that connection possesses a certificate from a trusted authority for that domain.

TLS certificates are in X.509 format and include a variety of information such as the organization's name, location, domain name, and a validity period. The validity period is specified as a pair of time values called `notBefore` and `notAfter`. Services like AWS IoT Core use limited validity periods (for example, one year) for their server certificates and begin serving new ones before the old ones expire.

Use a single identity per device

Use a single identity per client. Devices generally use X.509 client certificates. Web and mobile applications use Amazon Cognito Identity. This enables you to apply fine-grained permissions to your devices.

For example, you have an application that consists of a mobile phone device that receives status updates from two different smart home objects – a light bulb and a thermostat. The light bulb sends the status of its battery level, and a thermostat sends messages that report the temperature.

AWS IoT authenticates devices individually and treats each connection individually. You can apply fine-grained access controls using authorization policies. You can define a policy for the thermostat that allows it to publish to a topic space. You can define a separate policy for the light bulb that allows it to publish to a different topic space. Finally, you can define a policy for the mobile app that only allows it to connect and subscribe to the topics for the thermostat and the light bulb to receive messages from these devices.

Apply the principle of least privilege and scope down the permissions per device as much as possible. All devices or users should have an AWS IoT policy in AWS IoT that only allows it to connect with a known client ID, and to publish and subscribe to an identified and fixed set of topics.

Use just in time provisioning

Manually creating and provisioning each device can be time consuming. AWS IoT provides a way to define a template to provision devices when they first connect to AWS IoT. For more information, see Just-in-time provisioning (p. 666).

Permissions to run AWS IoT Device Advisor tests

The policy template below shows the minimum permissions and IAM entity required to run AWS IoT Device Advisor test cases. You will need to replace *`your-device-role-arn`* with the device role ARN you create under the prerequisites.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "your-device-role-arn",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iotdeviceadvisor.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "logs:DescribeLogStreams",
                "iot:DescribeThing",
                "iot:DescribeCertificate",
                "logs:CreateLogGroup",
                "logs:PutLogEvents",
                "iot:DescribeEndpoint",
                "execute-api:Invoke*",
                "logs:CreateLogStream",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:ListCertificates"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        }
    ]
}
```

AWS training and certification

Take the following course to learn about key concepts for AWS IoT security: AWS IoT Security Primer.

Monitoring AWS IoT

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT and your AWS solutions.

We strongly encourage you to collect monitoring data from all parts of your AWS solution to make it easier to debug a multi-point failure, if one occurs. Start by creating a monitoring plan that answers the following questions. If you're not sure how to answer these, you can still continue to enable logging (p. 310) and establish your performance baselines.

- What are your monitoring goals?
- Which resources will you monitor?
- How often will you monitor these resources?
- Which monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

Your next step is to enable logging (p. 310) and establish a baseline of normal AWS IoT performance in your environment by measuring performance at various times and under different load conditions. As you monitor AWS IoT, keep historical monitoring data so that you can compare it with current performance data. This will help you identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish your baseline performance for AWS IoT, you should monitor these metrics to start. You can always monitor more metrics later.

- `PublishIn.Success` (p. 324)
- `PublishOut.Success` (p. 324)
- `Subscribe.Success` (p. 324)
- `Ping.Success` (p. 324)
- `Connect.Success` (p. 324)
- `GetThingShadow.Accepted` (p. 326)
- `UpdateThingShadow.Accepted` (p. 326)
- `DeleteThingShadow.Accepted` (p. 326)
- `RulesExecuted` (p. 322)

The topics in this section can help you start logging and monitoring AWS IoT.

Topics

Configure AWS IoT logging

You must enable logging by using the AWS IoT console, CLI, or API before you can monitor and log AWS IoT activity.

You can enable logging for all of AWS IoT or only specific thing groups. You can configure AWS IoT logging by using the AWS IoT console, CLI, or API; however, you must use the CLI or API to configure logging for specific thing groups.

When considering how to configure your AWS IoT logging, the default logging configuration determines how AWS IoT activity will be logged unless specified otherwise. Starting out, you might want to obtain detailed logs with a default log level (p. 317) of `INFO` or `DEBUG`. After reviewing the initial logs, you can change the default log level to a less verbose level such as `WARN` or `ERROR` and set a more verbose resource-specific log level on resources that might need more attention. Log levels can be changed whenever you want.

Configure logging role and policy

Before you can enable logging in AWS IoT, you must create an IAM role and a policy that gives AWS permission to monitor AWS IoT activity on your behalf.

> **Note**
> Before you enable AWS IoT logging, make sure you understand the CloudWatch Logs access permissions. Users with access to CloudWatch Logs can see debugging information from your devices. For more information, see Authentication and Access Control for Amazon CloudWatch Logs.
> If you expect high traffic patterns in AWS IoT Core due to load testing, consider turning off IoT logging to prevent throttling. If high traffic is detected, our service may disable logging in your account.

Create a logging role

Use the IAM console to create a logging role.

To create a logging role for AWS IoT Core using the IAM console

1. Open the Roles hub of the IAM console, and then choose **Create role**.
2. To create a logging role for AWS IoT Core:

   a. Under **Select type of trusted entity**, choose **AWS Service**, **IoT**.
   b. Under **Select your use case**, choose **IoT**, and then choose **Next: Permissions**.
   c. On the page that displays the policies that are automatically attached to the service role.
   d. Choose **Next: Tags**, and then choose **Next: Review**.
   e. Enter a **Role name** and **Role description** for the role, and then choose **Create role**.
   f. In the list of **Roles**, find the role you created, open it, and copy the **Role ARN** (*`logging-role-arn`*) to use when you Configure default logging in the AWS IoT (console) (p. 313).
3. To create a logging role for AWS IoT Core for LoRaWAN:

   a. Under **Select type of trusted entity**, choose **Another AWS account**.
   b. In **Account ID**, enter your AWS account ID, and then choose **Next: Permissions**.
   c. In the search box, enter **`AWSIoTWirelessLogging`**.
   d. Select the box next to the policy named **AWSIoTWirelessLogging**, and then choose **Next: Tags**.
   e. Choose **Next: Review**.
   f. In **Role name**, enter **`IoTWirelessLogsRole`**, and then choose **Create role**.
   g. In the confirmation message, choose **IoTWirelessLogsRole**.
   h. In the role's **Summary** page, choose the **Trust relationships** tab, and choose **Edit trust relationship**.
   i. Edit the `Principal` entry so that it contains

```
"Service": "iotwireless.amazonaws.com"
```

as this example shows:

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

j. Choose **Update Trust Policy**. Logging for AWS IoT Core for LoRaWAN is now configured.

Logging role policy

The following policy documents provide the role policy and trust policy that allow AWS IoT, and optionally, AWS IoT Core for LoRaWAN, to submit log entries to CloudWatch on your behalf.

Note
These documents were created for you when you created the logging role.

Role policy:

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

Role policy for logging only AWS IoT Core for LoRaWAN activity:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/iotwireless*"
        }
    ]
}
```

Trust policy to log only AWS IoT Core activity:

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

Trust policy to log AWS IoT Core and AWS IoT Core for LoRaWAN activity:

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

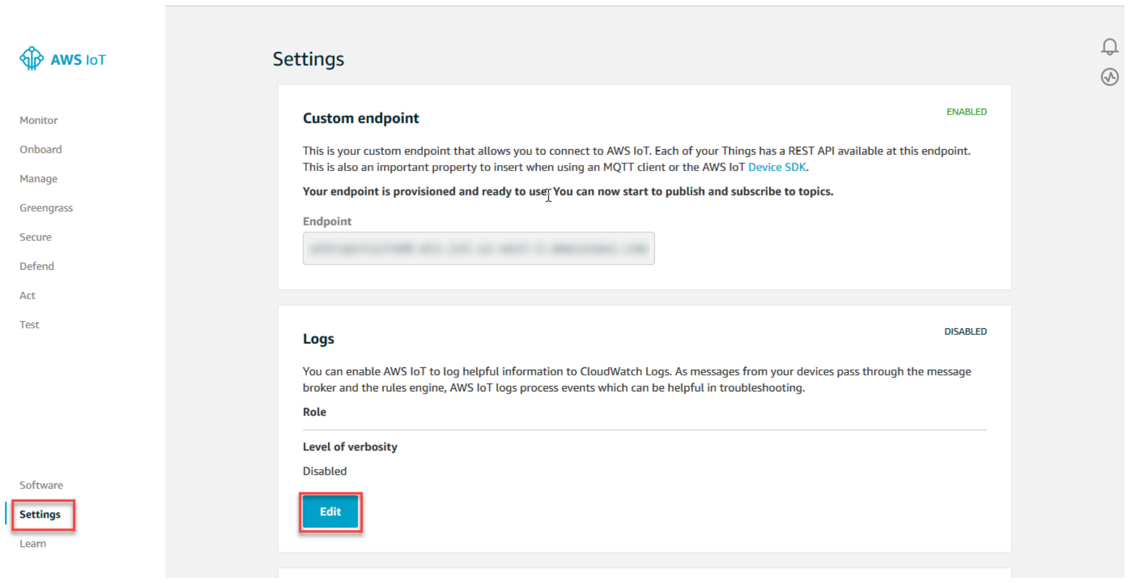
Configure default logging in the AWS IoT (console)

This section describes how use the AWS IoT console to configure logging for all of AWS IoT. To configure logging for only specific thing groups, you must use the CLI or API. For information about configuring logging for specific thing groups, see Configure resource-specific logging in AWS IoT (CLI) (p. 315).

To use the AWS IoT console to configure default logging for all of AWS IoT

1. Sign in to the AWS IoT console. For more information, see Open the AWS IoT console (p. 19).
2. In the left navigation pane, choose **Settings**. In the **Logs** section of the **Settings** page, choose **Edit**.

   The **Logs** section displays the logging role and level of verbosity used by all of AWS IoT.



Settings

Custom endpoint ENABLED

This is your custom endpoint that allows you to connect to AWS IoT. Each of your Things has a REST API available at this endpoint. This is also an important property to insert when using an MQTT client or the AWS IoT Device SDK.

Your endpoint is provisioned and ready to use. You can now start to publish and subscribe to topics.

Endpoint

Logs DISABLED

You can enable AWS IoT to log helpful information to CloudWatch Logs. As messages from your devices pass through the message broker and the rules engine, AWS IoT logs process events which can be helpful in troubleshooting.

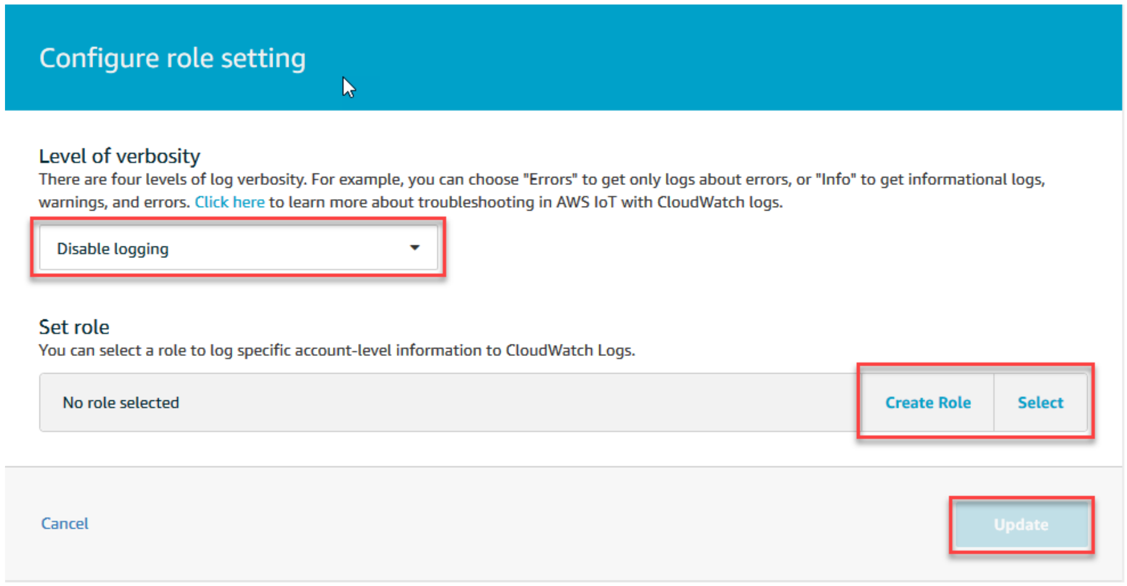
Role

Level of verbosity

Disabled

Edit

3. On the **Configure role setting** page, choose the **Level of verbosity** that describes the level of detail (p. 317) of the log entries that you want to appear in the CloudWatch logs.



Configure role setting

Level of verbosity

There are four levels of log verbosity. For example, you can choose "Errors" to get only logs about errors, or "Info" to get informational logs, warnings, and errors. Click here to learn more about troubleshooting in AWS IoT with CloudWatch logs.

Disable logging

Set role

You can select a role to log specific account-level information to CloudWatch Logs.

No role selected

4. Choose **Select** to specify a role that you created in Create a logging role (p. 311), or **Create Role** to create a new role to use for logging.
5. Choose **Update** to save your changes.

After you've enabled logging, visit Viewing AWS IoT logs in the CloudWatch console (p. 330) to learn more about viewing the log entries.

Configure default logging in AWS IoT (CLI)

This section describes how to configure global logging for AWS IoT by using the CLI.

> **Note**
> You need the Amazon Resource Name (ARN) of the role that you want to use. If you need to create a role to use for logging, see Create a logging role (p. 311) before continuing.

The principal used to call the API must have Pass role permissions (p. 354) for your logging role.

You can also perform this procedure with the API by using the methods in the AWS API that correspond to the CLI commands shown here.

To use the CLI to configure default logging for AWS IoT

1. Use the **set-v2-logging-options** command to set the logging options for your account.

```
aws iot set-v2-logging-options \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

   where:

   --role-arn

      The role ARN that grants AWS IoT permission to write to your logs in CloudWatch Logs.

   --default-log-level

      The log level (p. 317) to use. Valid values are: `ERROR`, `WARN`, `INFO`, `DEBUG`, or `DISABLED`

   --no-disable-all-logs

      An optional parameter that enables all AWS IoT logging. Use this parameter to enable logging when it is currently disabled.

   --disable-all-logs

      An optional parameter that disables all AWS IoT logging. Use this parameter to disable logging when it is currently enabled.

2. Use the **get-v2-logging-options** command to get your current logging options.

```
aws iot get-v2-logging-options
```

After you've enabled logging, visit Viewing AWS IoT logs in the CloudWatch console (p. 330) to learn more about viewing the log entries.

Note
AWS IoT continues to support older commands (**set-logging-options** and **get-logging-options**) to set and get global logging on your account. Be aware that when these commands are used, the resulting logs contain plain-text, rather than JSON payloads and logging latency is generally higher. No further improvements will be made to the implementation of these older commands. We recommend that you use the "v2" versions to configure your logging options and, when possible, change legacy applications that use the older versions.

Configure resource-specific logging in AWS IoT (CLI)

This section describes how to configure resource-specific logging for AWS IoT by using the CLI. Resource-specific logging allows you to specify a logging level for a specific thing group (p. 179).

Thing groups can contain other thing groups to create a hierarchical relationship. This procedure describes how to configure the logging of a single thing group. You can apply this procedure to the parent thing group in a hierarchy to configure the logging for all thing groups in the hierarchy. You can also apply this procedure to a child thing group to override the logging configuration of its parent.

Note
You need the Amazon Resource Name (ARN) of the role you want to use. If you need to create a role to use for logging, see Create a logging role (p. 311) before continuing.
The principal used to call the API must have Pass role permissions (p. 354) for your logging role.

You can also perform this procedure with the API by using the methods in the AWS API that correspond to the CLI commands shown here.

To use the CLI to configure resource-specific logging for AWS IoT

1. Use the **set-v2-logging-options** command to set the logging options for your account.

```
aws iot set-v2-logging-options \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

   where:

   --role-arn

   The role ARN that grants AWS IoT permission to write to your logs in CloudWatch Logs.

   --default-log-level

   The log level (p. 317) to use. Valid values are: `ERROR`, `WARN`, `INFO`, `DEBUG`, or `DISABLED`

   --no-disable-all-logs

   An optional parameter that enables all AWS IoT logging. Use this parameter to enable logging when it is currently disabled.

   --disable-all-logs

   An optional parameter that disables all AWS IoT logging. Use this parameter to disable logging when it is currently enabled.

2. Use the **set-v2-logging-level** command to configure resource-specific logging for a thing group.

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

   --log-target

   The type and name of the resource for which you are configuring logging. The `target_type` value must be `THING_GROUP`. The log-target parameter value can be text, as shown in the preceding command example, or a JSON string, such as the following example.

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
              --log-level log_level
```

   --log-level

   The logging level used when generating logs for the specified resource. Valid values are: **DEBUG**, **INFO**, **ERROR**, **WARN**, and **DISABLED**

3. Use the **list-v2-logging-levels** command to list the currently configured logging levels.

```
aws iot list-v2-logging-levels
```

4. Use the **delete-v2-logging-level** command to delete a resource-specific logging level.

```
aws iot delete-v2-logging-level \
              --targetType "THING_GROUP" \
              --targetName "thing_group_name"
```

--targetType

The `target_type` value must be `THING_GROUP`

--targetName

The name of the thing group for which to remove the logging level.

After you've enabled logging, visit Viewing AWS IoT logs in the CloudWatch console (p. 330) to learn more about viewing the log entries.

Log levels

These log levels determine the events that are logged and apply to default and resource-specific log levels.

ERROR

Any error that causes an operation to fail.

Logs include ERROR information only.

WARN

Anything that can potentially cause inconsistencies in the system, but might not cause the operation to fail.

Logs include ERROR and WARN information.

INFO

High-level information about the flow of things.

Logs include INFO, ERROR, and WARN information.

DEBUG

Information that might be helpful when debugging a problem.

Logs include DEBUG, INFO, ERROR, and WARN information.

DISABLED

All logging is disabled.

Monitor AWS IoT alarms and metrics using Amazon CloudWatch

You can monitor AWS IoT using CloudWatch, which collects and processes raw data from AWS IoT into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, AWS IoT metric data is sent automatically to CloudWatch in one minute intervals. For more information, see What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs? in the *Amazon CloudWatch User Guide*.

Using AWS IoT metrics

The metrics reported by AWS IoT provide information that you can analyze in different ways. The following use cases are based on a scenario where you have ten things that connect to the internet once a day. Each day:

- Ten things connect to AWS IoT at roughly the same time.
- Each thing subscribes to a topic filter, and then waits for an hour before disconnecting. During this period, things communicate with one another and learn more about the state of the world.
- Each thing publishes some perception it has based on its newly found data using `UpdateThingShadow`.
- Each thing disconnects from AWS IoT.

To help you get started, these topics explore some of the questions that you might have.

- How can I be notified if my things do not connect successfully each day? (p. 319)
- How can I be notified if my things are not publishing data each day? (p. 319)
- How can I be notified if my thing's shadow updates are being rejected each day? (p. 320)
- How can I create a CloudWatch alarm for Jobs? (p. 320)

More about CloudWatch alarms and metrics

- Creating CloudWatch alarms to monitor AWS IoT (p. 318)
- AWS IoT metrics and dimensions (p. 321)

Creating CloudWatch alarms to monitor AWS IoT

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period you specify. When the value of the metric exceeds a given threshold over a number of time periods, one or more actions are performed. The action can be a notification sent to an Amazon SNS topic or Auto Scaling policy. Alarms trigger actions for sustained state changes only. CloudWatch alarms do not trigger actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

The following topics describe some examples of using CloudWatch alarms.

- How can I be notified if my things do not connect successfully each day? (p. 319)
- How can I be notified if my things are not publishing data each day? (p. 319)
- How can I be notified if my thing's shadow updates are being rejected each day? (p. 320)
- How can I create a CloudWatch alarm for jobs? (p. 320)

You can see all the metrics that CloudWatch alarms can monitor at AWS IoT metrics and dimensions (p. 321).

How can I be notified if my things do not connect successfully each day?

1. Create an Amazon SNS topic named `things-not-connecting-successfully`, and record its Amazon Resource Name (ARN). This procedure will refer to your topic's ARN as *`sns-topic-arn`*.

   For more information on how to create an Amazon SNS notification, see Getting Started with Amazon SNS.
2. Create the alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name ConnectSuccessAlarm \
    --alarm-description "Alarm when my Things don't connect successfully" \
    --namespace AWS/IoT \
    --metric-name Connect.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Test the alarm.

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. Verify that the alarm appears in your CloudWatch console.

How can I be notified if my things are not publishing data each day?

1. Create an Amazon SNS topic named `things-not-publishing-data`, and record its Amazon Resource Name (ARN). This procedure will refer to your topic's ARN as *`sns-topic-arn`*.

   For more information on how to create an Amazon SNS notification, see Getting Started with Amazon SNS.
2. Create the alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name PublishInSuccessAlarm\
    --alarm-description "Alarm when my Things don't publish their data \
    --namespace AWS/IoT \
    --metric-name PublishIn.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Test the alarm.

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. Verify that the alarm appears in your CloudWatch console.

How can I be notified if my thing's shadow updates are being rejected each day?

1. Create an Amazon SNS topic named `things-shadow-updates-rejected`, and record its Amazon Resource Name (ARN). This procedure will refer to your topic's ARN as *`sns-topic-arn`*.

   For more information on how to create an Amazon SNS notification, see Getting Started with Amazon SNS.
2. Create the alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name UpdateThingShadowSuccessAlarm \
    --alarm-description "Alarm when my Things Shadow updates are getting rejected" \
    --namespace AWS/IoT \
    --metric-name UpdateThingShadow.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Test the alarm.

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value ALARM
```

4. Verify that the alarm appears in your CloudWatch console.

How can I create a CloudWatch alarm for jobs?

The Jobs service provides CloudWatch metrics for you to monitor your jobs. You can create CloudWatch alarms to monitor any Jobs metrics (p. 326).

The following command creates a CloudWatch alarm to monitor the total number of failed job executions for Job *`SampleOTAJob`* and notifies you when more than 20 job executions have failed. The alarm monitors the Jobs metric `FailedJobExecutionTotalCount` by checking the reported value every 300 seconds. It is activated when a single reported value is greater than 20, meaning there were more than 20 failed job executions since the job started. When the alarm goes off, it sends a notification to the provided Amazon SNS topic.

```
aws cloudwatch put-metric-alarm \
    --alarm-name TotalFailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when total number of failed job execution exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionTotalCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 20 \
    --comparison-operator GreaterThanThreshold \
    --period 300 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-many-
failed-job-ececutions
```

The following command creates a CloudWatch alarm to monitor the number of failed job executions for Job *`SampleOTAJob`* in a given period. It then notifies you when more than five job executions have failed during that period. The alarm monitors the Jobs metric `FailedJobExecutionCount` by checking the reported value every 3600 seconds. It is activated when a single reported value is greater than 5, meaning there were more than 5 failed job executions in the past hour. When the alarm goes off, it sends a notification to the provided Amazon SNS topic.

```
aws cloudwatch put-metric-alarm \
    --alarm-name FailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when number of failed job execution per hour exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --period 3600 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-many-
failed-job-ececutions-per-hour
```

AWS IoT metrics and dimensions

When you interact with AWS IoT, the service sends the following metrics and dimensions to CloudWatch every minute. You can use the following procedures to view the metrics for AWS IoT.

To view metrics (CloudWatch console)

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane, under the metrics category for AWS IoT, choose a metrics category, and then in the upper pane, scroll down to view the full list of metrics.

To view metrics (CLI)

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch displays the following groups of metrics for AWS IoT:

AWS IoT metrics

Metric	Description
`AddThingToDynamicThingGroupsFailed`	The number of failure events associated with adding a thing to a dynamic thing group. The `DynamicThingGroupName` dimension contains the name of the dynamic groups that failed to add things.
`NumLogBatchesFailedToPublishThrottled`	The singular batch of log events that has failed to publish due to throttling errors.
`NumLogEventsFailedToPublishThrottled`	The number of log events within the batch that have failed to publish due to throttling errors.
`RulesExecuted`	The number of AWS IoT rules executed.

Rule metrics

Metric	Description
`ParseError`	The number of JSON parse errors that occurred in messages published on a topic on which a rule is listening. The `RuleName` dimension contains the name of the rule.
`RuleMessageThrottled`	The number of messages throttled by the rules engine because of malicious behavior or because the number of messages exceeds the rules engine's throttle limit. The `RuleName` dimension contains the name of the rule to be triggered.

Metric	Description
`RuleNotFound`	The rule to be triggered could not be found. The `RuleName` dimension contains the name of the rule.
`TopicMatch`	The number of incoming messages published on a topic on which a rule is listening. The `RuleName` dimension contains the name of the rule.

Rule action metrics

Metric	Description
`Failure`	The number of failed rule action invocations. The `RuleName` dimension contains the name of the rule that specifies the action. The `RuleName` dimension contains the name of the rule that specifies the action. The `ActionType` dimension contains the type of action that was invoked.
`Success`	The number of successful rule action invocations. The `RuleName` dimension contains the name of the rule that specifies the action. The `ActionType` dimension contains the type of action that was invoked.

HTTP action specific metrics

Metric	Description
`HttpCode_Other`	Generated if the status code of the response from the downstream web service/application is not 2xx, 4xx or 5xx.
`HttpCode_4XX`	Generated if the status code of the response from the downstream web service/application is between 400 and 499.
`HttpCode_5XX`	Generated if the status code of the response from the downstream web service/application is between 500 and 599.
`HttpInvalidUrl`	Generated if an endpoint URL, after substitution templates are replaced, does not start with `https://`.
`HttpRequestTimeout`	Generated if the downstream web service/application does not return response within request timeout limit. For more information, see Service Quotas.
`HttpUnknownHost`	Generated if the URL is valid, but the service does not exist or is unreachable.

Message broker metrics

Metric	Description
`Connect.AuthError`	The number of connection requests that could not be authorized by the message broker. The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Connect.ClientError`	The number of connection requests rejected because the MQTT message did not meet the requirements defined in AWS IoT quotas (p. 1006). The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Connect.ClientIDThrottle`	The number of connection requests throttled because the client exceeded the allowed connect request rate for a specific client ID. The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Connect.ServerError`	The number of connection requests that failed because an internal error occurred. The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Connect.Success`	The number of successful connections to the message broker. The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Connect.Throttle`	The number of connection requests that were throttled because the account exceeded the allowed connect request rate. The `Protocol` dimension contains the protocol used to send the `CONNECT` message.
`Ping.Success`	The number of ping messages received by the message broker. The `Protocol` dimension contains the protocol used to send the ping message.
`PublishIn.AuthError`	The number of publish requests the message broker was unable to authorize. The `Protocol` dimension contains the protocol used to publish the message.
`PublishIn.ClientError`	The number of publish requests rejected by the message broker because the message did not meet the requirements defined in AWS IoT quotas (p. 1006). The `Protocol` dimension contains the protocol used to publish the message.
`PublishIn.ServerError`	The number of publish requests the message broker failed to process because an internal error occurred. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`PublishIn.Success`	The number of publish requests successfully processed by the message broker. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.

Metric	Description
`PublishIn.Throttle`	The number of publish request that were throttled because the client exceeded the allowed inbound message rate. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`PublishOut.AuthError`	The number of publish requests made by the message broker that could not be authorized by AWS IoT. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`PublishOut.ClientError`	The number of publish requests made by the message broker that were rejected because the message did not meet the requirements defined in AWS IoT quotas (p. 1006). The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`PublishOut.Success`	The number of publish requests successfully made by the message broker. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`PublishOut.Throttle`	The number of publish requests that were throttled because the client exceeded the allowed outbound message rate. The `Protocol` dimension contains the protocol used to send the `PUBLISH` message.
`Subscribe.AuthError`	The number of subscription requests made by a client that could not be authorized. The `Protocol` dimension contains the protocol used to send the `SUBSCRIBE` message.
`Subscribe.ClientError`	The number of subscribe requests that were rejected because the `SUBSCRIBE` message did not meet the requirements defined in AWS IoT quotas (p. 1006). The `Protocol` dimension contains the protocol used to send the `SUBSCRIBE` message.
`Subscribe.ServerError`	The number of subscribe requests that were rejected because an internal error occurred. The `Protocol` dimension contains the protocol used to send the `SUBSCRIBE` message.
`Subscribe.Success`	The number of subscribe requests that were successfully processed by the message broker. The `Protocol` dimension contains the protocol used to send the `SUBSCRIBE` message.
`Subscribe.Throttle`	The number of subscribe requests that were throttled because the client exceeded the allowed subscribe request rate. The `Protocol` dimension contains the protocol used to send the `SUBSCRIBE` message.
`Unsubscribe.ClientError`	The number of unsubscribe requests that were rejected because the `UNSUBSCRIBE` message did not meet the requirements defined in AWS IoT quotas (p. 1006). The `Protocol` dimension contains the protocol used to send the `UNSUBSCRIBE` message.

Metric	Description
`Unsubscribe.ServerError`	The number of unsubscribe requests that were rejected because an internal error occurred. The `Protocol` dimension contains the protocol used to send the `UNSUBSCRIBE` message.
`Unsubscribe.Success`	The number of unsubscribe requests that were successfully processed by the message broker. The `Protocol` dimension contains the protocol used to send the `UNSUBSCRIBE` message.
`Unsubscribe.Throttle`	The number of unsubscribe requests that were rejected because the client exceeded the allowed unsubscribe request rate. The `Protocol` dimension contains the protocol used to send the `UNSUBSCRIBE` message.

Note
The message broker metrics are displayed in the AWS IoT console under **Protocol Metrics**.

Device shadow metrics

Metric	Description
`DeleteThingShadow.Accepted`	The number of `DeleteThingShadow` requests processed successfully. The `Protocol` dimension contains the protocol used to make the request.
`GetThingShadow.Accepted`	The number of `GetThingShadow` requests processed successfully. The `Protocol` dimension contains the protocol used to make the request.
`UpdateThingShadow.Accepted`	The number of `UpdateThingShadow` requests processed successfully. The `Protocol` dimension contains the protocol used to make the request.

Note
The device shadow metrics are displayed in the AWS IoT console under **Protocol Metrics**.

Jobs metrics

Metric	Description
`CanceledJobExecutionCount`	The number of job executions whose status has changed to `CANCELED` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`CanceledJobExecutionTotalCount`	The total number of job executions whose status is `CANCELED` for the given job. The `JobId` dimension contains the ID of the job.

Metric	Description
`ClientError`	The number of client errors generated while executing the job. The `JobId` dimension contains the ID of the job.
`FailedJobExecutionCount`	The number of job executions whose status has changed to `FAILED` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`FailedJobExecutionTotalCount`	The total number of job executions whose status is `FAILED` for the given job. The `JobId` dimension contains the ID of the job.
`InProgressJobExecutionCount`	The number of job executions whose status has changed to `IN_PROGRESS` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`InProgressJobExecutionTotalCount`	The total number of job executions whose status is `IN_PROGRESS` for the given job. The `JobId` dimension contains the ID of the job.
`RejectedJobExecutionTotalCount`	The total number of job executions whose status is `REJECTED` for the given job. The `JobId` dimension contains the ID of the job.
`RemovedJobExecutionTotalCount`	The total number of job executions whose status is `REMOVED` for the given job. The `JobId` dimension contains the ID of the job.
`QueuedJobExecutionCount`	The number of job executions whose status has changed to `QUEUED` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`QueuedJobExecutionTotalCount`	The total number of job executions whose status is `QUEUED` for the given job. The `JobId` dimension contains the ID of the job.
`RejectedJobExecutionCount`	The number of job executions whose status has changed to `REJECTED` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.

Metric	Description
`RemovedJobExecutionCount`	The number of job executions whose status has changed to `REMOVED` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`ServerError`	The number of server errors generated while executing the job. The `JobId` dimension contains the ID of the job.
`SuccededJobExecutionCount`	The number of job executions whose status has changed to `SUCCESS` within a time period that is determined by CloudWatch. (For more information about CloudWatch metrics, see Amazon CloudWatch Metrics.) The `JobId` dimension contains the ID of the job.
`SuccededJobExecutionTotalCount`	The total number of job executions whose status is `SUCCESS` for the given job. The `JobId` dimension contains the ID of the job.

Device Defender audit metrics

Metric	Description
`NonCompliantResources`	The number of resources that were found to be noncompliant with a check. The system reports the number of resources that were out of compliance for each check of each audit performed.
`ResourcesEvaluated`	The number of resources that were evaluated for compliance. The system reports the number of resources that were evaluated for each check of each audit performed.

Device Defender detect metrics

Metric	Description
`Violations`	The number of new violations of security profile behaviors that have been found since the last time an evaluation was performed. The system reports the number of new violations for the account, for a specific security profile, and for a specific behavior of a specific security profile.
`ViolationsCleared`	The number of violations of security profile behaviors that have been resolved since the last time an evaluation was performed. The system reports the number of resolved violations for the account, for a

Metric	Description
	specific security profile, and for a specific behavior of a specific security profile.
`ViolationsInvalidated`	The number of violations of security profile behaviors for which information is no longer available since the last time an evaluation was performed (because the reporting device stopped reporting, or is no longer being monitored for some reason). The system reports the number of invalidated violations for the entire account, for a specific security profile, and for a specific behavior of a specific security profile.

Device provisioning metrics

AWS IoT Fleet provisioning metrics

Metric	Description
`ApproximateNumberOfThingsRegistered`	The count of devices that have been registered by Fleet Provisioning. While the count is generally accurate, the distributed architecture of AWS IoT Core makes it difficult to maintain a precise count of registered things.
`CreateKeysAndCertificateFailed`	The number of failures that occurred calling the `CreateKeysAndCertificate` MQTT API.
`RegisterThingFailed`	The number of failures that occurred when calling the MQTT `RegisterThing` API.

Just-in-time provisioning metrics

Metric	Description
`ProvisionThing.ClientError`	The number of times a device failed to provision due to a client error. For example, the policy specified in the template did not exist.
`ProvisionThing.ServerError`	The number of times a device failed to provision due to a server error. Customers can retry to provision the device after waiting and they can contact AWS IoT if the issue remains the same.
`ProvisionThing.Success`	The number of times a device was successfully provisioned.

Dimensions for metrics

Metrics use the namespace and provide metrics for the following dimensions

Dimension	Description
`ActionType`	The action type (p. 359) specified by the rule that triggered the request.
`BehaviorName`	The name of the Device Defender Detect security profile behavior that is being monitored.
`ClaimCertificateId`	The `certificateId` of the claim used to provision the devices.
`CheckName`	The name of the Device Defender audit check whose results are being monitored.
`JobId`	The ID of the job whose progress or message connection success/failure is being monitored.
`Protocol`	The protocol used to make the request. Valid values are: MQTT or HTTP
`RuleName`	The name of the rule triggered by the request.
`ScheduledAuditName`	The name of the Device Defender scheduled audit whose check results are being monitored. This has the value `OnDemand` if the results reported are for an audit that was performed on demand.
`SecurityProfileName`	The name of the Device Defender Detect security profile whose behaviors are being monitored.
`TemplateName`	The name of the provisioning template.

Monitor AWS IoT using CloudWatch Logs

When AWS IoT logging is enabled (p. 310), AWS IoT sends progress events about each message as it passes from your devices through the message broker and rules engine. In the CloudWatch console, CloudWatch logs appear in a log group named **AWSIotLogs**.

For more information about CloudWatch Logs, see CloudWatch Logs. For information about supported AWS IoT CloudWatch Logs, see CloudWatch AWS IoT log entries (p. 331).

Viewing AWS IoT logs in the CloudWatch console

Note
The `AWSIotLogsV2` log group is not visible in the CloudWatch console until:

- You've enabled logging in AWS IoT. For more info on how to enable logging in AWS IoT, see Configure AWS IoT logging (p. 310)
- Some log entries have been written by AWS IoT operations.

To view your AWS IoT logs in the CloudWatch console

1. Browse to https://console.aws.amazon.com/cloudwatch/. In the navigation pane, choose **Log groups**.
2. In the **Filter** text box, enter **`AWSIotLogsV2`** , and then press Enter.
3. Double-click the `AWSIotLogsV2` log group.
4. Choose **Search All**. A complete list of the AWS IoT logs generated for your account is displayed.
5. Choose the expand icon to look at an individual stream.

You can also enter a query in the **Filter events** text box. Here are some interesting queries to try:

- `{ $.logLevel = "INFO" }`

  Find all logs that have a log level of `INFO`.
- `{ $.status = "Success" }`

  Find all logs that have a status of `Success`.
- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  Find all logs that have a status of `Success` and an event type of `GetThingShadow`.

For more information about creating filter expressions, see CloudWatch Logs Queries.

CloudWatch AWS IoT log entries

Each component of AWS IoT generates its own log entries. Each log entry has an `eventType` that specifies the operation that caused the log entry to be generated. This section describes the log entries generated by the following AWS IoT components:

Topics

Message broker log entries

The AWS IoT message broker generates log entries for the following events:

Topics

Connect log entry

The AWS IoT message broker generates a log entry with an `eventType` of `Connect` when an MQTT client connects.

Connect log entry example

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `Connect` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

sourceIp

The IP address where the request originated.

sourcePort

The port where the request originated.

Disconnect log entry

The AWS IoT message broker generates a log entry with an `eventType` of `Disconnect` when an MQTT client disconnects.

Disconnect log entry example

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `Disconnect` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

sourceIp

The IP address where the request originated.

sourcePort

The port where the request originated.

Publish-In log entry

When the AWS IoT message broker receives an MQTT message, it generates a log entry with an `eventType` of `Publish-In`.

Publish-In log entry example

```
{
        "timestamp": "2017-08-10 15:39:30.961",
        "logLevel": "INFO",
        "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
        "accountId": "123456789012",
        "status": "Success",
        "eventType": "Publish-In",
        "protocol": "MQTT",
        "topicName": "$aws/things/MyThing/shadow/get",
        "clientId": "abf27092886e49a8a5c1922749736453",
        "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
        "sourceIp": "205.251.233.181",
        "sourcePort": 13490
    }
```

In addition to the Common CloudWatch Logs attributes (p. 349), `Publish-In` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

sourceIp

The IP address where the request originated.

sourcePort

The port where the request originated.

topicName

The name of the subscribed topic.

Publish-Out log entry

When the message broker publishes an MQTT message, it generates a log entry with an `eventType` of `Publish-Out`

Publish-Out log entry example

```
{
    "timestamp": "2017-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/get",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `Publish-Out` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

sourceIp

The IP address where the request originated.

sourcePort

The port where the request originated.

topicName

The name of the subscribed topic.

Subscribe log entry

The AWS IoT message broker generates a log entry with an `eventType` of `Subscribe` when an MQTT client subscribes to a topic.

Subscribe log entry example

```
{
    "timestamp": "2017-08-10 15:39:04.413",
    "logLevel": "INFO",
    "traceId": "7aa5c38d-1b49-3753-15dc-513ce4ab9fa6",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/#",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `Subscribe` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

sourceIp

The IP address where the request originated.

sourcePort

The port where the request originated.

topicName

The name of the subscribed topic.

Device Shadow log entries

The AWS IoT Device Shadow service generates log entries for the following events:

Topics

DeleteThingShadow log entry

The Device Shadow service generates a log entry with an `eventType` of `DeleteThingShadow` when a request to delete a device's shadow is received.

DeleteThingShadow log entry example

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/delete"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `DeleteThingShadow` log entries contain the following attributes:

deviceShadowName

    The name of the shadow to update.

protocol

    The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

    The name of the topic on which the request was published.

GetThingShadow log entry

The Device Shadow service generates a log entry with an `eventType` of `GetThingShadow` when a get request for a shadow is received.

GetThingShadow log entry example

```
{
    "timestamp": "2017-08-09 17:56:30.941",
    "logLevel": "INFO",
    "traceId": "b575f19a-97a2-cf72-0ed0-c64a783a2504",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "MyThing",
    "topicName": "$aws/things/MyThing/shadow/get"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `GetThingShadow` log entries contain the following attributes:

deviceShadowName

    The name of the requested shadow.

protocol

    The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

    The name of the topic on which the request was published.

UpdateThingShadow log entry

The Device Shadow service generates a log entry with an `eventType` of `UpdateThingShadow` when a request to update a device's shadow is received.

UpdateThingShadow log entry example

```
{
    "timestamp": "2017-08-07 18:43:59.436",
    "logLevel": "INFO",
    "traceId": "d0074ba8-0c4b-a400-69df-76326d414c28",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/update"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `UpdateThingShadow` log entries contain the following attributes:

deviceShadowName

The name of the shadow to update.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

The name of the topic on which the request was published.

AWS IoT Core for LoRaWAN log entries

AWS IoT Core for LoRaWAN actions generate log entries for the following events.

Topics

Join error log entry

AWS IoT Core for LoRaWAN generates a log entry with an `event` value of `Join` when a message fails its Message Integrity Code (MIC) check.

Join error log entry example

```
{
    "timestamp": "2020-11-24T01:46:50.883481989Z",
    "resource": "WirelessDevice",
    "resourceId": "cb4c087c-1be5-4990-8654-ccf543ee9fff",
    "devEui": "58a0cb000020255c",
    "event": "Join",
    "logLevel": "ERROR",
    "message": "invalid MIC. It's most likely caused by wrong root keys."
}
```

Uplink permission error log entry

AWS IoT Core for LoRaWAN generates a log entry with an `event` value of `Uplink` when it encounters a permission error while processing a message.

Uplink permission error log entry example

```
{
    "resource": "WirelessDevice",
    "resourceId": "cb4c087c-1be5-4990-8654-ccf543ee9fff",
    "event": "Uplink",
    "logLevel": "ERROR",
    "message": "Cannot assume role MessageId: ef38877f-3454-4c99-96ed-5088c1cd8dee.
 WirelessDeviceId: cb4c087c-1be5-4990-8654-ccf543ee9fff. Context: LorawanDataPlaneUplink.
 reason: AccessDenied: User: arn:aws:sts::005196538709:assumed-role/
DataRoutingServiceRole/6368b35fd48c445c9a14781b5d5890ed is not authorized to perform:
 sts:AssumeRole on resource: arn:aws:iam::400232685877:role/ExecuteRules_Role\tstatus code:
 403, request id: 471c3e35-f8f3-4e94-b734-c862f63f4edb" }
```

Downlink error log entry

AWS IoT Core for LoRaWAN generates a log entry with an `event` value of `Downlink` when a device session is not found in an attempt to send data to a device.

Downlink error log entry example

```
{
    "resource": "WirelessGateway",
    "resourceId": "af5bada3-9ded-3416-9e2f-0d5fe052aeb8",
    "event": "Downlink",
    "logLevel": "ERROR",
    "message": "Downlink returns an error when delivering application
 message. MessageId: 5fb8754d-2640-97975399914acd58-0007. WirelessDeviceId:
 af5bada3-9ded-3416-9e2f-0d5fe052aeb8. Context: LorawanDataPlaneDownlink.  reason:
 {\"Message\":\"device session not found. deviceId=af5bada3-9ded-3416-9e2f-0d5fe052aeb8\"}"
}
```

Rules engine log entries

The AWS IoT rules engine generates logs for the following events:

Topics

FunctionExecution log entry

The rules engine generates a log entry with an `eventType` of `FunctionExecution` when a rule's SQL query calls an external function. An external function is called when a rule's action makes an HTTP request to AWS IoT or another web service (for example, calling `get_thing_shadow` or `machinelearning_predict`).

FunctionExecution log entry example

```
{
    "timestamp": "2017-07-13 18:33:51.903",
    "logLevel": "DEBUG",
    "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
    "status": "Success",
    "eventType": "FunctionExecution",
    "clientId": "N/A",
    "topicName":"rules/test",
    "ruleName": "ruleTestPredict",
    "ruleAction": "MachinelearningPredict",
    "resources": {
        "ModelId": "predict-model"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `FunctionExecution` log entries contain the following attributes:

clientId

: `N/A` for `FunctionExecution` logs.

principalId

: The ID of the principal making the request.

resources

: A collection of resources used by the rule's actions.

ruleName

: The name of the matching rule.

topicName

: The name of the subscribed topic.

RuleExecution log entry

When the AWS IoT rules engine triggers a rule's action, it generates a `RuleExecution` log entry.

RuleExecution log entry example

```
{
    "timestamp": "2017-08-10 16:32:46.070",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "resources": {
        "RepublishTopic": "rules/republish"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `RuleExecution` log entries contain the following attributes:

clientId

: The ID of the client making the request.

principalId

The ID of the principal making the request.

resources

A collection of resources used by the rule's actions.

ruleAction

The name of the action triggered.

ruleName

The name of the matching rule.

topicName

The name of the subscribed topic.

RuleMatch log entry

The AWS IoT rules engine generates a log entry with an `eventType` of `RuleMatch` when the message broker receives a message that matches a rule.

RuleMatch log entry example

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleMatch",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `RuleMatch` log entries contain the following attributes:

clientId

The ID of the client making the request.

principalId

The ID of the principal making the request.

ruleName

The name of the matching rule.

topicName

The name of the subscribed topic.

RuleMessageThrottled log entry

When a message is throttled, the AWS IoT rules engine generates a log entry with an `eventType` of `RuleMessageThrottled`.

RuleMessageThrottled log entry example

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleMessageThrottled",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleExecutionThrottled",
    "details": "Message for Rule example_rule throttled"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `RuleMessageThrottled` log entries contain the following attributes:

clientId

The ID of the client making the request.

details

A brief explanation of the error.

principalId

The ID of the principal making the request.

reason

The string "RuleMessageThrottled".

ruleName

The name of the rule to be triggered.

topicName

The name of the topic that was published.

RuleNotFound log entry

When the AWS IoT rules engine cannot find a rule with a given name, it generates a log entry with an `eventType` of `RuleNotFound`.

RuleNotFound log entry example

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleNotFound",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleNotFound",
    "details": "Rule example_rule not found"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `RuleNotFound` log entries contain the following attributes:

clientId

    The ID of the client making the request.

details

    A brief explanation of the error.

principalId

    The ID of the principal making the request.

reason

    The string "RuleNotFound".

ruleName

    The name of the rule that could not be found.

topicName

    The name of the topic that was published.

StartingRuleExecution log entry

When the AWS IoT rules engine starts to trigger a rule's action, it generates a log entry with an `eventType` of `StartingRuleExecution`.

StartingRuleExecution log entry example

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "DEBUG",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartingRuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `rule-` log entries contain the following attributes:

clientId

    The ID of the client making the request.

principalId

    The ID of the principal making the request.

ruleAction

    The name of the action triggered.

ruleName

The name of the matching rule.

topicName

The name of the subscribed topic.

Job log entries

The AWS IoT Job service generates log entries for the following events. Log entries are generated when an MQTT or HTTP request is received from the device.

Topics

DescribeJobExecution log entry

The AWS IoT Jobs service generates a log entry with an `eventType` of `DescribeJobExecution` when the service receives a request to describe a job execution.

DescribeJobExecution log entry example

```
{
    "timestamp": "2017-08-10 19:13:22.841",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DescribeJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/get",
    "clientToken": "myToken",
    "details": "The request status is SUCCESS."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `GetJobExecution` log entries contain the following attributes:

clientId

The ID of the client making the request.

clientToken

A unique, case-sensitive identifier to ensure the idempotency of the request. For more information, see How to Ensure Idempotency.

details

Other information from the Jobs service.

jobId

The job ID for the job execution.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

The topic used to make the request.

GetPendingJobExecution log entry

The AWS IoT Jobs service generates a log entry with an `eventType` of `GetPendingJobExecution` when the service receives a job execution request.

GetPendingJobExecution log entry example

```
{
    "timestamp": "2018-06-13 17:45:17.197",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "299966ad-54de-40b4-99d3-4fc8b52da0c5",
    "topicName": "$aws/things/299966ad-54de-40b4-99d3-4fc8b52da0c5/jobs/get",
    "clientToken": "24b9a741-15a7-44fc-bd3c-1ff2e34e5e82",
    "details": "The request status is SUCCESS."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `GetPendingJobExecution` log entries contain the following attributes:

clientId

The ID of the client making the request.

clientToken

A unique, case sensitive identifier to ensure the idempotency of the request. For more information, see How to Ensure Idempotency.

details

Other information from the Jobs service.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

The name of the subscribed topic.

ReportFinalJobExecutionCount log entry

The AWS IoT Jobs service generates a log entry with an `entryType` of `ReportFinalJobExecutionCount` when a job is completed.

ReportFinalJobExecutionCount log entry example

```
{
    "timestamp": "2017-08-10 19:44:16.776",
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ReportFinalJobExecutionCount",
    "jobId": "002",
    "details": "Job 002 completed. QUEUED job execution count: 0 IN_PROGRESS job execution
 count: 0 FAILED job execution count: 0 SUCCEEDED job execution count: 1 CANCELED job
 execution count: 0 REJECTED job execution count: 0 REMOVED job execution count: 0"
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `ReportFinalJobExecutionCount` log entries contain the following attributes:

details

    Other information from the Jobs service.

jobId

    The job ID for the job execution.

StartNextPendingJobExecution log entry

When it receives a request to start the next pending job execution, the AWS IoT Jobs service generates a log entry with an `eventType` of `StartNextPendingJobExecution`.

StartNextPendingJobExecution log entry example

```
{
    "timestamp": "2018-06-13 17:49:51.036",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartNextPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "95c47808-b1ca-4794-bc68-a588d6d9216c",
    "topicName": "$aws/things/95c47808-b1ca-4794-bc68-a588d6d9216c/jobs/start-next",
    "clientToken": "bd7447c4-3a05-49f4-8517-dd89b2c68d94",
    "details": "The request status is SUCCESS."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `StartNextPendingJobExecution` log entries contain the following attributes:

clientId

    The ID of the client making the request.

clientToken

    A unique, case sensitive identifier to ensure the idempotency of the request. For more information, see How to Ensure Idempotency.

details

    Other information from the Jobs service.

protocol

    The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

    The topic used to make the request.

UpdateJobExecution log entry

The AWS IoT Jobs service generates a log entry with an `eventType` of `UpdateJobExecution` when the service receives a request to update a job execution.

UpdateJobExecution log entry example

```
{
    "timestamp": "2017-08-10 19:25:14.758",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/update",
    "clientToken": "myClientToken",
    "versionNumber": "1",
    "details": "The destination status is IN_PROGRESS. The request status is SUCCESS."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `UpdateJobExecution` log entries contain the following attributes:

clientId

The ID of the client making the request.

clientToken

A unique, case sensitive identifier to ensure the idempotency of the request. For more information, see How to Ensure Idempotency.

details

Other information from the Jobs service.

jobId

The job ID for the job execution.

protocol

The protocol used when making the request. Valid values are `MQTT` or `HTTP`.

topicName

The topic used to make the request.

versionNumber

The version of the job execution.

Device provisioning log entries

The AWS IoT Device Provisioning service generates logs for the following events.

Topics

GetDeviceCredentials log entry

The AWS IoT Device Provisioning service generates a log entry with an `eventType` of `GetDeviceCredential` when a client calls `GetDeviceCredential`.

GetDeviceCredentials log entry example

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "GetDeviceCredentials",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `GetDeviceCredentials` log entries contain the following attributes:

details

    A brief explanation of the error.

deviceCertificateId

    The ID of the device certificate.

ProvisionDevice log entry

The AWS IoT Device Provisioning service generates a log entry with an `eventType` of `ProvisionDevice` when a client calls `ProvisionDevice`.

ProvisionDevice log entry example

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "ProvisionDevice",
  "provisioningTemplateName" : "myTemplate",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
 }
```

In addition to the Common CloudWatch Logs attributes (p. 349), `ProvisionDevice` log entries contain the following attributes:

details

    A brief explanation of the error.

deviceCertificateId

    The ID of the device certificate.

provisioningTemplateName

The name of the provisioning template.

Dynamic thing group log entries

AWS IoT Dynamic Thing Groups generate logs for the following event.

Topics

AddThingToDynamicThingGroupsFailed log entry

When AWS IoT was not able to add a thing to the specified dynamic groups, it generates a log entry with an `eventType` of `AddThingToDynamicThingGroupsFailed`. This happens when a thing met the criteria to be in the dynamic thing group; however, it could not be added to the dynamic group or it was removed from the dynamic group. This can happen because:

- The thing already belongs to the maximum number of groups.
- The **--override-dynamic-groups** option was used to add the thing to a static thing group. It was removed from a dynamic thing group to make that possible.

For more information, see Dynamic Thing Group Limitations and Conflicts (p. 191).

AddThingToDynamicThingGroupsFailed log entry example

This example shows the log entry of an `AddThingToDynamicThingGroupsFailed` error. In this example, *TestThing* met the criteria to be in the dynamic thing groups listed in `dynamicThingGroupNames`, but could not be added to those dynamic groups, as described in `reason`.

```
{
 "timestamp": "2020-03-16 22:24:43.804",
 "logLevel": "ERROR",
 "traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
 "accountId": "571032923833",
 "status": "Failure",
 "eventType": "AddThingToDynamicThingGroupsFailed",
 "thingName": "TestThing",
 "dynamicThingGroupNames": [
  "DynamicThingGroup11",
  "DynamicThingGroup12",
  "DynamicThingGroup13",
  "DynamicThingGroup14"
 ],
 "reason": "The thing failed to be added to the given dynamic thing group(s) because the
 thing already belongs to the maximum allowed number of groups."
}
```

In addition to the Common CloudWatch Logs attributes (p. 349), `AddThingToDynamicThingGroupsFailed` log entries contain the following attributes:

dynamicThingGroupNames

An array of the dynamic thing groups to which the thing could not be added.

reason

The reason why the thing could not be added to the dynamic thing groups.

thingName

The name of the thing that could not be added to a dynamic thing group.

Common CloudWatch Logs attributes

All CloudWatch Logs log entries include these attributes:

accountId

Your AWS account ID.

eventType

The event type for which the log was generated. The value of the event type depends on the event that generated the log entry. Each log entry description includes the value of `eventType` for that log entry.

logLevel

The log level being used. For more information, see the section called “Log levels” (p. 317).

status

The status of the request.

timestamp

The UNIX timestamp of when the client connected to the AWS IoT message broker.

traceId

A randomly generated identifier that can be used to correlate all logs for a specific request.

Log AWS IoT API calls using AWS CloudTrail

AWS IoT is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS IoT. CloudTrail captures all API calls for AWS IoT as events, including calls from the AWS IoT console and from code calls to the AWS IoT APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS IoT. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT, the IP address from which the request was made, who made the request, when it was made, and other details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS IoT information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS IoT, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS IoT, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. You can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Note
AWS IoT data plane actions (device side) are not logged by CloudTrail. Use CloudWatch to monitor these actions.

Generally speaking, AWS IoT control plane actions that make changes are logged by CloudTrail. Calls such as **CreateThing**, **CreateKeysAndCertificate**, and **UpdateCertificate** leave CloudTrail entries, while calls such as **ListThings** and **ListTopicRules** do not.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

AWS IoT actions are documented in the AWS IoT API Reference. AWS IoT Wireless actions are documented in the AWS IoT Wireless API Reference.

Understanding AWS IoT log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `AttachPolicy` action.

```
{
    "timestamp":"1460159496",
    "AdditionalEventData":"",
    "Annotation":"",
    "ApiVersion":"",
    "ErrorCode":"",
    "ErrorMessage":"",
    "EventID":"8bff4fed-c229-4d2d-8264-4ab28a487505",
    "EventName":"AttachPolicy",
    "EventTime":"2016-04-08T23:51:36Z",
    "EventType":"AwsApiCall",
    "ReadOnly":"",
    "RecipientAccountList":"",
    "RequestID":"d4875df2-fde4-11e5-b829-23bf9b56cbcd",
    "RequestParamters":{
        "principal":"arn:aws:iot:us-
east-1:123456789012:cert/528ce36e8047f6a75ee51ab7beddb4eb268ad41d2ea881a10b67e8e76924d894",
        "policyName":"ExamplePolicyForIoT"
    },
    "Resources":"",
    "ResponseElements":"",
    "SourceIpAddress":"52.90.213.26",
    "UserAgent":"aws-internal/3",
    "UserIdentity":{
        "type":"AssumedRole",
        "principalId":"AKIAI44QH8DHBEXAMPLE",
        "arn":"arn:aws:sts::12345678912:assumed-role/iotmonitor-us-east-1-beta-
InstanceRole-1C5T1YCYMHPYT/i-35d0a4b6",
        "accountId":"222222222222",
        "accessKeyId":"access-key-id",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"Fri Apr 08 23:51:10 UTC 2016"
            },
            "sessionIssuer":{
                "type":"Role",
                "principalId":"AKIAI44QH8DHBEXAMPLE",
                "arn":"arn:aws:iam::123456789012:role/executionServiceEC2Role/iotmonitor-
us-east-1-beta-InstanceRole-1C5T1YCYMHPYT",
                "accountId":"222222222222",
                "userName":"iotmonitor-us-east-1-InstanceRole-1C5T1YCYMHPYT"
            }
        },
        "invokedBy":{
            "serviceAccountId":"111111111111"
        }
    },
    "VpcEndpointId":""
}
```

Rules for AWS IoT

Rules give your devices the ability to interact with AWS services. Rules are analyzed and actions are performed based on the MQTT topic stream. You can use rules to support tasks like these:

- Augment or filter data received from a device.
- Write data received from a device to an Amazon DynamoDB database.
- Save a file to Amazon S3.
- Send a push notification to all users using Amazon SNS.
- Publish data to an Amazon SQS queue.
- Invoke a Lambda function to extract data.
- Process messages from a large number of devices using Amazon Kinesis.
- Send data to the Amazon Elasticsearch Service.
- Capture a CloudWatch metric.
- Change a CloudWatch alarm.
- Send the data from an MQTT message to Amazon Machine Learning to make predictions based on an Amazon ML model.
- Send a message to a Salesforce IoT Input Stream.
- Send message data to an AWS IoT Analytics channel.
- Start execution of a Step Functions state machine.
- Send message data to an AWS IoT Events input.
- Send message data an asset property in AWS IoT SiteWise.
- Send message data to a web application or service.

Your rules can use MQTT messages that pass through the publish/subscribe protocol supported by the the section called “Device communication protocols” (p. 76) or, using the Basic Ingest (p. 429) feature, you can securely send device data to the AWS services listed above without incurring messaging costs. (The Basic Ingest (p. 429) feature optimizes data flow by removing the publish/subscribe message broker from the ingestion path, so it is more cost effective while keeping the security and data processing features of AWS IoT.)

Before AWS IoT can perform these actions, you must grant it permission to access your AWS resources on your behalf. When the actions are performed, you incur the standard charges for the AWS services you use.

Contents

- Working with topic rule destinations (p. 403)
- Reducing messaging costs with basic ingest (p. 429)
- AWS IoT SQL reference (p. 430)

Granting AWS IoT the required access

You use IAM roles to control the AWS resources to which each rule has access. Before you create a rule, you must create an IAM role with a policy that allows access to the required AWS resources. AWS IoT assumes this role when executing a rule.

To create an IAM role (AWS CLI)

1. Save the following trust policy document, which grants AWS IoT permission to assume the role, to a file named `iot-role-trust.json`:

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

   Use the create-role command to create an IAM role specifying the `iot-role-trust.json` file:

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document file://iot-role-trust.json
```

   The output of this command looks like the following:

```
{
  "Role": {
    "AssumeRolePolicyDocument": "url-encoded-json",
    "RoleId": "AKIAIOSFODNN7EXAMPLE",
    "CreateDate": "2015-09-30T18:43:32.821Z",
    "RoleName": "my-iot-role",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:role/my-iot-role"
  }
}
```

2. Save the following JSON into a file named `my-iot-policy.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "*"
  }]
}
```

   This JSON is an example policy document that grants AWS IoT administrator access to DynamoDB.

Use the create-policy command to grant AWS IoT access to your AWS resources upon assuming the role, passing in the `my-iot-policy.json` file:

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-
policy.json
```

For more information about how to grant access to AWS services in policies for AWS IoT, see Creating an AWS IoT rule (p. 355).

The output of the create-policy command contains the ARN of the policy. You need to attach the policy to a role.

```
{
  "Policy": {
    "PolicyName": "my-iot-policy",
    "CreateDate": "2015-09-30T19:31:18.620Z",
    "AttachmentCount": 0,
    "IsAttachable": true,
    "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
    "DefaultVersionId": "v1",
    "Path": "/",
    "Arn": "arn:aws:iam::123456789012:policy/my-iot-policy",
    "UpdateDate": "2015-09-30T19:31:18.620Z"
  }
}
```

3. Use the attach-role-policy command to attach your policy to your role:

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

Pass role permissions

Part of a rule definition is an IAM role that grants permission to access resources specified in the rule's action. The rules engine assumes that role when the rule's action is triggered. The role must be defined in the same AWS account as the rule.

When creating or replacing a rule you are, in effect, passing a role to the rules engine. The user performing this operation requires the `iam:PassRole` permission. To ensure you have this permission, create a policy that grants the `iam:PassRole` permission and attach it to your IAM user. The following policy shows how to allow `iam:PassRole` permission for a role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/myRole"
      ]
    }
  ]
}
```

In this policy example, the `iam:PassRole` permission is granted for the role `myRole`. The role is specified using the role's ARN. You must attach this policy to your IAM user or role to which your user belongs. For more information, see Working with Managed Policies.

> **Note**
> Lambda functions use resource-based policy, where the policy is attached directly to the Lambda function itself. When you create a rule that invokes a Lambda function, you do not pass a role, so the user creating the rule does not need the `iam:PassRole` permission. For more information about Lambda function authorization, see Granting Permissions Using a Resource Policy.

Creating an AWS IoT rule

You configure rules to route data from your connected things. Rules consist of the following:

Rule name

The name of the rule.

> **Note**
> We do not recommend the use of personally identifiable information in your rule names.

Optional description

A textual description of the rule.

> **Note**
> We do not recommend the use of personally identifiable information in your rule descriptions.

SQL statement

A simplified SQL syntax to filter messages received on an MQTT topic and push the data elsewhere. For more information, see AWS IoT SQL reference (p. 430).

SQL version

The version of the SQL rules engine to use when evaluating the rule. Although this property is optional, we strongly recommend that you specify the SQL version. If this property is not set, the default, `2015-10-08`, is used. For more information, see SQL versions (p. 489).

One or more actions

The actions AWS IoT performs when executing the rule. For example, you can insert data into a DynamoDB table, write data to an Amazon S3 bucket, publish to an Amazon SNS topic, or invoke a Lambda function.

An error action

The action AWS IoT performs when it is unable to perform a rule's action.

When you create a rule, be aware of how much data you are publishing on topics. If you create rules that include a wildcard topic pattern, they might match a large percentage of your messages, and you might need to increase the capacity of the AWS resources used by the target actions. Also, if you create a republish rule that includes a wildcard topic pattern, you can end up with a circular rule that causes an infinite loop.

> **Note**
> Creating and updating rules are administrator-level actions. Any user who has permission to create or update rules is able to access data processed by the rules.

To create a rule (AWS CLI)

Use the create-topic-rule command to create a rule:

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://my-rule.json
```

The following is an example payload file with a rule that inserts all messages sent to the `iot/test` topic into the specified DynamoDB table. The SQL statement filters the messages and the role ARN grants AWS IoT permission to write to the DynamoDB table.

```
{
  "sql": "SELECT * FROM 'iot/test'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "dynamoDB": {
          "tableName": "my-dynamodb-table",
          "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
          "hashKeyField": "topic",
          "hashKeyValue": "${topic(2)}",
          "rangeKeyField": "timestamp",
          "rangeKeyValue": "${timestamp()}"
      }
  }]
}
```

The following is an example payload file with a rule that inserts all messages sent to the `iot/test` topic into the specified S3 bucket. The SQL statement filters the messages, and the role ARN grants AWS IoT permission to write to the Amazon S3 bucket.

```
{
  "awsIotSqlVersion": "2016-03-23",
  "sql": "SELECT * FROM 'iot/test'",
  "ruleDisabled": false,
  "actions": [
    {
      "s3": {
        "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
        "bucketName": "my-bucket",
        "key": "myS3Key"
      }
    }
  ]
}
```

The following is an example payload file with a rule that pushes data to Amazon Elasticsearch Service:

```
{
  "sql":"SELECT *, timestamp() as timestamp FROM 'iot/test'",
  "ruleDisabled":false,
  "awsIotSqlVersion": "2016-03-23",
  "actions":[
    {
      "elasticsearch":{
        "roleArn":"arn:aws:iam::123456789012:role/aws_iot_es",
        "endpoint":"https://my-endpoint",
        "index":"my-index",
        "type":"my-type",
        "id":"${newuuid()}"
      }
    }
  ]
}
```

The following is an example payload file with a rule that invokes a Lambda function:

```
{
  "sql": "expression",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "lambda": {
          "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function"
      }
  }]
}
```

The following is an example payload file with a rule that publishes to an Amazon SNS topic:

```
{
  "sql": "expression",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "sns": {
          "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
          "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
      }
  }]
}
```

The following is an example payload file with a rule that republishes on a different MQTT topic:

```
{
  "sql": "expression",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "republish": {
          "topic": "my-mqtt-topic",
          "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
      }
  }]
}
```

The following is an example payload file with a rule that pushes data to an Amazon Kinesis Data Firehose stream:

```
{
  "sql": "SELECT * FROM 'my-topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "firehose": {
          "roleArn": ""arn:aws:iam::123456789012:role/my-iot-role",
          "deliveryStreamName": "my-stream-name"
      }
  }]
}
```

The following is an example payload file with a rule that uses the Amazon Machine Learning `machinelearning_predict` function to republish to a topic if the data in the MQTT payload is classified as a 1.

```
{
  "sql": "SELECT * FROM 'iot/test' where machinelearning_predict('my-model',
 'arn:aws:iam::123456789012:role/my-iot-aml-role', *).predictedLabel=1",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
    "republish": {
        "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
        "topic": "my-mqtt-topic"
    }
  }]
}
```

The following is an example payload file with a rule that publishes messages to a Salesforce IoT Cloud input stream.

```
{
  "sql": "expression",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "salesforce": {
          "token": "ABCDEFGHI123456789abcdefghi123456789",
          "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/stream-id/
connection-id/my-event"
      }
  }]
}
```

The following is an example payload file with a rule that starts an execution of a Step Functions state machine.

```
{
  "sql": "expression",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "stepFunctions": {
          "stateMachineName": "myCoolStateMachine",
          "executionNamePrefix": "coolRunning",
          "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
      }
  }]
}
```

Viewing your rules

Use the list-topic-rules command to list your rules:

```
aws iot list-topic-rules
```

Use the get-topic-rule command to get information about a rule:

```
aws iot get-topic-rule --rule-name my-rule
```

Deleting a rule

When you are finished with a rule, you can delete it.

To delete a rule (AWS CLI)

Use the delete-topic-rule command to delete a rule:

```
aws iot delete-topic-rule --rule-name my-rule
```

AWS IoT rule actions

AWS IoT rule actions specify what to do when a rule is triggered. You can define actions to send data to an Amazon DynamoDB database, send data to Amazon Kinesis Data Streams, invoke an AWS Lambda function, and so on. AWS IoT supports the following actions in AWS Regions where the action's service is available.

Rule action	Description	Name in API
Apache Kafka (p. 360)	Sends a message to an Apache Kafka cluster.	`kafka`
CloudWatch alarms (p. 365)	Changes the state of an Amazon CloudWatch alarm.	`cloudwatchAlarm`
CloudWatch Logs (p. 366)	Sends a message to Amazon CloudWatch Logs.	`cloudwatchLogs`
CloudWatch metrics (p. 367)	Sends a message to a CloudWatch metric.	`cloudwatchMetric`
DynamoDB (p. 369)	Sends a message to a DynamoDB table.	`dynamoDB`
DynamoDBv2 (p. 371)	Sends message data to multiple columns in a DynamoDB table.	`dynamoDBv2`
Elasticsearch (p. 373)	Sends a message to an Amazon Elasticsearch Service endpoint.	`elasticsearch`
HTTPS (p. 374)	Posts a message to an HTTPS endpoint.	`http`
IoT Analytics (p. 377)	Sends a message to an AWS IoT Analytics channel.	`iotAnalytics`
IoT Events (p. 378)	Sends a message to an AWS IoT Events input.	`iotEvents`
IoT SiteWise (p. 380)	Sends message data to AWS IoT SiteWise asset properties.	`iotSiteWise`
Kinesis Data Firehose (p. 383)	Sends a message to a Kinesis Data Firehose delivery stream.	`firehose`

Rule action	Description	Name in API
Kinesis Data Streams (p. 385)	Sends a message to a Kinesis data stream.	`kinesis`
Lambda (p. 387)	Invokes a Lambda function with message data as input.	`lambda`
Republish (p. 389)	Republishes a message to another MQTT topic.	`republish`
S3 (p. 390)	Stores a message in an Amazon Simple Storage Service (Amazon S3) bucket.	`s3`
Salesforce IoT (p. 391)	Sends a message to a Salesforce IoT input stream.	`salesforce`
SNS (p. 392)	Publishes a message as an Amazon Simple Notification Service (Amazon SNS) push notification.	`sns`
SQS (p. 394)	Sends a message to an Amazon Simple Queue Service (Amazon SQS) queue.	`sqs`
Step Functions (p. 395)	Starts an AWS Step Functions state machine.	`stepFunctions`
the section called "Timestream" (p. 396)	Sends a message to an Amazon Timestream database table.	`timestream`
the section called "VPC" (p. 401)	Sends data to an Amazon Virtual Private Cloud (Amazon VPC).	`VPC`

Notes

- You must define the rule in the same AWS Region as another service's resource, so that the rule action can interact with that resource.
- The AWS IoT rules engine might make multiple attempts to perform an action in case of intermittent errors. If all attempts fail, the message is discarded and the error is available in your CloudWatch logs. You can specify an error action for each rule that is invoked after a failure occurs. For more information, see Error handling (error action) (p. 401).
- Some rule actions trigger actions in services that integrate with AWS Key Management Service (AWS KMS) to support data encryption at rest. If you use a customer-managed AWS KMS customer master key (CMK) to encrypt data at rest, the service must have permission to use the CMK on the caller's behalf. See the data encryption topics in the appropriate service guide to learn how to manage permissions for your customer-managed CMK. For more information about CMKs and customer-managed CMKs, see AWS Key Management Service concepts in the *AWS Key Management Service Developer Guide*.

Apache Kafka

The Apache Kafka (Kafka) action sends messages directly to your Amazon Managed Streaming for Apache Kafka (Amazon MSK) or self-managed Apache Kafka clusters for data analysis and visualization.

Note
This topic assumes familiarity with the Apache Kafka platform and related concepts. For more information about Apache Kafka, see Apache Kafka.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces`, `ec2:CreateNetworkInterfacePermission`, `ec2:DeleteNetworkInterface`, `ec2:DescribeSubnets`, `ec2:DescribeVpcs`, and `ec2:DescribeVpcAttribute` operations. This role creates and manages elastic network interfaces to your Amazon Virtual Private Cloud to reach your Kafka broker. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT Core to perform this rule action.

  For more information about network interfaces, see Elastic network interfaces in the Amazon EC2 User Guide.

  The policy attached to the role you specify should look like the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute"
            ],
            "Resource": "*"
        }
    ]
}
```

- If you use AWS Secrets Manager to store the credentials required to connect to your Kafka broker, you must create an IAM role that AWS IoT Core can assume to perform the `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` operations.

  The policy attached to the role you specify should look like the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "secretsmanager:DescribeSecret"
            ],
            "Resource": [
 "arn:aws:secretsmanager:region:123456789012:secret:kafka_client_truststore-*",
                "arn:aws:secretsmanager:region:123456789012:secret:kafka_keytab-*"
            ]
        }
    ]
}
```

- You must create a virtual private cloud (VPC) destination. (You can run your Apache Kafka clusters inside Amazon Virtual Private Cloud.) The AWS IoT rules engine creates an network interface in each of the subnets listed in the VPC destination. This allows the rules engine to route traffic directly to the VPC. When you create a VPC destination, the AWS IoT rules engine automatically creates a VPC rule action. For more information about VPC rule actions, see the section called "VPC" (p. 401).
- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt data at rest, the service must have permission to use the CMK on the caller's behalf. For more information, see Amazon MSK encryption in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

destinationArn

> The Amazon Resource Name (ARN) of the VPC destination. For information about creating a VPC destination, see the section called "VPC" (p. 401).

topic

> The Kafka topic for messages to be sent to the Kafka broker.

key (optional)

> The Kafka message key.

partition (optional)

> The Kafka message partition.

clientProperties

> An object that defines the properties of the Apache Kafka producer client.
>
> acks (optional)
>
> > The number of acknowledgments the producer requires the server to have received before considering a request complete.
> >
> > If you specify 0 as the value, the producer will not wait for any acknowledgment from the server. If the server doesn't receive the message, the producer won't retry to send the message.
> >
> > Valid values: 0, 1. The default value is 1.
>
> bootstrap.servers
>
> > A list of host and port pairs (`host1:port1`, `host2:port2`, etc.) used to establish the initial connection to your Kafka cluster.
>
> compression.type (optional)
>
> > The compression type for all data generated by the producer.
> >
> > Valid values: `none`, `gzip`, `snappy`, `lz4`, `zstd`. The default value is `none`.

security.protocol

The security protocol used to attach to your Kafka broker.

Valid values: `SSL`, `SASL_SSL`. The default value is `SSL`.

key.serializer

Specifies how to turn the key objects you provide with the`ProducerRecord` into bytes.

Valid value: `StringSerializer`.

value.serializer

Specifies how to turn value objects you provide with the `ProducerRecord` into bytes.

Valid value: `ByteBufferSerializer`.

ssl.truststore

The truststore file in base64 format or the location of the truststore file in AWS Secrets Manager. This value isn't required if your truststore is trusted by Amazon certificate authoriies (CA).

This field supports substitution templates. If you use Secrets Manager to store the credentials required to connect to your Kafka broker, you can use the get_secret SQL function to retrieve the value for this field. For more information about substitution templates, see the section called "Substitution templates" (p. 485). For more information about the get_secret SQL function, see the section called "get_secret(secretId, secretType, key, roleArn) " (p. 458). If the truststore is in the form of a file, use the `SecretBinary` parameter. If the truststore is in the form of a string, use the `SecretString` parameter.

ssl.truststore.password

The password for the truststore. This value is required only if you've created a password for the truststore.

ssl.keystore

The keystore file. This value is required when you specify `SSL` as the value for `security.protocol`.

This field supports substitution templates. You must use Secrets Manager to store the credentials required to connect to your Kafka broker. Use the get_secret SQL function to retrieve the value for this field. For more information about substitution templates, see the section called "Substitution templates" (p. 485). For more information about the get_secret SQL function, see the section called "get_secret(secretId, secretType, key, roleArn) " (p. 458). Use the `SecretBinary` parameter.

ssl.keystore.password

The store password for the keystore file. This value is required if you specify a value for `ssl.keystore`.

The value of this field can be plain text. This field also supports substitution templates. You must use Secrets Manager to store the credentials required to connect to your Kafka broker. Use the get_secret SQL function to retrieve the value for this field. For more information about substitution templates, see the section called "Substitution templates" (p. 485). For more information about the get_secret SQL function, see the section called "get_secret(secretId, secretType, key, roleArn) " (p. 458).Use the `SecretString` parameter.

ssl.key.password

The password of the private key in your keystore file.

This field supports substitution templates. You must use Secrets Manager to store the credentials required to connect to your Kafka broker. Use the get_secret SQL function to retrieve the value for this field. For more information about substitution templates, see the section called "Substitution templates" (p. 485). For more information about the get_secret SQL function, see the section called "get_secret(secretId, secretType, key, roleArn) " (p. 458). Use the `SecretString` parameter.

sasl.mechanism

The security mechanism used to connect to your Kafka broker. This value is required when you specify `SASL_SSL` for `security.protocol`.

Valid values: `PLAIN`, `GSSAPI`.

sasl.plain.username

The user name used to retrieve the secret string from Secrets Manager. This value is required when you specify `SASL_SSL` for `security.protocol` and `PLAIN` for `sasl.mechanism`.

sasl.plain.password

The password used to retrieve the secret string from Secrets Manager. This value is required when you specify `SASL_SSL` for `security.protocol` and `PLAIN` for `sasl.mechanism`.

sasl.kerberos.keytab

The keytab file for Kerberos authentication in Secrets Manager. This value is required when you specify `SASL_SSL` for `security.protocol` and `GSSAPI` for `sasl.mechanism`.

This field supports substitution templates. You must use Secrets Manager to store the credentials required to connect to your Kafka broker. Use the get_secret SQL function to retrieve the value for this field. For more information about substitution templates, see the section called "Substitution templates" (p. 485). For more information about the get_secret SQL function, see the section called "get_secret(secretId, secretType, key, roleArn) " (p. 458).Use the `SecretBinary` parameter.

sasl.kerberos.service.name

The Kerberos principal name under which Apache Kafka runs. This value is required when you specify `SASL_SSL` for `security.protocol` and `GSSAPI` for `sasl.mechanism`.

sasl.kerberos.krb5.kdc

The hostname of the key distribution center (KDC) to which your Apache Kafka producer client connects. This value is required when you specify `SASL_SSL` for `security.protocol` and `GSSAPI` for `sasl.mechanism`.

sasl.kerberos.krb5.realm

The realm to which your Apache Kafka producer client connects. This value is required when you specify `SASL_SSL` for `security.protocol` and `GSSAPI` for `sasl.mechanism`.

sasl.kerberos.principal

The unique Kerberos identity to which Kerberos can assign tickets to access Kerberos-aware services. This value is required when you specify `SASL_SSL` for `security.protocol` and `GSSAPI` for `sasl.mechanism`.

Examples

The following JSON example defines an Apache Kafka action in an AWS IoT rule.

```
{
"topicRulePayload": {
    "sql": "SELECT * FROM 'some/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
    {
        "kafka": {
            "destinationArn": "arn:aws:iot:region:123456789012:ruledestination/
vpc/VPCDestinationARN",
            "topic": "TopicName",
            "clientProperties": {
                "bootstrap.servers": "kafka.com:9092",
                "security.protocol": "SASL_SSL",
                "ssl.truststore": "${get_secret('kafka_client_truststore', 'SecretBinary',
'arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
                "ssl.truststore.password": "kafka password",
               "sasl.mechanism": "GSSAPI",
               "sasl.kerberos.service.name": "kafka",
               "sasl.kerberos.krb5.kdc": "kerberosdns.com",
               "sasl.kerberos.keytab": "${get_secret('kafka_keytab',
               'SecretBinary', 'arn:aws:iam::123456789012:role/kafka-get-secret-role-
name')}",
               "sasl.kerberos.krb5.realm": "KERBEROSREALM",
               "sasl.kerberos.principal": "kafka-keytab/kafka-keytab.com"
            }
        }
    }
]

}
```

Important notes about your Kerberos setup

- Your key distribution center (KDC) must be resolvable through private Domain Name System (DNS) within your target VPC. One possible approach is to add the KDC DNS entry to a private hosted zone. For more information about this approach, see Working with private hosted zones.
- Each VPC must have DNS resolution enabled. For more information, see Using DNS with your VPC.
- Network interface security groups and instance-level security groups in the VPC destination must allow traffic from within your VPC on the following ports.
  - TCP traffic on the bootstrap broker listener port (often 9092, but must be within the 9000 - 9100 range)
  - TCP and UDP traffic on port 88 for the KDC

CloudWatch alarms

The CloudWatch alarm (`cloudWatchAlarm`) action changes the state of an Amazon CloudWatch alarm. You can specify the state change reason and value in this call.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `cloudwatch:SetAlarmState` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`alarmName`

The CloudWatch alarm name.

Supports substitution templates (p. 485): API and AWS CLI only

`stateReason`

Reason for the alarm change.

Supports substitution templates (p. 485): Yes

`stateValue`

The value of the alarm state. Valid values: `OK`, `ALARM`, `INSUFFICIENT_DATA`.

Supports substitution templates (p. 485): Yes

`roleArn`

The IAM role that allows access to the CloudWatch alarm. For more information, see Requirements (p. 365).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a CloudWatch alarm action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchAlarm": {
                    "alarmName": "IotAlarm",
                    "stateReason": "Temperature stabilized.",
                    "stateValue": "OK",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

See also

- What is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*
- Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*

CloudWatch Logs

The CloudWatch Logs (`cloudwatchLogs`) action sends data to Amazon CloudWatch Logs. You can specify the log group to which the action sends data.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `logs:CreateLogStream`, `logs:DescribeLogStreams`, and `logs:PutLogEvents` operations. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt log data in CloudWatch Logs, the service must have permission to use the CMK on the caller's behalf. For more information, see Encrypt log data in CloudWatch Logs using AWS KMS in the *Amazon CloudWatch Logs User Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`logGroupName`

The CloudWatch log group to which the action sends data.

Supports substitution templates (p. 485): API and AWS CLI only

`roleArn`

The IAM role that allows access to the CloudWatch log group. For more information, see Requirements (p. 366).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a CloudWatch Logs action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchLogs": {
                    "logGroupName": "IotLogs",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

See also

- What is Amazon CloudWatch Logs? in the *Amazon CloudWatch Logs User Guide*

CloudWatch metrics

The CloudWatch metric (`cloudwatchMetric`) action captures an Amazon CloudWatch metric. You can specify the metric namespace, name, value, unit, and timestamp.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `cloudwatch:PutMetricData` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`metricName`

The CloudWatch metric name.

Supports substitution templates (p. 485): Yes

`metricNamespace`

The CloudWatch metric namespace name.

Supports substitution templates (p. 485): Yes

`metricUnit`

The metric unit supported by CloudWatch.

Supports substitution templates (p. 485): Yes

`metricValue`

A string that contains the CloudWatch metric value.

Supports substitution templates (p. 485): Yes

`metricTimestamp`

(Optional) A string that contains the timestamp, expressed in seconds in Unix epoch time. Defaults to the current Unix epoch time.

Supports substitution templates (p. 485): Yes

`roleArn`

The IAM role that allows access to the CloudWatch metric. For more information, see Requirements (p. 368).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a CloudWatch metric action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "IotMetric",
                    "metricNamespace": "IotNamespace",
                    "metricUnit": "Count",
                    "metricValue": "1",
                    "metricTimestamp": "1456821314",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

The following JSON example defines a CloudWatch metric action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "${topic()}",
                    "metricNamespace": "${namespace}",
                    "metricUnit": "${unit}",
                    "metricValue": "${value}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

See also

- What is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*
- Using Amazon CloudWatch metrics in the *Amazon CloudWatch User Guide*

DynamoDB

The DynamoDB (`dynamoDB`) action writes all or part of an MQTT message to an Amazon DynamoDB table.

You can follow a tutorial that shows you how to create and test a rule with a DynamoDB action. For more information, see Store device data in a DynamoDB table (p. 148).

Note
This rule writes non-JSON data to DynamoDB as binary data. The DynamoDB console displays the data as base64-encoded text.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `dynamodb:PutItem` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt data at rest in DynamoDB, the service must have permission to use the CMK on the caller's behalf. For more information, see Customer Managed CMK in the *Amazon DynamoDB Getting Started Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`tableName`

The name of the DynamoDB table.

Supports substitution templates (p. 485): API and AWS CLI only

`hashKeyField`

The name of the hash key (also called the partition key).

Supports substitution templates (p. 485): API and AWS CLI only

`hashKeyType`

(Optional) The data type of the hash key (also called the partition key). Valid values: `STRING`, `NUMBER`.

Supports substitution templates (p. 485): API and AWS CLI only

`hashKeyValue`

The value of the hash key. Consider using a substitution template such as `${topic()}` or `${timestamp()}`.

Supports substitution templates (p. 485): Yes

`rangeKeyField`

(Optional) The name of the range key (also called the sort key).

Supports substitution templates (p. 485): API and AWS CLI only

`rangeKeyType`

(Optional) The data type of the range key (also called the sort key). Valid values: `STRING`, `NUMBER`.

Supports substitution templates (p. 485): API and AWS CLI only

`rangeKeyValue`

(Optional) The value of the range key. Consider using a substitution template such as `${topic()}` or `${timestamp()}`.

Supports substitution templates (p. 485): Yes

`payloadField`

(Optional) The name of the column where the payload is written. If you omit this value, the payload is written to the column named `payload`.

Supports substitution templates (p. 485): Yes

`operation`

(Optional) The type of operation to be performed. Valid values: `INSERT`, `UPDATE`, `DELETE`.

Supports substitution templates (p. 485): Yes

`roleARN`

The IAM role that allows access to the DynamoDB table. For more information, see Requirements (p. 369).

Supports substitution templates (p. 485): No

The data written to the DynamoDB table is the result from the SQL statement of the rule.

Examples

The following JSON example defines a DynamoDB action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDB": {
                    "tableName": "my_ddb_table",
                    "hashKeyField": "key",
                    "hashKeyValue": "${topic()}",
                    "rangeKeyField": "timestamp",
                    "rangeKeyValue": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
                }
            }
        ]
    }
}
```

See also

- What is Amazon DynamoDB? in the *Amazon DynamoDB Developer Guide*
- Getting started with DynamoDB in the *Amazon DynamoDB Developer Guide*
- Store device data in a DynamoDB table (p. 148)

DynamoDBv2

The DynamoDBv2 (`dynamoDBv2`) action writes all or part of an MQTT message to an Amazon DynamoDB table. Each attribute in the payload is written to a separate column in the DynamoDB database.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `dynamodb:PutItem` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- The MQTT message payload must contain a root-level key that matches the table's primary partition key and a root-level key that matches the table's primary sort key, if one is defined.
- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt data at rest in DynamoDB, the service must have permission to use the CMK on the caller's

behalf. For more information, see Customer Managed CMK in the *Amazon DynamoDB Getting Started Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`putItem`

An object that specifies the DynamoDB table to which the message data will be written. This object must contain the following information:

`tableName`

The name of the DynamoDB table.

Supports substitution templates (p. 485): API and AWS CLI only

`roleARN`

The IAM role that allows access to the DynamoDB table. For more information, see Requirements (p. 371).

Supports substitution templates (p. 485): No

The data written to the DynamoDB table is the result from the SQL statement of the rule.

Examples

The following JSON example defines a DynamoDBv2 action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "my_ddb_table"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2",
                }
            }
        ]
    }
}
```

The following JSON example defines a DynamoDB action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2015-10-08",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "${topic()}"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2"
                }
            }
        ]
    }
}
```

See also

- What is Amazon DynamoDB? in the *Amazon DynamoDB Developer Guide*
- Getting started with DynamoDB in the *Amazon DynamoDB Developer Guide*

Elasticsearch

The Elasticsearch (`elasticsearch`) action writes data from MQTT messages to an Amazon Elasticsearch Service domain. You can then use tools like Kibana to query and visualize data in Elasticsearch.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `es:ESHttpPut` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt data at rest in Elasticsearch, the service must have permission to use the CMK on the caller's behalf. For more information, see Encryption of data at rest for Amazon Elasticsearch Service in the *Amazon Elasticsearch Service Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`endpoint`

  The endpoint of your Amazon Elasticsearch Service domain.

  Supports substitution templates (p. 485): API and AWS CLI only

`index`

  The Elasticsearch index where you want to store your data.

  Supports substitution templates (p. 485): Yes

`type`

  The type of document you are storing.

  Supports substitution templates (p. 485): Yes

`id`

  The unique identifier for each document.

  Supports substitution templates (p. 485): Yes

`roleARN`

The IAM role that allows access to the Elasticsearch domain. For more information, see Requirements (p. 373).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an Elasticsearch action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "my-type",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

The following JSON example defines an Elasticsearch action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

See also

- What is Amazon Elasticsearch Service? in the *Amazon Elasticsearch Service Developer Guide*

HTTPS

The HTTPS (`http`) action sends data from an MQTT message to a web application or service.

Requirements

This rule action has the following requirements:

- You must confirm and enable HTTPS endpoints before the rules engine can use them. For more information, see Working with topic rule destinations (p. 403).

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`url`

The HTTPS endpoint where the message is sent using the HTTP POST method. If you use an IP address in place of a hostname, it must be an IPv4 address. IPv6 addresses are not supported.

Supports substitution templates (p. 485): Yes

`confirmationUrl`

(Optional) If specified, AWS IoT uses the confirmation URL to create a matching topic rule destination. You must enable the topic rule destination before using it in an HTTPS action. For more information, see Working with topic rule destinations (p. 403). If you use substitution templates, you must manually create topic rule destinations before the `http` action can be used. `confirmationUrl` must be a prefix of `url`.

The relationship between `url` and `confirmationUrl` is described by the following:

- If `url` is hardcoded and `confirmationUrl` is not provided, we implicitly treat the `url` field as the `confirmationUrl`. AWS IoT creates a topic rule destination for `url`.
- If `url` and `confirmationUrl` are hardcoded, `url` must begin with `confirmationUrl`. AWS IoT creates a topic rule destination for `confirmationUrl`.
- If `url` contains a substitution template, you must specify `confirmationUrl` and `url` must begin with `confirmationUrl`. If `confirmationUrl` contains substitution templates, you must manually create topic rule destinations before the `http` action can be used. If `confirmationUrl` does not contain substitution templates, AWS IoT creates a topic rule destination for `confirmationUrl`.

Supports substitution templates (p. 485): Yes

`headers`

(Optional) The list of headers to include in HTTP requests to the endpoint. Each header must contain the following information:

`key`

The key of the header.

Supports substitution templates (p. 485): No

`value`

The value of the header.

Supports substitution templates (p. 485): Yes

Note
The default content type is application/json when the payload is in JSON format. Otherwise, it is application/octet-stream. You can overwrite it by specifying the exact content type in the header with the key content-type (case insensitive).

`auth`

(Optional) The authentication used by the rules engine to connect to the endpoint URL specified in the `url` argument. Currently, Signature Version 4 is the only supported authentication type. For more information, see HTTP Authorization.

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an AWS IoT rule with an HTTPS action.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                        }
                    ]
                }
            }
        ]
    }
}
```

HTTP action retry logic

The AWS IoT rules engine retries the HTTPS action according to these rules:

- The rules engine tries to send a message at least once.
- The rules engine retries at most twice. The maximum number of tries is three.
- The rules engine does not attempt a retry if:
  - The previous try provided a response larger than 16384 bytes.
  - The downstream web service or application closes the TCP connection after the try.
  - The total time to complete a request with retries exceeded the request timeout limit.
  - The request returns an HTTP status code other than 429, 500-599.

Note
Standard data transfer costs apply to retries.

See also

- Working with topic rule destinations (p. 403)

- Route data directly from AWS IoT Core to your web services in the *Internet of Things on AWS* blog

IoT Analytics

The IoT Analytics (`iotAnalytics`) action sends data from an MQTT message to an AWS IoT Analytics channel.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `iotanalytics:BatchPutMessage` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

  The policy attached to the role you specify should look like the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotanalytics:BatchPutMessage",
            "Resource": [
                "arn:aws:iotanalytics:us-west-2:account-id:channel/mychannel"
            ]
        }
    ]
}
```

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`batchMode`

(Optional) Whether to process the action as a batch. The default value is `false`.

When `batchMode` is `true` and the rule SQL statement evaluates to an Array, each Array element is delivered as a separate message when passed by `BatchPutMessage` to the AWS IoT Analytics channel. The resulting array can't have more than 100 messages.

Supports substitution templates (p. 485): No

`channelName`

The name of the AWS IoT Analytics channel to which to write the data.

Supports substitution templates (p. 485): API and AWS CLI only

`roleArn`

The IAM role that allows access to the AWS IoT Analytics channel. For more information, see Requirements (p. 377).

Supports substitution templates (p. 485): No

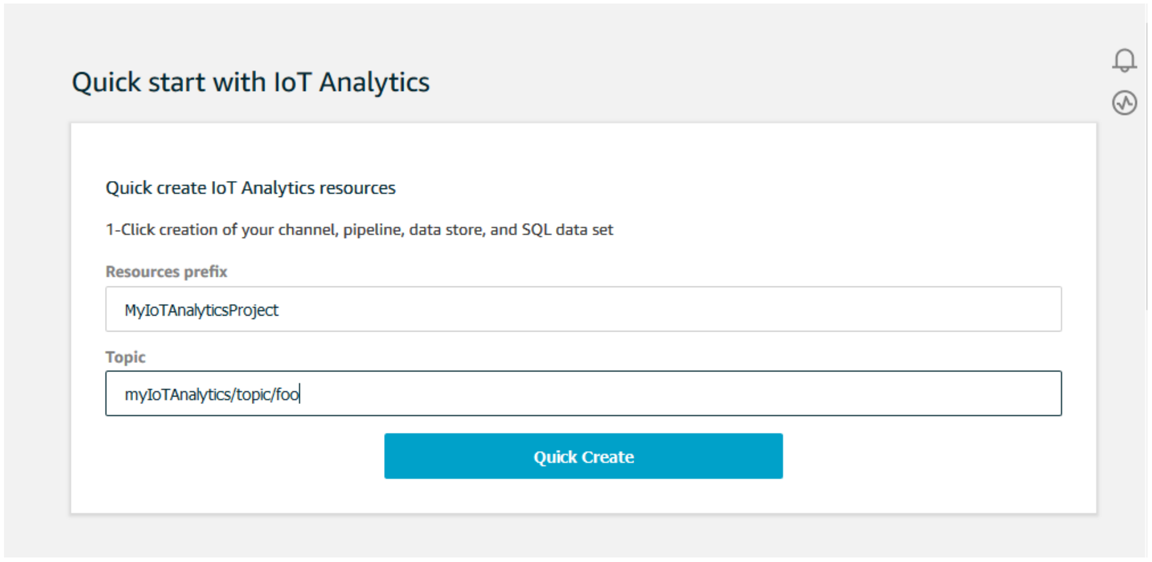
Examples

The following JSON example defines an IoT Analytics action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotAnalytics": {
                    "channelName": "mychannel",
                    "roleArn": "arn:aws:iam::123456789012:role/analyticsRole",
                }
            }
        ]
    }
}
```

See also

- What is AWS IoT Analytics? in the *AWS IoT Analytics User Guide*
- The AWS IoT Analytics console also has a **Quick start** feature that lets you create a channel, data store, pipeline, and data store with one click. For more information, see AWS IoT Analytics console quickstart guide in the *AWS IoT Analytics User Guide*.


Quick start with IoT Analytics

Quick create IoT Analytics resources

1-Click creation of your channel, pipeline, data store, and SQL data set

Resources prefix

MyIoTAnalyticsProject

Topic

myIoTAnalytics/topic/foo

IoT Events

The IoT Events (`iotEvents`) action sends data from an MQTT message to an AWS IoT Events input.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `iotevents:BatchPutMessage` operation. For more information, see Granting AWS IoT the required access (p. 353).

In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`batchMode`

(Optional) Whether to process the event actions as a batch. The default value is `false`.

When `batchMode` is `true` and the rule SQL statement evaluates to an Array, each Array element is treated as a separate message when it's sent to AWS IoT Events by calling `BatchPutMessage`. The resulting array can't have more than 10 messages.

When `batchMode` is `true`, you can't specify a `messageId`.

Supports substitution templates (p. 485): No

`inputName`

The name of the AWS IoT Events input.

Supports substitution templates (p. 485): API and AWS CLI only

`messageId`

(Optional) Use this to ensure that only one input (message) with a given `messageId` is processed by an AWS IoT Events detector. You can use the `${newuuid()}` substitution template to generate a unique ID for each request.

When `batchMode` is `true`, you can't specify a `messageId`--a new UUID value will be assigned.

Supports substitution templates (p. 485): Yes

`roleArn`

The IAM role that allows AWS IoT to send an input to an AWS IoT Events detector. For more information, see Requirements (p. 378).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an IoT Events action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotEvents": {
                    "inputName": "MyIoTEventsInput",
                    "messageId": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_events"
                }
            }
        ]
    }
}
```

See also

- What is AWS IoT Events? in the *AWS IoT Events Developer Guide*

IoT SiteWise

The IoT SiteWise (`iotSiteWise`) action sends data from an MQTT message to asset properties in AWS IoT SiteWise.

You can follow a tutorial that shows you how to ingest data from AWS IoT things. For more information, see Ingesting data to AWS IoT SiteWise from AWS IoT things in the *AWS IoT SiteWise User Guide*.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `iotsitewise:BatchPutAssetPropertyValue` operation. For more information, see Granting AWS IoT the required access (p. 353).

  You can attach the following example trust policy to the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        }
    ]
}
```

  To improve security, you can specify an AWS IoT SiteWise asset hierarchy path in the `Condition` property. The following example is a trust policy that specifies an asset hierarchy path.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        }
    ]
}
```

- When you send data to AWS IoT SiteWise with this action, your data must meet the requirements of the `BatchPutAssetPropertyValue` operation. For more information, see BatchPutAssetPropertyValue in the *AWS IoT SiteWise API Reference*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`putAssetPropertyValueEntries`

A list of asset property value entries that each contain the following information:

`propertyAlias`

(Optional) The property alias associated with your asset property. You must specify either a `propertyAlias` or both an `assetId` and a `propertyId`. For more information about property aliases, see Mapping industrial data streams to asset properties in the *AWS IoT SiteWise User Guide*.

Supports substitution templates (p. 485): Yes

`assetId`

(Optional) The ID of the AWS IoT SiteWise asset. You must specify either a `propertyAlias` or both an `assetId` and a `propertyId`.

Supports substitution templates (p. 485): Yes

`propertyId`

(Optional) The ID of the asset's property. You must specify either a `propertyAlias` or both an `assetId` and a `propertyId`.

Supports substitution templates (p. 485): API and AWS CLI only

`entryId`

(Optional) A unique identifier for this entry. You can define the `entryId` to better track which message caused an error in case of failure. Defaults to a new UUID.

Supports substitution templates (p. 485): Yes

`propertyValues`

A list of property values to insert that each contain timestamp, quality, and value (TQV) in the following format:

`timestamp`

A timestamp structure that contains the following information:

`timeInSeconds`

A string that contains the time in seconds in Unix epoch time. If your message payload doesn't have a timestamp, you can use timestamp() (p. 479), which returns the current time in milliseconds. To convert that time to seconds, you can use the following substitution template: **`${floor(timestamp() / 1E3)}`**.

Supports substitution templates (p. 485): Yes

`offsetInNanos`

(Optional) A string that contains the nanosecond time offset from the time in seconds. If your message payload doesn't have a timestamp, you can use timestamp() (p. 479), which returns the current time in milliseconds. To calculate the nanosecond offset from that time, you can use the following substitution template: **`${(timestamp() % 1E3) * 1E6}`**.

Supports substitution templates (p. 485): Yes

With respect to Unix epoch time, AWS IoT SiteWise accepts only entries that have a timestamp of up to 7 days in the past and up to 5 minutes in the future.

`quality`

(Optional) A string that describes the quality of the value. Valid values: `GOOD`, `BAD`, `UNCERTAIN`.

Supports substitution templates (p. 485): Yes

`value`

A value structure that contains one of the following value fields, depending on the asset property's data type:

`booleanValue`

(Optional) A string that contains the Boolean value of the value entry.

Supports substitution templates (p. 485): Yes

`doubleValue`

(Optional) A string that contains the double value of the value entry.

Supports substitution templates (p. 485): Yes

`integerValue`

(Optional) A string that contains the integer value of the value entry.

Supports substitution templates (p. 485): Yes

`stringValue`

(Optional) The string value of the value entry.

Supports substitution templates (p. 485): Yes

`roleArn`

The ARN of the IAM role that grants AWS IoT permission to send an asset property value to AWS IoT SiteWise. For more information, see Requirements (p. 380).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a basic IoT SiteWise action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "/some/property/alias",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${my.payload.timeInSeconds}"
                                    },
                                    "value": {
                                        "integerValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

The following JSON example defines an IoT SiteWise action in an AWS IoT rule. This example uses the topic as the property alias and the `timestamp()` function. For example, if you publish data to `/company/windfarm/3/turbine/7/rpm`, this action sends the data to the asset property with a property alias that is the same as the topic that you specified.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "${topic()}",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${floor(timestamp() / 1E3)}",
                                        "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                                    },
                                    "value": {
                                        "doubleValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

See also

- What is AWS IoT SiteWise? in the *AWS IoT SiteWise User Guide*
- Ingesting data using AWS IoT Core rules in the *AWS IoT SiteWise User Guide*
- Ingesting data to AWS IoT SiteWise from AWS IoT things in the *AWS IoT SiteWise User Guide*
- Troubleshooting an AWS IoT SiteWise rule action in the *AWS IoT SiteWise User Guide*

Kinesis Data Firehose

The Kinesis Data Firehose(`firehose`) action sends data from an MQTT message to an Amazon Kinesis Data Firehose stream.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `firehose:PutRecord` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use Kinesis Data Firehose to send data to an Amazon S3 bucket, and you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Amazon S3, Kinesis Data Firehose must have access to your bucket and permission to use the CMK on the caller's behalf. For more information, see Grant Kinesis Data Firehose access to an Amazon S3 destination in the *Amazon Kinesis Data Firehose Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`batchMode`

(Optional) Whether to deliver the Kinesis Data Firehose stream as a batch by using `PutRecordBatch` . The default value is `false`.

When `batchMode` is `true` and the rule's SQL statement evaluates to an Array, each Array element forms one record in the `PutRecordBatch` request. The resulting array can't have more than 500 records.

Supports substitution templates (p. 485): No

`deliveryStreamName`

The Kinesis Data Firehose stream to which to write the message data.

Supports substitution templates (p. 485): API and AWS CLI only

`separator`

(Optional) A character separator that is used to separate records written to the Kinesis Data Firehose stream. If you omit this parameter, the stream uses no separator. Valid values: `,` (comma), `\t` (tab), `\n` (newline), `\r\n` (Windows newline).

Supports substitution templates (p. 485): No

`roleArn`

The IAM role that allows access to the Kinesis Data Firehose stream. For more information, see Requirements (p. 384).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a Kinesis Data Firehose action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "my_firehose_stream",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

The following JSON example defines a Kinesis Data Firehose action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

See also

- What is Amazon Kinesis Data Firehose? in the *Amazon Kinesis Data Firehose Developer Guide*

Kinesis Data Streams

The Kinesis Data Streams (`kinesis`) action writes data from an MQTT message to Amazon Kinesis Data Streams.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `kinesis:PutRecord` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Kinesis Data Streams, the service must have permission to use the CMK on the caller's behalf. For more information, see Permissions to use user-generated KMS master keys in the *Amazon Kinesis Data Streams Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`stream`

  The Kinesis data stream to which to write data.

Supports substitution templates (p. 485): API and AWS CLI only

`partitionKey`

The partition key used to determine to which shard the data is written. The partition key is usually composed of an expression (for example, `${topic()}` or `${timestamp()}`).

Supports substitution templates (p. 485): Yes

`roleArn`

The ARN of the IAM role that grants AWS IoT permission to access the Kinesis data stream. For more information, see Requirements (p. 385).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a Kinesis Data Streams action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "my_kinesis_stream",
                    "partitionKey": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

The following JSON example defines a Kinesis action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "${topic()}",
                    "partitionKey": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

See also

- What is Amazon Kinesis Data Streams? in the *Amazon Kinesis Data Streams Developer Guide*

Lambda

A Lambda (`lambda`) action invokes an AWS Lambda function, passing in an MQTT message. AWS IoT invokes Lambda functions asynchronously.

You can follow a tutorial that shows you how to create and test a rule with a Lambda action. For more information, see Format a notification by using an AWS Lambda function (p. 154).

Requirements

This rule action has the following requirements:

- For AWS IoT to invoke a Lambda function, you must configure a policy that grants the `lambda:InvokeFunction` permission to AWS IoT. You can only invoke a Lambda function defined in the same AWS Region where your Lambda policy exists. Lambda functions use resource-based policies, so you must attach the policy to the Lambda function itself.

  Use the following AWS CLI command to attach a policy that grants the `lambda:InvokeFunction` permission.

  ```
  aws lambda add-permission --function-name function_name --region region --principal
   iot.amazonaws.com --source-arn arn:aws:iot:region:account-id:rule/rule_name --source-
  account account-id --statement-id unique_id --action "lambda:InvokeFunction"
  ```

  The `add-permission` command expects the following parameters:

  `--function-name`

  Name of the Lambda function. You add a new permission to update the function's resource policy.

  `--region`

  The AWS Region of the function.

  `--principal`

  The principal that gets the permission. This should be `iot.amazonaws.com` to allow AWS IoT permission to call the Lambda function.

  `--source-arn`

  The ARN of the rule. You can use the `get-topic-rule` AWS CLI command to get the ARN of a rule.

  `--source-account`

  The AWS account where the rule is defined.

  `--statement-id`

  A unique statement identifier.

  `--action`

  The Lambda action you want to allow in this statement. To allow AWS IoT to invoke a Lambda function, specify `lambda:InvokeFunction`.

  Important
  If you add a permission for an AWS IoT principal without providing the source ARN, any AWS account that creates a rule with your Lambda action can trigger rules to invoke your Lambda function from AWS IoT.

  For more information, see AWS Lambda permissions.

- If you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Lambda, the service must have permission to use the CMK on the caller's behalf. For more information, see Encryption at rest in the *AWS Lambda Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`functionArn`

The ARN of the Lambda function to invoke. AWS IoT must have permission to invoke the function. For more information, see Requirements (p. 387).

If you don't specify a version or alias for your Lambda function, the most recent version of the function is executed. You can specify a version or alias if you want to execute a specific version of your Lambda function. To specify a version or alias, append the version or alias to the ARN of the Lambda function.

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

For more information about versioning and aliases see AWS Lambda function versioning and aliases.

Supports substitution templates (p. 485): API and AWS CLI only

Examples

The following JSON example defines a Lambda action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
                 }
            }
        ]
    }
}
```

The following JSON example defines a Lambda action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

See also

- What is AWS Lambda? in the *AWS Lambda Developer Guide*
- Format a notification by using an AWS Lambda function (p. 154)

Republish

The republish (`republish`) action republishes an MQTT message to another MQTT topic.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `iot:Publish` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`topic`

The MQTT topic to which to republish the message.

To republish to a reserved topic, which begins with `$`, use `$$` instead. For example, to republish to the device shadow topic `$aws/things/MyThing/shadow/update`, specify the topic as `$$aws/things/MyThing/shadow/update`.

> **Note**
> Republishing to reserved job topics (p. 92) is not supported.

Supports substitution templates (p. 485): Yes

`qos`

(Optional) The Quality of Service (QoS) level to use when republishing messages. Valid values: `0`, `1`. The default value is `0`. For more information about MQTT QoS, see MQTT (p. 78).

Supports substitution templates (p. 485): No

`roleArn`

The IAM role that allows AWS IoT to publish to the MQTT topic. For more information, see Requirements (p. 389).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a republish action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "another/topic",
                    "qos": 1,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

The following JSON example defines a republish action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

S3

The S3 (`s3`) action writes the data from an MQTT message to an Amazon Simple Storage Service (Amazon S3) bucket.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `s3:PutObject` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Amazon S3, the service must have permission to use the CMK on the caller's behalf. For more information, see AWS managed CMKs and customer managed CMKs in the *Amazon Simple Storage Service Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`bucket`

    The Amazon S3 bucket to which to write data.

Supports substitution templates (p. 485): API and AWS CLI only

`cannedacl`

(Optional) The Amazon S3 canned ACL that controls access to the object identified by the object key. For more information, including allowed values, see Canned ACL.

Supports substitution templates (p. 485): No

`key`

The path to the file where the data is written.

Consider an example where this parameter is `${topic()}/${timestamp()}` and the rule receives a message where the topic is `some/topic`. If the current timestamp is `1460685389`, then this action writes the data to a file called `1460685389` in the `some/topic` folder of the S3 bucket.

Note
If you use a static key, AWS IoT overwrites a single file each time the rule invokes. We recommend that you use the message timestamp or another unique message identifier so that a new file is saved in Amazon S3 for each message received.

Supports substitution templates (p. 485): Yes

`roleArn`

The IAM role that allows access to the Amazon S3 bucket. For more information, see Requirements (p. 390).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an S3 action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "bucketName": "my-bucket",
                    "cannedacl": "public-read",
                    "key": "${topic()}/${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3"
                }
            }
        ]
    }
}
```

See also

- What is Amazon S3? in the *Amazon Simple Storage Service Developer Guide*

Salesforce IoT

The Salesforce IoT (`salesforce`) action sends data from the MQTT message that triggered the rule to a Salesforce IoT input stream.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`url`

The URL exposed by the Salesforce IoT input stream. The URL is available from the Salesforce IoT platform when you create an input stream. For more information, see the Salesforce IoT documentation.

Supports substitution templates (p. 485): No

`token`

The token used to authenticate access to the specified Salesforce IoT input stream. The token is available from the Salesforce IoT platform when you create an input stream. For more information, see the Salesforce IoT documentation.

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a Salesforce IoT action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "salesforce": {
                    "token": "ABCDEFGHI123456789abcdefghi123456789",
                    "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/stream-
id/connection-id/my-event"
                }
            }
        ]
    }
}
```

SNS

The SNS (`sns`) action sends the data from an MQTT message as an Amazon Simple Notification Service (Amazon SNS) push notification.

You can follow a tutorial that shows you how to create and test a rule with an SNS action. For more information, see Send an Amazon SNS notification (p. 141).

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `sns:Publish` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

- If you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Amazon SNS, the service must have permission to use the CMK on the caller's behalf. For more information, see Key management in the *Amazon Simple Notification Service Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`targetArn`

The SNS topic or individual device to which the push notification is sent.

Supports substitution templates (p. 485): API and AWS CLI only

`messageFormat`

(Optional) The message format. Amazon SNS uses this setting to determine if the payload should be parsed and if relevant platform-specific parts of the payload should be extracted. Valid values: `JSON`, `RAW`. Defaults to `RAW`.

Supports substitution templates (p. 485): No

`roleArn`

The IAM role that allows access to SNS. For more information, see Requirements (p. 392).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an SNS action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

The following JSON example defines an SNS action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-1:123456789012:${topic()}",
                    "messageFormat": "JSON",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

See also

- What is Amazon Simple Notification Service? in the *Amazon Simple Notification Service Developer Guide*
- Send an Amazon SNS notification (p. 141)

SQS

The SQS (`sqs`) action sends data from an MQTT message to an Amazon Simple Queue Service (Amazon SQS) queue.

> **Note**
> The SQS action doesn't support Amazon SQS FIFO (First-In-First-Out) queues. Because the rules engine is a fully distributed service, there is no guarantee of message order when the SQS action is triggered.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `sqs:SendMessage` operation. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.
- If you use an AWS Key Management Service (AWS KMS) customer managed key (CMK) to encrypt data at rest in Amazon SQS, the service must have permission to use the CMK on the caller's behalf. For more information, see Key management in the *Amazon Simple Queue Service Developer Guide*.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`queueUrl`

The URL of the Amazon SQS queue to which to write the data.

Supports substitution templates (p. 485): API and AWS CLI only

`useBase64`

Set this parameter to `true` to configure the rule action to base64-encode the message data before it writes the data to the Amazon SQS queue. Defaults to `false`.

Supports substitution templates (p. 485): No

`roleArn`

The IAM role that allows access to the Amazon SQS queue. For more information, see Requirements (p. 394).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines an SQS action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
my_sqs_queue",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

The following JSON example defines an SQS action with substitution templates in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
${topic()}",
                    "useBase64": true,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

See also

- What is Amazon Simple Queue Service? in the *Amazon Simple Queue Service Developer Guide*

Step Functions

The Step Functions (`stepFunctions`) action starts an AWS Step Functions state machine.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `states:StartExecution` operation. For more information, see Granting AWS IoT the required access (p. 353).

In the AWS IoT console, you can choose or create a role to allow AWS IoT to perform this rule action.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`stateMachineName`

The name of the Step Functions state machine to start.

Supports substitution templates (p. 485): API and AWS CLI only

`executionNamePrefix`

(Optional) The name given to the state machine execution consists of this prefix followed by a UUID. Step Functions creates a unique name for each state machine execution if one is not provided.

Supports substitution templates (p. 485): Yes

`roleArn`

The ARN of the role that grants AWS IoT permission to start the state machine. For more information, see Requirements (p. 395).

Supports substitution templates (p. 485): No

Examples

The following JSON example defines a Step Functions action in an AWS IoT rule.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "stepFunctions": {
                    "stateMachineName": "myStateMachine",
                    "executionNamePrefix": "myExecution",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_step_functions"
                }
            }
        ]
    }
}
```

See also

- What is AWS Step Functions? in the *AWS Step Functions Developer Guide*

Timestream

The Timestream rule action writes attributes (measures) from an MQTT message into an Amazon Timestream table.

Important
The attributes that this rule stores in the Timestream database are those that result from the rule's SQL statement. Each attribute is stored as a separate record in the Timestream database table.
For more information about Amazon Timestream, see What Is Amazon Timestream?.

The value of each attribute in the SQL statement's result is parsed to infer its data type (as in a the section called "DynamoDBv2" (p. 371) action). Each attribute's value is written to its own record in the Timestream table. To specify or change a value's data type, use the `cast()` (p. 448) function in rule's the SQL statement. For more information about the contents of each Timestream record, see the section called "Amazon Timestream record content" (p. 398).

Note
With SQL V2 (2016-03-23), numeric values that are whole numbers, such as `10.0`, are converted their Integer representation (`10`). Explicitly casting them to a `Decimal` value, such as by using the cast() (p. 448) function, does not prevent this behavior—the result is still an `Integer` value. This can cause type mismatch errors that prevent data from being recorded in the Timestream database. To reliably process whole number numeric values as `Decimal` values, use SQL V1 (2015-10-08) for the rule query statement.

Requirements

This rule action has the following requirements:

- An IAM role that AWS IoT can assume to perform the `timestream:DescribeEndpoints` and `timestream:WriteRecords` operations. For more information, see Granting AWS IoT the required access (p. 353).

  In the AWS IoT console, you can choose, update, or create a role to allow AWS IoT to perform this rule action.
- If you use a customer-managed AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt data at rest in Timestream, the service must have permission to use the CMK on the caller's behalf. For more information, see How AWS services use AWS KMS.

Parameters

When you create an AWS IoT rule with this action, you must specify the following information:

`databaseName`

The name of an Amazon Timestream database that has the table to receive the records this action creates. See also **`tableName`**.

Supports substitution templates (p. 485): API and AWS CLI only

`dimensions`

Metadata attributes of the time series that are written in each measure record. For example, the name and Availability Zone of an EC2 instance or the name of the manufacturer of a wind turbine are dimensions.

`name`

The metadata dimension name. This is the name of the column in the database table record.

Dimensions can't be named: `measure_name`, `measure_value`, or `time`. These names are reserved. Dimension names can't start with `ts_` or `measure_value` and they can't contain the colon (`:`) character.

Supports substitution templates (p. 485): No

`value`

The value to write in this column of the database record.

Supports substitution templates (p. 485): Yes

`roleArn`

The Amazon Resource Name (ARN) of the role that grants AWS IoT permission to write to the Timestream database table. For more information, see Requirements (p. 397).

Supports substitution templates (p. 485): No

`tableName`

The name of the database table into which to write the measure records. See also **`databaseName`**.

Supports substitution templates (p. 485): API and AWS CLI only

`timestamp`

The value to use for the entry's timestamp. If blank, the time that the entry was processed is used.

`unit`

The precision of the timestamp value that results from the expression described in `value`.

Valid values: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`. The default is `MILLISECONDS`.

`value`

An expression that returns a long epoch time value.

You can use the the section called "time_to_epoch(String, String)" (p. 478) function to create a valid timestamp from a date or time value passed in the message payload.

Amazon Timestream record content

The data written to the Amazon Timestream table by this action include a timestamp, metadata from the Timestream rule action, and the result of the rule's SQL statement.

For each attribute (measure) in the result of the SQL statement, this rule action writes a record to the specified Timestream table with these columns.

Column name	Attribute type	Value	Comments
`dimension-name`	DIMENSION	The value specified in the Timestream rule action entry.	Each **Dimension** specified in the rule action entry creates a column in the Timestream database with the dimension's name.
measure_name	MEASURE_NAME	The attribute's name	The name of the attribute in the result of the SQL statement whose value is specified in the `measure_value::`*`data-type`* column.

Column name	Attribute type	Value	Comments
measure_value::*data-type*	MEASURE_VALUE	The value of the attribute in the result of the SQL statement. The attribute's name is in the `measure_name` column.	The value is interpreted* and cast as the best match of: `bigint`, `boolean`, `double`, or `varchar`. Amazon Timestream creates a separate column for each data type. The value in the message can be cast to another data type by using the `cast()` (p. 448) function in the rule's SQL statement.
time	TIMESTAMP	The date and time of the record in the database.	This value is assigned by rules engine or the `timestamp` property, if it is defined.

* The attribute value read from the message payload is interpreted as follows. See the the section called “Examples” (p. 399) for an illustration of each of these cases.

- An unquoted value of `true` or `false` is interpreted as a `boolean` type.
- A decimal numeric is interpreted as a `double` type.
- A numeric value without a decimal point is interpreted as a `bigint` type.
- A quoted string is interpreted as a `varchar` type.
- Objects and array values are converted to JSON strings and stored as a `varchar` type.

Examples

The following JSON example defines a Timestream rule action with a substitution template in an AWS IoT rule.

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "timestream": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_timestream",
          "tableName": "devices_metrics",
          "dimensions": [
            {
              "name": "device_id",
              "value": "${clientId()}"
            },
            {
              "name": "device_firmware_sku",
              "value": "My Static Metadata"
            }
          ],
          "databaseName": "record_devices"
        }
      }
    ]
  }
}
```

Using the Timestream topic rule action defined in the previous example with the following message payload results in the Amazon Timestream records written in the table that follows.

```
{
  "boolean_value": true,
  "integer_value": 123456789012,
  "double_value": 123.456789012,
  "string_value": "AWS IoT is super!",
  "boolean_value_as_string": "true",
  "integer_value_as_string": "123456789012",
  "double_value_as_string": "123.456789012",
  "array_of_integers": [23,36,56,72],
  "array of strings": ["red", "green","blue"],
  "complex_value": {
    "simple_element": 42,
    "array_of_integers": [23,36,56,72],
    "array of strings": ["red", "green","blue"]
  }
}
```

The following table displays the database columns and records that using the specified topic rule action to process the previous message payload creates. The `device_firmware_sku` and `device_id` columns are the DIMENSIONS defined in the topic rule action. The Timestream topic rule action creates the `time` column and the `measure_name` and `measure_value::*` columns, which it fills with the values from the result of the topic rule action's SQL statement.

device_firm	device_id	measure_na	measure_va	measure_va	measure_va	measure_va	time
My Static Metadata	iotconsole-159EXAMPLE738-0	complex_value	-	{"simple_element":42,"array_of_integers": [23,36,56,72],"array of strings": ["red","green","blue"]}	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	integer_value_as_string	-	123456789012	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	boolean_value	-	-	-	TRUE	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	integer_value	123456789012	-	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	string_value	-	AWS IoT is super!	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	array_of_integers	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	array of strings	-	["red","green","blue"]	-	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	boolean_value_as_string	-	TRUE	-	-	2020-08-26 22:42:16.423

device_firm	device_id	measure_na	measure_va	measure_va	measure_va	measure_va	time
My Static Metadata	iotconsole-159EXAMPLE738-0	double_value		-	123.456789012	-	2020-08-26 22:42:16.423
My Static Metadata	iotconsole-159EXAMPLE738-0	double_value_as_string		123.45679	-	-	2020-08-26 22:42:16.423

VPC

The virtual private cloud (VPC) action routes traffic to your Amazon Virtual Private Cloud (Amazon VPC). Unlike other rule actions, the VPC action doesn't require any configuration and is automatically enabled when you specify a VPC destination for your rule action. A VPC destination contains a list of subnets inside the VPC. The rules engine creates an elastic network interface in each subnet that you specify in this list.

Note
For more information about network interfaces, see Elastic network interfaces in the Amazon EC2 User Guide.

Currently the VPC action works with the following rule actions:

- the section called "Apache Kafka" (p. 360)

For pricing purposes, a VPC rule action is metered in addition to the action that sends a message to a resource when the resource is in your VPC. For pricing information, see AWS IoT Core pricing.

For information about creating VPC destinations, see the section called "Creating a VPC topic rule destination" (p. 404).

Troubleshooting a rule

If you have an issue with your rules, you should enable CloudWatch Logs. You can analyze your logs to determine whether the issue is authorization or whether, for example, a WHERE clause condition didn't match. For more information, see Setting Up CloudWatch Logs.

Error handling (error action)

When AWS IoT receives a message from a device, the rules engine checks to see if the message matches a rule. If so, the rule's SQL statement is evaluated and the rule's actions are triggered, passing the SQL statement's result.

If a problem occurs when triggering an action, the rules engine triggers an error action, if one is specified for the rule. This might happen when:

- A rule doesn't have permission to access an Amazon S3 bucket.
- A user error causes DynamoDB provisioned throughput to be exceeded.

Error action message format

A single message is generated per rule and message. For example, if two rule actions in the same rule fail, the error action receives one message that contains both errors.

The error action message looks like the following example.

```
{
  "ruleName": "TestAction",
  "topic": "testme/action",
  "cloudwatchTraceId": "7e146a2c-95b5-6caf-98b9-50e3969734c7",
  "clientId": "iotconsole-1511213971966-0",
  "base64OriginalPayload": "ewogICJtZXNzYWdlIjogIkhlbGxvIHZyb20gQVdTIElvVCBjb25zb2xlIgp9",
  "failures": [
    {
      "failedAction": "S3Action",
      "failedResource": "us-east-1-s3-verify-user",
      "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist (Service: Amazon S3; Status Code: 404; Error
 Code: NoSuchBucket; Request ID: 9DF5416B9B47B9AF; S3 Extended Request ID:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y=).
 Message arrived on: error/action, Action: s3, Bucket: us-
east-1-s3-verify-user, Key: \"aaa\". Value of x-amz-id-2:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y="
    }
  ]
}
```

ruleName

The name of the rule that triggered the error action.

topic

The topic on which the original message was received.

cloudwatchTraceId

A unique identity referring to the error logs in CloudWatch.

clientId

The client ID of the message publisher.

base64OriginalPayload

The original message payload Base64-encoded.

failures

failedAction

The name of the action that failed to complete (for example, "S3Action").

failedResource

The name of the resource (for example, the name of an S3 bucket).

errorMessage

The description and explanation of the error.

Error action example

Here is an example of a rule with an added error action. The following rule has an action that writes message data to a DynamoDB table and an error action that writes data to an Amazon S3 bucket:

```
{
    "sql" : "SELECT * FROM ..."
    "actions" : [{
        "dynamoDB" : {
            "table" : "PoorlyConfiguredTable",
            "hashKeyField" : "AConstantString",
            "hashKeyValue" : "AHashKey"}}
    ],
    "errorAction" : {
        "s3" : {
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
            "bucket" : "message-processing-errors",
            "key" : "${replace(topic(), '/', '-') + '-' + timestamp() + '-' + newuuid()}"
        }
    }
}
```

You can use any function or substitution in an error action's SQL statement, except for external functions (for example, `get_thing_shadow`, `aws_lambda`, and `machinelearning_predict`.)

For more information about rules and how to specify an error action, see Creating an AWS IoT Rule.

For more information about using CloudWatch to monitor the success or failure of rules, see AWS IoT metrics and dimensions (p. 321).

Working with topic rule destinations

A destination is a resource that defines where rules engine can route data. The AWS IoT rules engine supports two kinds of destinations: HTTP destinations and VPC destinations. Destinations make it possible for the rules engine to send data to other services that are not natively integrated with AWS IoT. A destination can be reused across rules.

HTTP destinations might require confirmation or configuration before they can be used. The following paragraphs describe how the confirmation flow for HTTP destinations works.

Topic rule destinations are used to verify that you own or have access to the endpoint to which you want to route data. When you create a rule action with an HTTP endpoint (for example, an `http` action) or update an existing rule action's endpoint, AWS IoT sends a confirmation message to the endpoint that contains a unique token. To verify your endpoint, you must call `ConfirmTopicRuleDestination` with the token to verify you own or have access to the endpoint. The token is valid for three days, after which you need to create a new `http` action or call the `UpdateTopicRuleDestination` API to restart the confirmation process. You can also confirm your topic rule destination by browsing to `enableUrl` or provide the token from the **Destination** page in the AWS IoT console.

When AWS IoT receives the token sent to your endpoint, the HTTP destination is confirmed. You must enable the destination before it can be used by rules engine. A destination can be in one of the following states:

ENABLED

The destination is enabled. A destination must be in the `ENABLED` state for it to be used in a rule. You can only enable a destinations in DISABLED status.

DISABLED

The destination has been confirmed but disabled. This is useful if you want to temporarily prevent traffic to your endpoint without having to go through the confirmation process again. You can only disable a destinations in ENABLED status.

IN_PROGRESS

Confirmation of the destination is in progress.

ERROR

Destination confirmation timed out.

After they are confirmed and enabled, destinations can be used with any rule in your account.

Creating an HTTP topic rule destination

A destination is created when you use AWS IoT to create a rule with an `http` action. You can also create a destination using the `CreateTopicRuleDestination` API or the AWS IoT console.

When you create a destination, AWS IoT verifies the endpoint URL is valid. If the URL is valid, a confirmation message is sent to that URL. The confirmation request has the following format:

```
HTTP POST {confirmationUrl}/?confirmationToken={confirmationToken}
Headers:
x-amz-rules-engine-message-type: DestinationConfirmation
x-amz-rules-engine-destination-arn:"arn:aws:iot:us-east-1:123456789012:ruledestination/
http/7a280e37-b9c6-47a2-a751-0703693f46e4"
Content-Type: application/json
Body:
{
    "arn":"arn:aws:iot:us-east-1:123456789012:ruledestination/http/7a280e37-b9c6-47a2-
a751-0703693f46e4",
    "confirmationToken": "AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "enableUrl": "https://iot.us-east-1.amazonaws.com/confirmdestination/
AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "messageType": "DestinationConfirmation"
}
```

The fields of this message are defined as follows:

arn

    The Amazon Resource Name (ARN) for the topic rule destination to confirm.

confirmationToken

    The confirmation token. The token in the example is truncated. Your token will be longer.

enableUrl

    The URL to which you browse to confirm a topic rule destination.

messageType

    The type of message.

Creating a VPC topic rule destination

You create a virtual private cloud (VPC) destination by using the CreateTopicRuleDestination API or the AWS IoT Core console. When you create a VPC destination, you must specify the following information.

vpcId

    The unique ID of the VPC destination.

subnetIds

    A list of subnets in which the rules engine creates elastic network interfaces. The rules engine allocates a single network interface for each subnet in the list.

securityGroups (optional)

    A list of security groups to apply to the network interfaces.

roleArn

    The ARN of a role that has permission to create network interfaces on your behalf.

This ARN should have a policy attached to it that looks like the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute"
            ],
            "Resource": "*"
        }
    ]
}
```

Creating a VPC destination by using the AWS CLI

The following example shows how to create a VPC destination by using the AWS CLI.

```
aws --region regions iot create-topic-rule-destination --destination-configuration
 'vpcConfiguration={subnetIds=["subnet-123456789101230456"],securityGroups=[],vpcId="vpc-
123456789101230456",roleArn="arn:aws:iam::123456789012:role/role-name"}'
```

After you run this command, the VPC destination status will be `IN PROGRESS`. After a few minutes, its status will change to either `ERROR` (if the command isn't successful) or `ENABLED`. When the destination status is `ENABLED`, it is ready to use.

You can use the following command to get the status of your VPC destination.

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

Creating a VPC destination by using the AWS IoT Core console

The following steps describe how to create a VPC destination by using the AWS IoT Core console.

1. Navigate to the AWS IoT Core console. Choose **Destinations** under the **Act** tab in the left pane.
2. Enter values for the following fields.
   - **VPC ID**
   - **Subnet IDs**
   - **Security Group**
3. Select a role that has the permissions required to create ENIs. The preceding example role contains these permissions.

When the VPC destination status is **ENABLED**, it is ready to use.

Confirming an HTTP topic rule destination

You can confirm a destination by making an HTTP GET request to the `enableUrl` that is included in the confirmation message. You can call `ConfirmTopicRuleDestination` with the token from the confirmation message or you can use the AWS IoT console.

The token must be valid for the destination to be put in the `ENABLED` state. If the token has expired, you must call `UpdateTopicRuleDestination` to restart the confirmation process.

> **Note**
> If you confirm the topic rule destination by calling `ConfirmTopicRuleDestination`, the destination status will be `DISABLED` and you need to explicitly enable your destination by calling `UpdateTopicRuleDestination` before using it.

Disabling a topic rule destination

To disable a destination, call `UpdateTopicRuleDestination` and set the topic rule destination's status to `DISABLED`.

Enabling a topic rule destination

To enable a destination, call `UpdateTopicRuleDestination` and set the topic rule's status to `ENABLED`. You do not need to re-validate the URL.

Sending a new confirmation message

To trigger a new confirmation message for a destination, call `UpdateTopicRuleDestination` and set the topic rule destination's status to `IN_PROGRESS`.

Deleting a topic rule destination

To delete a topic rule destination, call `DeleteTopicRuleDestination`.

Certificate authorities supported by HTTPS endpoints in topic rule destinations

The following certificate authorities are supported by HTTPS endpoints in topic rule destinations.

```
Alias name: swisssignplatinumg2ca
Certificate fingerprints:
  MD5:  C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6
  SHA1: 56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
  SHA256:
 3B:22:2E:56:67:11:E9:92:30:0D:C0:B1:5A:B9:47:3D:AF:DE:F8:C8:4D:0C:EF:7D:33:17:B4:C1:82:1D:14:36

Alias name: hellenicacademicandresearchinstitutionsrootca2011
Certificate fingerprints:
  MD5:  73:9F:4C:4B:73:5B:79:E9:FA:BA:1C:EF:6E:CB:D5:C9
  SHA1: FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
  SHA256:
 BC:10:4F:15:A4:8B:E7:09:DC:A5:42:A7:E1:D4:B9:DF:6F:05:45:27:E8:02:EA:A9:2D:59:54:44:25:8A:FE:71

Alias name: teliasonerarootcav1
```

```
Certificate fingerprints:
  MD5:  37:41:49:1B:18:56:9A:26:F5:AD:C2:66:FB:40:A5:4C
  SHA1: 43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
  SHA256:
 DD:69:36:FE:21:F8:F0:77:C1:23:A1:A5:21:C1:22:24:F7:22:55:B7:3E:03:A7:26:06:93:E8:A2:4B:0F:A3:89

Alias name: geotrustprimarycertificationauthority
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6C

Alias name: trustisfpsrootca
Certificate fingerprints:
  MD5:  30:C9:E7:1E:6B:E6:14:EB:65:B2:16:69:20:31:67:4D
  SHA1: 3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
  SHA256:
 C1:B4:82:99:AB:A5:20:8F:E9:63:0A:CE:55:CA:68:A0:3E:DA:5A:51:9C:88:02:A0:D3:A6:73:BE:8F:8E:55:7D

Alias name: quovadisrootca3g3
Certificate fingerprints:
  MD5:  DF:7D:B9:AD:54:6F:68:A1:DF:89:57:03:97:43:B0:D7
  SHA1: 48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
  SHA256:
 88:EF:81:DE:20:2E:B0:18:45:2E:43:F8:64:72:5C:EA:5F:BD:1F:C2:D9:D2:05:73:07:09:C5:D8:B8:69:0F:46

Alias name: buypassclass2ca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:48

Alias name: secureglobalca
Certificate fingerprints:
  MD5:  CF:F4:27:0D:D4:ED:DC:65:16:49:6D:3D:DA:BF:6E:DE
  SHA1: 3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
  SHA256:
 42:00:F5:04:3A:C8:59:0E:BB:52:7D:20:9E:D1:50:30:29:FB:CB:D4:1C:A1:B5:06:EC:27:F1:5A:DE:7D:AC:69

Alias name: chunghwaepkirootca
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D5

Alias name: verisignclass2g2ca
Certificate fingerprints:
  MD5:  2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
  SHA256:
 3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A1

Alias name: szafirrootca2
Certificate fingerprints:
  MD5:  11:64:C1:89:B0:24:B1:8C:B1:07:7E:89:9E:51:9E:99
  SHA1: E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
  SHA256:
 A1:33:9D:33:28:1A:0B:56:E5:57:D3:D3:2B:1C:E7:F9:36:7E:B0:94:BD:5F:A7:2A:7E:50:04:C8:DE:D7:CA:FE

Alias name: quovadisrootca1g3
Certificate fingerprints:
  MD5:  A4:BC:5B:3F:FE:37:9A:FA:64:F0:E2:FA:05:3D:0B:AB
  SHA1: 1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
  SHA256:
 8A:86:6F:D1:B2:76:B5:7E:57:8E:92:1C:65:82:8A:2B:ED:58:E9:F2:F2:88:05:41:34:B7:F1:F4:BF:C9:CC:74

Alias name: utndatacorpsgcca
Certificate fingerprints:
  MD5:  B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
  SHA1: 58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
  SHA256:
 85:FB:2F:91:DD:12:27:5A:01:45:B6:36:53:4F:84:02:4A:D6:8B:69:B8:EE:88:68:4F:F7:11:37:58:05:B3:48

Alias name: autoridaddecertificacionfirmaprofesionalcifa62634068
Certificate fingerprints:
  MD5:  73:3A:74:7A:EC:BB:A3:96:A6:C2:E4:E2:C8:9B:C0:C3
  SHA1: AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
  SHA256:
 04:04:80:28:BF:1F:28:64:D4:8F:9A:D4:D8:32:94:36:6A:82:88:56:55:3F:3B:14:30:3F:90:14:7F:5D:40:EF

Alias name: securesignrootca11
Certificate fingerprints:
  MD5:  B7:52:74:E2:92:B4:80:93:F2:75:E4:CC:D7:F2:EA:26
  SHA1: 3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
  SHA256:
 BF:0F:EE:FB:9E:3A:58:1A:D5:F9:E9:DB:75:89:98:57:43:D2:61:08:5C:4D:31:4F:6F:5D:72:59:AA:42:16:12

Alias name: amazon-ca-g4-acm2
Certificate fingerprints:
  MD5:  B2:F1:03:2B:93:64:05:80:B8:A8:17:36:B9:1B:52:3C
  SHA1: A7:E6:45:32:1F:7A:B7:AD:C0:70:EA:73:5F:AB:ED:C3:DA:B4:D0:C8
  SHA256:
 D7:A8:7C:69:95:D0:E2:04:2A:32:70:A7:E2:87:FE:A7:E8:F4:C1:70:62:F7:90:C3:EB:BB:53:F2:AC:39:26:BE

Alias name: isrgrootx1
Certificate fingerprints:
  MD5:  0C:D2:F9:E0:DA:17:73:E9:ED:86:4D:A5:E3:70:E7:4E
  SHA1: CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
  SHA256:
 96:BC:EC:06:26:49:76:F3:74:60:77:9A:CF:28:C5:A7:CF:E8:A3:C0:AA:E1:1A:8F:FC:EE:05:C0:BD:DF:08:C6

Alias name: amazon-ca-g4-acm1
Certificate fingerprints:
  MD5:  E2:F1:18:19:61:5C:43:E0:D4:A8:5D:0B:FA:7C:89:1B
  SHA1: F2:0D:28:B6:29:C2:2C:5E:84:05:E6:02:4D:97:FE:8F:A0:84:93:A0
  SHA256:
 B0:11:A4:F7:29:6C:74:D8:2B:F5:62:DF:87:D7:28:C7:1F:B5:8C:F4:E6:73:F2:78:FC:DA:F3:FF:83:A6:8C:87

Alias name: etugracertificationauthority
Certificate fingerprints:
  MD5:  B8:A1:03:63:B0:BD:21:71:70:8A:6F:13:3A:BB:79:49
  SHA1: 51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
  SHA256:
 B0:BF:D5:2B:B0:D7:D9:BD:92:BF:5D:4D:C1:3D:A2:55:C0:2C:54:2F:37:83:65:EA:89:39:11:F5:5E:55:F2:3C

Alias name: geotrustuniversalca2
Certificate fingerprints:
  MD5:  34:FC:B8:D0:36:DB:9E:14:B3:C2:F2:DB:8F:E4:94:C7
  SHA1: 37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
  SHA256:
 A0:23:4F:3B:C8:52:7C:A5:62:8E:EC:81:AD:5D:69:89:5D:A5:68:0D:C9:1D:1C:B8:47:7F:33:F8:78:B9:5B:0B

Alias name: digicertglobalrootca
Certificate fingerprints:
  MD5:  79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
  SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
  SHA256:
 43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:7F:89:34:A4:43:C7:01:61
```

```
Alias name: staatdernederlandenevrootca
Certificate fingerprints:
  MD5:  FC:06:AF:7B:E8:1A:F1:9A:B4:E8:D2:70:1F:C0:F5:BA
  SHA1: 76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
  SHA256:
 4D:24:91:41:4C:FE:95:67:46:EC:4C:EF:A6:CF:6F:72:E2:8A:13:29:43:2F:9D:8A:90:7A:C4:CB:5D:AD:C1:5A

Alias name: utnuserfirstclientauthemailca
Certificate fingerprints:
  MD5:  D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7
  SHA1: B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
  SHA256:
 43:F2:57:41:2D:44:0D:62:74:76:97:4F:87:7D:A8:F1:FC:24:44:56:5A:36:7A:E6:0E:DD:C2:7A:41:25:31:AE

Alias name: actalisauthenticationrootca
Certificate fingerprints:
  MD5:  69:C1:0D:4F:07:A3:1B:C3:FE:56:3D:04:BC:11:F6:A6
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:66

Alias name: amazonrootca4
Certificate fingerprints:
  MD5:  89:BC:27:D5:EB:17:8D:06:6A:69:D5:FD:89:47:B4:CD
  SHA1: F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
  SHA256:
 E3:5D:28:41:9E:D0:20:25:CF:A6:90:38:CD:62:39:62:45:8D:A5:C6:95:FB:DE:A3:C2:2B:0B:FB:25:89:70:92

Alias name: amazonrootca3
Certificate fingerprints:
  MD5:  A0:D4:EF:0B:F7:B5:D8:49:95:2A:EC:F5:C4:FC:81:87
  SHA1: 0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
  SHA256:
 18:CE:6C:FE:7B:F1:4E:60:B2:E3:47:B8:DF:E8:68:CB:31:D0:2E:BB:3A:DA:27:15:69:F5:03:43:B4:6D:B3:A4

Alias name: amazonrootca2
Certificate fingerprints:
  MD5:  C8:E5:8D:CE:A8:42:E2:7A:C0:2A:5C:7C:9E:26:BF:66
  SHA1: 5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
  SHA256:
 1B:A5:B2:AA:8C:65:40:1A:82:96:01:18:F8:0B:EC:4F:62:30:4D:83:CE:C4:71:3A:19:C3:9C:01:1E:A4:6D:B4

Alias name: amazonrootca1
Certificate fingerprints:
  MD5:  43:C6:BF:AE:EC:FE:AD:2F:18:C6:88:68:30:FC:C8:E6
  SHA1: 8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
  SHA256:
 8E:CD:E6:88:4F:3D:87:B1:12:5B:A3:1A:C3:FC:B1:3D:70:16:DE:7F:57:CC:90:4F:E1:CB:97:C6:AE:98:19:6E

Alias name: affirmtrustpremium
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9A

Alias name: keynectisrootca
Certificate fingerprints:
  MD5:  CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26
  SHA1: 9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
  SHA256:
 42:10:F1:99:49:9A:9A:C3:3C:8D:E0:2B:A6:DB:AA:14:40:8B:DD:8A:6E:32:46:89:C1:92:2D:06:97:15:A3:32

Alias name: equifaxsecureglobalebusinessca1
Certificate fingerprints:
  MD5:  51:F0:2A:33:F1:F5:55:39:07:F2:16:7A:47:C7:5D:63
  SHA1: 3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
  SHA256:
 86:AB:5A:65:71:D3:32:9A:BC:D2:E4:E6:37:66:8B:A8:9C:73:1E:C2:93:B6:CB:A6:0F:71:63:40:A0:91:CE:AE

Alias name: affirmtrustpremiumca
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9A

Alias name: baltimorecodesigningca
Certificate fingerprints:
  MD5:  90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22
  SHA1: 30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
  SHA256:
 A9:15:45:DB:D2:E1:9C:4C:CD:F9:09:AA:71:90:0D:18:C7:35:1C:89:B3:15:F0:F1:3D:05:C1:3A:8F:FB:46:87

Alias name: gdcatrustauthr5root
Certificate fingerprints:
  MD5:  63:CC:D9:3D:34:35:5C:6F:53:A3:E2:08:70:48:1F:B4
  SHA1: 0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
  SHA256:
 BF:FF:8F:D0:44:33:48:7D:6A:8A:A6:0C:1A:29:76:7A:9F:C2:BB:B0:5E:42:0F:71:3A:13:B9:92:89:1D:38:93

Alias name: certinomisrootca
Certificate fingerprints:
  MD5:  14:0A:FD:8D:A8:28:B5:38:69:DB:56:7E:61:22:03:3F
  SHA1: 9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
  SHA256:
 2A:99:F5:BC:11:74:B7:3C:BB:1D:62:08:84:E0:1C:34:E5:1C:CB:39:78:DA:12:5F:0E:33:26:88:83:BF:41:58

Alias name: verisignclass3publicprimarycertificationauthorityg5
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:DF

Alias name: verisignclass3publicprimarycertificationauthorityg4
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:79

Alias name: verisignclass3publicprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:44

Alias name: swisssignsilverg2ca
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D5

Alias name: swisssignsilvercag2
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D5
```

```
Alias name: atostrustedroot2011
Certificate fingerprints:
  MD5:  AE:B9:C4:32:4B:AC:7F:5D:66:CC:77:94:BB:2A:77:56
  SHA1: 2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
  SHA256:
 F3:56:BE:A2:44:B7:A9:1E:B3:5D:53:CA:9A:D7:86:4A:CE:01:8E:2D:35:D5:F8:F9:6D:DF:68:A6:F4:1A:A4:74

Alias name: comodoecccertificationauthority
Certificate fingerprints:
  MD5:  7C:62:FF:74:9D:31:53:5E:68:4A:D5:78:AA:1E:BF:23
  SHA1: 9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
  SHA256:
 17:93:92:7A:06:14:54:97:89:AD:CE:2F:8F:34:F7:F0:B6:6D:0F:3A:E3:A3:B8:4D:21:EC:15:DB:BA:4F:AD:C7

Alias name: securetrustca
Certificate fingerprints:
  MD5:  DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
  SHA1: 87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
  SHA256:
 F1:C1:B5:0A:E5:A2:0D:D8:03:0E:C9:F6:BC:24:82:3D:D3:67:B5:25:57:59:B4:E7:1B:61:FC:E9:F7:37:5D:73

Alias name: soneraclass1ca
Certificate fingerprints:
  MD5:  33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
  SHA1: 07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
  SHA256:
 CD:80:82:84:CF:74:6F:F2:FD:6E:B5:8A:A1:D5:9C:4A:D4:B3:CA:56:FD:C6:27:4A:89:26:A7:83:5F:32:31:3D

Alias name: cadisigrootr2
Certificate fingerprints:
  MD5:  26:01:FB:D8:27:A7:17:9A:45:54:38:1A:43:01:3B:03
  SHA1: B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
  SHA256:
 E2:3D:4A:03:6D:7B:70:E9:F5:95:B1:42:20:79:D2:B9:1E:DF:BB:1F:B6:51:A0:63:3E:AA:8A:9D:C5:F8:07:03

Alias name: cadisigrootr1
Certificate fingerprints:
  MD5:  BE:EC:11:93:9A:F5:69:21:BC:D7:C1:C0:67:89:CC:2A
  SHA1: 8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
  SHA256:
 F9:6F:23:F4:C3:E7:9C:07:7A:46:98:8D:5A:F5:90:06:76:A0:F0:39:CB:64:5D:D1:75:49:B2:16:C8:24:40:CE

Alias name: verisignclass3g5ca
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:DF

Alias name: utnuserfirsthardwareca
Certificate fingerprints:
  MD5:  4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
  SHA1: 04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
  SHA256:
 6E:A5:47:41:D0:04:66:7E:ED:1B:48:16:63:4A:A3:A7:9E:6E:4B:96:95:0F:82:79:DA:FC:8D:9B:D8:81:21:37

Alias name: addtrustqualifiedca
Certificate fingerprints:
  MD5:  27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
  SHA1: 4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
  SHA256:
 80:95:21:08:05:DB:4B:BC:35:5E:44:28:D8:FD:6E:C2:CD:E3:AB:5F:B9:7A:99:42:98:8E:B8:F4:DC:D0:60:16

Alias name: verisignclass3g3ca
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:44

Alias name: thawtepersonalfreemailca
Certificate fingerprints:
  MD5:  53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65
  SHA1: E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
  SHA256:
 5B:38:BD:12:9E:83:D5:A0:CA:D2:39:21:08:94:90:D5:0D:4A:AE:37:04:28:F8:DD:FF:FF:FA:4C:15:64:E1:84

Alias name: certplusclass3pprimaryca
Certificate fingerprints:
  MD5:  E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB
  SHA1: 21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
  SHA256:
 CC:C8:94:89:37:1B:AD:11:1C:90:61:9B:EA:24:0A:2E:6D:AD:D9:9F:9F:6E:1D:4D:41:E5:8E:D6:DE:3D:02:85

Alias name: swisssigngoldg2ca
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:95

Alias name: swisssigngoldcag2
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:95

Alias name: dtrustrootclass3ca22009
Certificate fingerprints:
  MD5:  CD:E0:25:69:8D:47:AC:9C:89:35:90:F7:FD:51:3D:2F
  SHA1: 58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
  SHA256:
 49:E7:A4:42:AC:F0:EA:62:87:05:00:54:B5:25:64:B6:50:E4:F4:9E:42:E3:48:D6:AA:38:E0:39:E9:57:B1:C1

Alias name: acraizfnmtrcm
Certificate fingerprints:
  MD5:  E2:09:04:B4:D3:BD:D1:A0:14:FD:1A:D2:47:C4:57:1D
  SHA1: EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
  SHA256:
 EB:C5:57:0C:29:01:8C:4D:67:B1:AA:12:7B:AF:12:F7:03:B4:61:1E:BC:17:B7:DA:B5:57:38:94:17:9B:93:FA

Alias name: securitycommunicationevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:37

Alias name: starfieldclass2ca
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:58

Alias name: opentrustrootcag3
Certificate fingerprints:
  MD5:  21:37:B4:17:16:92:7B:67:46:70:A9:96:D7:A8:13:24
  SHA1: 6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
  SHA256:
 B7:C3:62:31:70:6E:81:07:8C:36:7C:B8:96:19:8F:1E:32:08:DD:92:69:49:DD:8F:57:09:A4:10:F7:5B:62:92

Alias name: opentrustrootcag2
Certificate fingerprints:
  MD5:  57:24:B6:59:24:6B:AE:C8:FE:1C:0C:20:F2:C0:4E:EB
  SHA1: 79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
  SHA256:
 27:99:58:29:FE:6A:75:15:C1:BF:E8:48:F9:C4:76:1D:B1:6C:22:59:29:25:7B:F4:0D:08:94:F2:9E:A8:BA:F2

Alias name: buypassclass2rootca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:48

Alias name: opentrustrootcag1
Certificate fingerprints:
  MD5:  76:00:CC:81:29:CD:55:5E:88:6A:7A:2E:F7:4D:39:DA
  SHA1: 79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
  SHA256:
 56:C7:71:28:D9:8C:18:D9:1B:4C:FD:FF:BC:25:EE:91:03:D4:75:8E:A2:AB:AD:82:6A:90:F3:45:7D:46:0E:B4

Alias name: globalsignr2ca
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9E

Alias name: buypassclass3rootca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4D

Alias name: ecacc
Certificate fingerprints:
  MD5:  EB:F5:9D:29:0D:61:F9:42:1F:7C:C2:BA:6D:E3:15:09
  SHA1: 28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
  SHA256:
 88:49:7F:01:60:2F:31:54:24:6A:E2:8C:4D:5A:EF:10:F1:D8:7E:BB:76:62:6F:4A:E0:B7:F9:5B:A7:96:87:99

Alias name: epkirootcertificationauthority
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D5

Alias name: verisignclass1g2ca
Certificate fingerprints:
  MD5:  DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
  SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
  SHA256:
 34:1D:E9:8B:13:92:AB:F7:F4:AB:90:A9:60:CF:25:D4:BD:6E:C6:5B:9A:51:CE:6E:D0:67:D0:0E:C7:CE:9B:7F

Alias name: certigna
Certificate fingerprints:
  MD5:  AB:57:A6:5B:7D:42:82:19:B5:D8:58:26:28:5E:FD:FF
  SHA1: B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
  SHA256:
 E3:B6:A2:DB:2E:D7:CE:48:84:2F:7A:C5:32:41:C7:B7:1D:54:14:4B:FB:40:C1:1F:3F:1D:0B:42:F5:EE:A1:2D
```

```
Alias name: camerfirmaglobalchambersignroot
Certificate fingerprints:
  MD5:  C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19
  SHA1: 33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
  SHA256:
 EF:3C:B4:17:FC:8E:BF:6F:97:87:6C:9E:4E:CE:39:DE:1E:A5:FE:64:91:41:D1:02:8B:7D:11:C0:B2:29:8C:ED

Alias name: cfcaevroot
Certificate fingerprints:
  MD5:  74:E1:B6:ED:26:7A:7A:44:30:33:94:AB:7B:27:81:30
  SHA1: E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
  SHA256:
 5C:C3:D7:8E:4E:1D:5E:45:54:7A:04:E6:87:3E:64:F9:0C:F9:53:6D:1C:CC:2E:F8:00:F3:55:C4:C5:FD:70:FD

Alias name: soneraclass2rootca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:27

Alias name: certumtrustednetworkca
Certificate fingerprints:
  MD5:  D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
  SHA1: 07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
  SHA256:
 5C:58:46:8D:55:F5:8E:49:7E:74:39:82:D2:B5:00:10:B6:D1:65:37:4A:CF:83:A7:D4:A3:2D:B7:68:C4:40:8E

Alias name: securitycommunicationrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F6

Alias name: globalsigneccrootcar5
Certificate fingerprints:
  MD5:  9F:AD:3B:1C:02:1E:8A:BA:17:74:38:81:0C:A2:BC:08
  SHA1: 1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
  SHA256:
 17:9F:BC:14:8A:3D:D0:0F:D2:4E:A1:34:58:CC:43:BF:A7:F5:9C:81:82:D7:83:A5:13:F6:EB:EC:10:0C:89:24

Alias name: globalsigneccrootcar4
Certificate fingerprints:
  MD5:  20:F0:27:68:D1:7E:A0:9D:0E:E6:2A:CA:DF:5C:89:8E
  SHA1: 69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
  SHA256:
 BE:C9:49:11:C2:95:56:76:DB:6C:0A:55:09:86:D7:6E:3B:A0:05:66:7C:44:2C:97:62:B4:FB:B7:73:DE:22:8C

Alias name: chambersofcommerceroot2008
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C0

Alias name: pscprocert
Certificate fingerprints:
  MD5:  E6:24:E9:12:01:AE:0C:DE:8E:85:C4:CE:A3:12:DD:EC
  SHA1: 70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
  SHA256:
 3C:FC:3C:14:D1:F6:84:FF:17:E3:8C:43:CA:44:0C:00:B9:67:EC:93:3E:8B:FE:06:4C:A1:D7:2C:90:F2:AD:B0

Alias name: thawteprimaryrootcag3
Certificate fingerprints:
  MD5:  FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
  SHA1: F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
  SHA256:
 4B:03:F4:58:07:AD:70:F2:1B:FC:2C:AE:71:C9:FD:E4:60:4C:06:4C:F5:FF:B6:86:BA:E5:DB:AA:D7:FD:D3:4C

Alias name: quovadisrootca
Certificate fingerprints:
  MD5:  27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
  SHA1: DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
  SHA256:
 A4:5E:DE:3B:BB:F0:9C:8A:E1:5C:72:EF:C0:72:68:D6:93:A2:1C:99:6F:D5:1E:67:CA:07:94:60:FD:6D:88:73

Alias name: thawteprimaryrootcag2
Certificate fingerprints:
  MD5:  74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
  SHA1: AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
  SHA256:
 A4:31:0D:50:AF:18:A6:44:71:90:37:2A:86:AF:AF:8B:95:1F:FB:43:1D:83:7F:1E:56:88:B4:59:71:ED:15:57

Alias name: deprecateditsecca
Certificate fingerprints:
  MD5:  A5:96:0C:F6:B5:AB:27:E5:01:C6:00:88:9E:60:33:E5
  SHA1: 12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
  SHA256:
 9A:59:DA:86:24:1A:FD:BA:A3:39:FA:9C:FD:21:6A:0B:06:69:4D:E3:7E:37:52:6B:BE:63:C8:BC:83:74:2E:CB

Alias name: usertrustrsacertificationauthority
Certificate fingerprints:
  MD5:  1B:FE:69:D1:91:B7:19:33:A3:72:A8:0F:E1:55:E5:B5
  SHA1: 2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
  SHA256:
 E7:93:C9:B0:2F:D8:AA:13:E2:1C:31:22:8A:CC:B0:81:19:64:3B:74:9C:89:89:64:B1:74:6D:46:C3:D4:CB:D2

Alias name: entrustrootcag2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:39

Alias name: networksolutionscertificateauthority
Certificate fingerprints:
  MD5:  D3:F3:A6:16:C0:FA:6B:1D:59:B1:2D:96:4D:0E:11:2E
  SHA1: 74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
  SHA256:
 15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:C0:97:F4:01:64:B2:F8:59:8A:BD:83:86:0C

Alias name: trustcenterclass4caii
Certificate fingerprints:
  MD5:  9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0
  SHA1: A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
  SHA256:
 32:66:96:7E:59:CD:68:00:8D:9D:D3:20:81:11:85:C7:04:20:5E:8D:95:FD:D8:4F:1C:7B:31:1E:67:04:FC:32

Alias name: oistewisekeyglobalrootgaca
Certificate fingerprints:
  MD5:  BC:6C:51:33:A7:E9:D3:66:63:54:15:72:1B:21:92:93
  SHA1: 59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
  SHA256:
 41:C9:23:86:6A:B4:CA:D6:B7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:D7:F5

Alias name: verisignuniversalrootcertificationauthority
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3C
```

```

Alias name: ttelesecglobalrootclass3ca
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:BD

Alias name: starfieldservicesrootg2ca
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B5

Alias name: addtrustexternalroot
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F2

Alias name: turktrustelektroniksertifikahizmetsaglayicisih5
Certificate fingerprints:
  MD5:  DA:70:8E:F0:22:DF:93:26:F6:5F:9F:D3:15:06:52:4E
  SHA1: C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
  SHA256:
 49:35:1B:90:34:44:C1:85:CC:DC:5C:69:3D:24:D8:55:5C:B2:08:D6:A8:14:13:07:69:9F:4A:F0:63:19:9D:78

Alias name: camerfirmachambersca
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C0

Alias name: certsignrootca
Certificate fingerprints:
  MD5:  18:98:C0:D6:E9:3A:FC:F9:B0:F5:0C:F7:4B:01:44:17
  SHA1: FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
  SHA256:
 EA:A9:62:C4:FA:4A:6B:AF:EB:E4:15:19:6D:35:1C:CD:88:8D:4F:53:F3:FA:8A:E6:D7:C4:66:A9:4E:60:42:BB

Alias name: verisignuniversalrootca
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3C

Alias name: geotrustuniversalca
Certificate fingerprints:
  MD5:  92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
  SHA1: E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
  SHA256:
 A0:45:9B:9F:63:B2:25:59:F5:FA:5D:4C:6D:B3:F9:F7:2F:F1:93:42:03:35:78:F0:73:BF:1D:1B:46:CB:B9:12

Alias name: luxtrustglobalroot2
Certificate fingerprints:
  MD5:  B2:E1:09:00:61:AF:F7:F1:91:6F:C4:AD:8D:5E:3B:7C
  SHA1: 1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
  SHA256:
 54:45:5F:71:29:C2:0B:14:47:C4:18:F9:97:16:8F:24:C5:8F:C5:02:3B:F5:DA:5B:E2:EB:6E:1D:D8:90:2E:D5

Alias name: twcaglobalrootca
Certificate fingerprints:
  MD5:  F9:03:7E:CF:E6:9E:3C:73:7A:2A:90:07:69:FF:2B:96
  SHA1: 9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
  SHA256:
 59:76:90:07:F7:68:5D:0F:CD:50:87:2F:9F:95:D5:75:5A:5B:2B:45:7D:81:F3:69:2B:61:0A:98:67:2F:0E:1B

Alias name: tubitakkamusmsslkoksertifikasisurum1
Certificate fingerprints:
  MD5:  DC:00:81:DC:69:2F:3E:2F:B0:3B:F6:3D:5A:91:8E:49
  SHA1: 31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
  SHA256:
 46:ED:C3:68:90:46:D5:3A:45:3F:B3:10:4A:B8:0D:CA:EC:65:8B:26:60:EA:16:29:DD:7E:86:79:90:64:87:16

Alias name: affirmtrustnetworkingca
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1B

Alias name: affirmtrustcommercialca
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A7

Alias name: godaddyrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:DA

Alias name: starfieldrootg2ca
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F5

Alias name: dtrustrootclass3ca2ev2009
Certificate fingerprints:
  MD5:  AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  SHA256:
 EE:C5:49:6B:98:8C:E9:86:25:B9:34:09:2E:EC:29:08:BE:D0:B0:F3:16:C2:D4:73:0C:84:EA:F1:F3:D3:48:81

Alias name: buypassclass3ca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4D

Alias name: verisignclass2g3ca
Certificate fingerprints:
  MD5:  F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
  SHA1: 61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
  SHA256:
 92:A9:D9:83:3F:E1:94:4D:B3:66:E8:BF:AE:7A:95:B6:48:0C:2D:6C:6C:2A:1B:E6:5D:42:36:B6:08:FC:A1:BB

Alias name: digicerttrustedrootg4
Certificate fingerprints:
  MD5:  78:F2:FC:AA:60:1F:2F:B4:EB:C9:37:BA:53:2E:75:49
  SHA1: DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
  SHA256:
 55:2F:7B:DC:F1:A7:AF:9E:6C:E6:72:01:7F:4F:12:AB:F7:72:40:C7:8E:76:1A:C2:03:D1:D9:D2:0A:C8:99:88

Alias name: quovadisrootca2g3
Certificate fingerprints:
  MD5:  AF:0C:86:6E:BF:40:2D:7F:0B:3E:12:50:BA:12:3D:06
  SHA1: 09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
  SHA256:
 8F:E4:FB:0A:F9:3A:4D:0D:67:DB:0B:EB:B2:3E:37:C7:1B:F3:25:DC:BC:DD:24:0E:A0:4D:AF:58:B4:7E:18:40

Alias name: geotrustprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D4

Alias name: geotrustprimarycertificationauthorityg2
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:66

Alias name: godaddyclass2ca
Certificate fingerprints:
  MD5:  91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
  SHA256:
 C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E4

Alias name: trustcoreca1
Certificate fingerprints:
  MD5:  27:92:23:1D:0A:F5:40:7C:E9:E6:6B:9D:D8:F5:E7:6C
  SHA1: 58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
  SHA256:
 5A:88:5D:B1:9C:01:D9:12:C5:75:93:88:93:8C:AF:BB:DF:03:1A:B2:D4:8E:91:EE:15:58:9B:42:97:1D:03:9C

Alias name: hellenicacademicandresearchinstitutionseccrootca2015
Certificate fingerprints:
  MD5:  81:E5:B4:17:EB:C2:F5:E1:4B:0D:41:7B:49:92:FE:EF
  SHA1: 9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
  SHA256:
 44:B5:45:AA:8A:25:E6:5A:73:CA:15:DC:27:FC:36:D2:4C:1C:B9:95:3A:06:65:39:B1:15:82:DC:48:7B:48:33

Alias name: utnuserfirstobjectca
Certificate fingerprints:
  MD5:  A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9
  SHA1: E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
  SHA256:
 6F:FF:78:E4:00:A7:0C:11:01:1C:D8:59:77:C4:59:FB:5A:F9:6A:3D:F0:54:08:20:D0:F4:B8:60:78:75:E5:8F

Alias name: ttelesecglobalrootclass3
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:BD

Alias name: ttelesecglobalrootclass2
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:52
```

```
Alias name: addtrustclass1ca
Certificate fingerprints:
  MD5:  1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC
  SHA1: CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
  SHA256:
 8C:72:09:27:9A:C0:4E:27:5E:16:D0:7F:D3:B7:75:E8:01:54:B5:96:80:46:E3:1F:52:DD:25:76:63:24:E9:A7

Alias name: amzninternalrootca
Certificate fingerprints:
  MD5:  08:09:73:AC:E0:78:41:7C:0A:26:33:51:E8:CF:E6:60
  SHA1: A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
  SHA256:
 0E:DE:63:C1:DC:7A:8E:11:F1:AB:BC:05:4F:59:EE:49:9D:62:9A:2F:DE:9C:A7:16:32:A2:64:29:3E:8B:66:AA

Alias name: starfieldrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F5

Alias name: camerfirmachambersignca
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:CA

Alias name: secomscrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F6

Alias name: entrustevca
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4C

Alias name: secomscrootca1
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6C

Alias name: affirmtrustcommercial
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A7

Alias name: digicertassuredidrootg3
Certificate fingerprints:
  MD5:  7C:7F:65:31:0C:81:DF:8D:BA:3E:99:E2:5C:AD:6E:FB
  SHA1: F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
  SHA256:
 7E:37:CB:8B:4C:47:09:0C:AB:36:55:1B:A6:F4:5D:B8:40:68:0F:BA:16:6A:95:2D:B1:00:71:7F:43:05:3F:C2

Alias name: affirmtrustnetworking
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1B

Alias name: izenpecom
Certificate fingerprints:
  MD5:  A6:B0:CD:85:80:DA:5C:50:34:A3:39:90:2F:55:67:73
  SHA1: 2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
  SHA256:
 25:30:CC:8E:98:32:15:02:BA:D9:6F:9B:1F:BA:1B:09:9E:2D:29:9E:0F:45:48:BB:91:4F:36:3B:C0:D4:53:1F

Alias name: amazon-ca-g4-legacy
Certificate fingerprints:
  MD5:  6C:E5:BD:67:A4:4F:E3:FD:C2:4C:46:E6:06:5B:6D:55
  SHA1: EA:E7:DE:F9:0A:BE:9F:0B:68:CE:B7:24:0D:80:74:03:BF:6E:B1:6E
  SHA256:
 CD:72:C4:7F:B4:AD:28:A4:67:2B:E1:86:47:D4:40:E9:3B:16:2D:95:DB:3C:2F:94:BB:81:D9:09:F7:91:24:5E

Alias name: digicertassuredidrootg2
Certificate fingerprints:
  MD5:  92:38:B9:F8:63:24:82:65:2C:57:33:E6:FE:81:8F:9D
  SHA1: A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
  SHA256:
 7D:05:EB:B6:82:33:9F:8C:94:51:EE:09:4E:EB:FE:FA:79:53:A1:14:ED:B2:F4:49:49:45:2F:AB:7D:2F:C1:85

Alias name: comodoaaaservicesroot
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F4

Alias name: entrustnetpremium2048secureserverca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:77

Alias name: trustcorrootcertca2
Certificate fingerprints:
  MD5:  A2:E1:F8:18:0B:BA:45:D5:C7:41:2A:BB:37:52:45:64
  SHA1: B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
  SHA256:
 07:53:E9:40:37:8C:1B:D5:E3:83:6E:39:5D:AE:A5:CB:83:9E:50:46:F1:BD:0E:AE:19:51:CF:10:FE:C7:C9:65

Alias name: entrust2048ca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:77

Alias name: trustcorrootcertca1
Certificate fingerprints:
  MD5:  6E:85:F1:DC:1A:00:D3:22:D5:B2:B2:AC:6B:37:05:45
  SHA1: FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
  SHA256:
 D4:0E:9C:86:CD:8F:E4:68:C1:77:69:59:F4:9E:A7:74:FA:54:86:84:B6:C4:06:F3:90:92:61:F4:DC:E2:57:5C

Alias name: baltimorecybertrustroot
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:EB
```

```
Alias name: eecertificationcentrerootca
Certificate fingerprints:
  MD5:  43:5E:88:D4:7D:1A:4A:7E:FD:84:2E:52:EB:01:D4:6F
  SHA1: C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
  SHA256:
 3E:84:BA:43:42:90:85:16:E7:75:73:C0:99:2F:09:79:CA:08:4E:46:85:68:1F:F1:95:CC:BA:8A:22:9B:8A:76

Alias name: dstacescax6
Certificate fingerprints:
  MD5:  21:D8:4C:82:2B:99:09:33:A2:EB:14:24:8D:8E:5F:E8
  SHA1: 40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
  SHA256:
 76:7C:95:5A:76:41:2C:89:AF:68:8E:90:A1:C7:0F:55:6C:FD:6B:60:25:DB:EA:10:41:6D:7E:B6:83:1F:8C:40

Alias name: comodocertificationauthority
Certificate fingerprints:
  MD5:  5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  SHA256:
 0C:2C:D6:3D:F7:80:6F:A3:99:ED:E8:09:11:6B:57:5B:F8:79:89:F0:65:18:F9:80:8C:86:05:03:17:8B:AF:66

Alias name: thawteserverca
Certificate fingerprints:
  MD5:  EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2
  SHA1: 9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
  SHA256:
 87:C6:78:BF:B8:B2:5F:38:F7:E9:7B:33:69:56:BB:CF:14:4B:BA:CA:A5:36:47:E6:1A:23:25:BC:10:55:31:6B

Alias name: secomvalicertclass1ca
Certificate fingerprints:
  MD5:  65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
  SHA1: E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
  SHA256:
 F4:C1:49:55:1A:30:13:A3:5B:C7:BF:FE:17:A7:F3:44:9B:C1:AB:5B:5A:0A:E7:4B:06:C2:3B:90:00:4C:01:04

Alias name: godaddyrootg2ca
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:DA

Alias name: globalchambersignroot2008
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:CA

Alias name: equifaxsecureebusinessca1
Certificate fingerprints:
  MD5:  14:C0:08:E5:A3:85:03:A3:BE:78:E9:67:4F:27:CA:EE
  SHA1: AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
  SHA256:
 2E:3A:2B:B5:11:25:05:83:6C:A8:96:8B:E2:CB:37:27:CE:9B:56:84:5C:6E:E9:8E:91:85:10:4A:FB:9A:F5:96

Alias name: quovadisrootca3
Certificate fingerprints:
  MD5:  31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
  SHA1: 1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
  SHA256:
 18:F1:FC:7F:20:5D:F8:AD:DD:EB:7F:E0:07:DD:57:E3:AF:37:5A:9C:4D:8D:73:54:6B:F4:F1:FE:D1:E1:8D:35

Alias name: usertrustecccertificationauthority
Certificate fingerprints:
  MD5:  FA:68:BC:D9:B5:7F:AD:FD:C9:1D:06:83:28:CC:24:C1
  SHA1: D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
  SHA256:
 4F:F4:60:D5:4B:9C:86:DA:BF:BC:FC:57:12:E0:40:0D:2B:ED:3F:BC:4D:4F:BD:AA:86:E0:6A:DC:D2:A9:AD:7A

Alias name: quovadisrootca2
Certificate fingerprints:
  MD5:  5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B
  SHA1: CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
  SHA256:
 85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D:C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:86

Alias name: soneraclass2ca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:27

Alias name: twcarootcertificationauthority
Certificate fingerprints:
  MD5:  AA:08:8F:F6:F9:7B:B7:F2:B1:A7:1E:9B:EA:EA:BD:79
  SHA1: CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
  SHA256:
 BF:D8:8F:E1:10:1C:41:AE:3E:80:1B:F8:BE:56:35:0E:E9:BA:D1:A6:B9:BD:51:5E:DC:5C:6D:5B:87:11:AC:44

Alias name: baltimorecybertrustca
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:EB

Alias name: cia-crt-g3-01-ca
Certificate fingerprints:
  MD5:  E3:66:DD:D6:A0:D5:40:8F:FF:29:E2:C0:CB:6E:62:1A
  SHA1: 2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
  SHA256:
 20:48:AD:4C:EC:90:7F:FA:4A:15:D4:CE:45:E3:C8:E4:2C:EA:78:33:DC:C7:D3:40:48:FC:60:47:27:42:99:EC

Alias name: entrustrootcertificationauthorityg2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:39

Alias name: verisignclass3g4ca
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:79

Alias name: xrampglobalcaroot
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A2

Alias name: identrustcommercialrootca1
Certificate fingerprints:
  MD5:  B3:3E:77:73:75:EE:A0:D3:E3:7E:49:63:49:59:BB:C7
  SHA1: DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
  SHA256:
 5D:56:49:9B:E4:D2:E0:8B:CF:CA:D0:8A:3E:38:72:3D:50:50:3B:DE:70:69:48:E4:2F:55:60:30:19:E5:28:AE

Alias name: camerfirmachamberscommerceca
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C3

Alias name: verisignclass3g2ca
Certificate fingerprints:
  MD5:  A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  SHA256:
 83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8B

Alias name: deutschetelekomrootca2
Certificate fingerprints:
  MD5:  74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08
  SHA1: 85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
  SHA256:
 B6:19:1A:50:D0:C3:97:7F:7D:A9:9B:CD:AA:C8:6A:22:7D:AE:B9:67:9E:C7:0B:A3:B0:C9:D9:22:71:C1:70:D3

Alias name: certumca
Certificate fingerprints:
  MD5:  2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
  SHA1: 62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
  SHA256:
 D8:E0:FE:BC:1D:B2:E3:8D:00:94:0F:37:D2:7D:41:34:4D:99:3E:73:4B:99:D5:65:6D:97:78:D4:D8:14:36:24

Alias name: cybertrustglobalroot
Certificate fingerprints:
  MD5:  72:E4:4A:87:E3:69:40:80:77:EA:BC:E3:F4:FF:F0:E1
  SHA1: 5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
  SHA256:
 96:0A:DF:00:63:E9:63:56:75:0C:29:65:DD:0A:08:67:DA:0B:9C:BD:6E:77:71:4A:EA:FB:23:49:AB:39:3D:A3

Alias name: globalsignrootca
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:99

Alias name: secomevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:37

Alias name: globalsignr3ca
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3B

Alias name: staatdernederlandenrootcag3
Certificate fingerprints:
  MD5:  0B:46:67:07:DB:10:2F:19:8C:35:50:60:D1:0B:F4:37
  SHA1: D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
  SHA256:
 3C:4F:B0:B9:5A:B8:B3:00:32:F4:32:B8:6F:53:5F:E1:72:C1:85:D0:FD:39:86:58:37:CF:36:18:7F:A6:F4:28
```

```
Alias name: staatdernederlandenrootcag2
Certificate fingerprints:
  MD5:  7C:A5:0F:F8:5B:9A:7D:6D:30:AE:54:5A:E3:42:A2:8A
  SHA1: 59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
  SHA256:
 66:8C:83:94:7D:A6:3B:72:4B:EC:E1:74:3C:31:A0:E6:AE:D0:DB:8E:C5:B3:1B:E3:77:BB:78:4F:91:B6:71:6F

Alias name: aolrootca2
Certificate fingerprints:
  MD5:  D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
  SHA1: 85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
  SHA256:
 7D:3B:46:5A:60:14:E5:26:C0:AF:FC:EE:21:27:D2:31:17:27:AD:81:1C:26:84:2D:00:6A:F3:73:06:CC:80:BD

Alias name: dstrootcax3
Certificate fingerprints:
  MD5:  41:03:52:DC:0F:F7:50:1B:16:F0:02:8E:BA:6F:45:C5
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
  SHA256:
 06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:39

Alias name: trustcenteruniversalcai
Certificate fingerprints:
  MD5:  45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C
  SHA1: 6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
  SHA256:
 EB:F3:C0:2A:87:89:B1:FB:7D:51:19:95:D6:63:B7:29:06:D9:13:CE:0D:5E:10:56:8A:8A:77:E2:58:61:67:E7

Alias name: aolrootca1
Certificate fingerprints:
  MD5:  14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
  SHA1: 39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
  SHA256:
 77:40:73:12:C6:3A:15:3D:5B:C0:0B:4E:51:75:9C:DF:DA:C2:37:DC:2A:33:B6:79:46:E9:8E:9B:FA:68:0A:E3

Alias name: affirmtrustpremiumecc
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:23

Alias name: microseceszignorootca2009
Certificate fingerprints:
  MD5:  F8:49:F4:03:BC:44:2D:83:BE:48:69:7D:29:64:FC:B1
  SHA1: 89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
  SHA256:
 3C:5F:81:FE:A5:FA:B8:2C:64:BF:A2:EA:EC:AF:CD:E8:E0:77:FC:86:20:A7:CA:E5:37:16:3D:F3:6E:DB:F3:78

Alias name: verisignclass1g3ca
Certificate fingerprints:
  MD5:  B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
  SHA1: 20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
  SHA256:
 CB:B5:AF:18:5E:94:2A:24:02:F9:EA:CB:C0:ED:5B:B8:76:EE:A3:C1:22:36:23:D0:04:47:E4:F3:BA:55:4B:65

Alias name: certplusrootcag2
Certificate fingerprints:
  MD5:  A7:EE:C4:78:2D:1B:EE:2D:B9:29:CE:D6:A7:96:32:31
  SHA1: 4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
  SHA256:
 6C:C0:50:41:E6:44:5E:74:69:6C:4C:FB:C9:F8:0F:54:3B:7E:AB:BB:44:B4:CE:6F:78:7C:6A:99:71:C4:2F:17

Alias name: certplusrootcag1
Certificate fingerprints:
  MD5:  7F:09:9C:F7:D9:B9:5C:69:69:56:D5:37:3E:14:0D:42
  SHA1: 22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
  SHA256:
 15:2A:40:2B:FC:DF:2C:D5:48:05:4D:22:75:B3:9C:7F:CA:3E:C0:97:80:78:B0:F0:EA:76:E5:61:A6:C7:43:3E

Alias name: addtrustexternalca
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F2

Alias name: entrustrootcertificationauthority
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4C

Alias name: verisignclass3ca
Certificate fingerprints:
  MD5:  EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  SHA256:
 A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:05

Alias name: digicertassuredidrootca
Certificate fingerprints:
  MD5:  87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
  SHA1: 05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
  SHA256:
 3E:90:99:B5:01:5E:8F:48:6C:00:BC:EA:9D:11:1E:E7:21:FA:BA:35:5A:89:BC:F1:DF:69:56:1E:3D:C6:32:5C

Alias name: globalsignrootcar3
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3B

Alias name: globalsignrootcar2
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9E

Alias name: verisignclass1ca
Certificate fingerprints:
  MD5:  86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E
  SHA1: CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
  SHA256:
 51:84:7C:8C:BD:2E:9A:72:C9:1E:29:2D:2A:E2:47:D7:DE:1E:3F:D2:70:54:7A:20:EF:7D:61:0F:38:B8:84:2C

Alias name: thawtepremiumserverca
Certificate fingerprints:
  MD5:  A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46
  SHA1: E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
  SHA256:
 3F:9F:27:D5:83:20:4B:9E:09:C8:A3:D2:06:6C:4B:57:D3:A2:47:9C:36:93:65:08:80:50:56:98:10:5D:BC:E9

Alias name: verisigntsaca
Certificate fingerprints:
  MD5:  F2:89:95:6E:4D:05:F0:F1:A7:21:55:7D:46:11:BA:47
  SHA1: 20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
  SHA256:
 CB:6B:05:D9:E8:E5:7C:D8:82:B1:0B:4D:B7:0D:E4:BB:1D:E4:2B:A4:8A:7B:D0:31:8B:63:5B:F6:E7:78:1A:9D
```

```
Alias name: thawteprimaryrootca
Certificate fingerprints:
  MD5:  8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
  SHA1: 91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
  SHA256:
 8D:72:2F:81:A9:C1:13:C0:79:1D:F1:36:A2:96:6D:B2:6C:95:0A:97:1D:B4:6B:41:99:F4:EA:54:B7:8B:FB:9F

Alias name: visaecommerceroot
Certificate fingerprints:
  MD5:  FC:11:B8:D8:08:93:30:00:6D:23:F9:7E:EB:52:1E:02
  SHA1: 70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
  SHA256:
 69:FA:C9:BD:55:FB:0A:C7:8D:53:BB:EE:5C:F1:D5:97:98:9F:D0:AA:AB:20:A2:51:51:BD:F1:73:3E:E7:D1:22

Alias name: digicertglobalrootg3
Certificate fingerprints:
  MD5:  F5:5D:A4:50:A5:FB:28:7E:1E:0F:0D:CC:96:57:56:CA
  SHA1: 7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
  SHA256:
 31:AD:66:48:F8:10:41:38:C7:38:F3:9E:A4:32:01:33:39:3E:3A:18:CC:02:29:6E:F9:7C:2A:C9:EF:67:31:D0

Alias name: xrampglobalca
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A2

Alias name: digicertglobalrootg2
Certificate fingerprints:
  MD5:  E4:A6:8A:C8:54:AC:52:42:46:0A:FD:72:48:1B:2A:44
  SHA1: DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
  SHA256:
 CB:3C:CB:B7:60:31:E5:E0:13:8F:8D:D3:9A:23:F9:DE:47:FF:C3:5E:43:C1:14:4C:EA:27:D4:6A:5A:B1:CB:5F

Alias name: valicertclass2ca
Certificate fingerprints:
  MD5:  A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
  SHA1: 31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
  SHA256:
 58:D0:17:27:9C:D4:DC:63:AB:DD:B1:96:A6:C9:90:6C:30:C4:E0:87:83:EA:E8:C1:60:99:54:D6:93:55:59:6B

Alias name: geotrustprimaryca
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6C

Alias name: netlockaranyclassgoldfotanusitvany
Certificate fingerprints:
  MD5:  C5:A1:B7:FF:73:DD:D6:D7:34:32:18:DF:FC:3C:AD:88
  SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
  SHA256:
 6C:61:DA:C3:A2:DE:F0:31:50:6B:E0:36:D2:A6:FE:40:19:94:FB:D1:3D:F9:C8:D4:66:59:92:74:C4:46:EC:98

Alias name: geotrustglobalca
Certificate fingerprints:
  MD5:  F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
  SHA1: DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
  SHA256:
 FF:85:6A:2D:25:1D:CD:88:D3:66:56:F4:50:12:67:98:CF:AB:AA:DE:40:79:9C:72:2D:E4:D2:B5:DB:36:A7:3A

Alias name: oistewisekeyglobalrootgbca
Certificate fingerprints:
  MD5:  A4:EB:B9:61:28:2E:B7:2F:98:B0:35:26:90:99:51:1D
  SHA1: 0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
  SHA256:
 6B:9C:08:E8:6E:B0:F7:67:CF:AD:65:CD:98:B6:21:49:E5:49:4A:67:F5:84:5E:7B:D1:ED:01:9F:27:B8:6B:D6

Alias name: certumtrustednetworkca2
Certificate fingerprints:
  MD5:  6D:46:9E:D9:25:6D:08:23:5B:5E:74:7D:1E:27:DB:F2
  SHA1: D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
  SHA256:
 B6:76:F2:ED:DA:E8:77:5C:D3:6C:B0:F6:3C:D1:D4:60:39:61:F4:9E:62:65:BA:01:3A:2F:03:07:B6:D0:B8:04

Alias name: starfieldservicesrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B5

Alias name: comodorsacertificationauthority
Certificate fingerprints:
  MD5:  1B:31:B0:71:40:36:CC:14:36:91:AD:C4:3E:FD:EC:18
  SHA1: AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
  SHA256:
 52:F0:E1:C4:E5:8E:C6:29:29:1B:60:31:7F:07:46:71:B8:5D:7E:A8:0D:5B:07:27:34:63:53:4B:32:B4:02:34

Alias name: comodoaaaca
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F4

Alias name: identrustpublicsectorrootca1
Certificate fingerprints:
  MD5:  37:06:A5:B0:FC:89:9D:BA:F4:6B:8C:1A:64:CD:D5:BA
  SHA1: BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
  SHA256:
 30:D0:89:5A:9A:44:8A:26:20:91:63:55:22:D1:F5:20:10:B5:86:7A:CA:E1:2C:78:EF:95:8F:D4:F4:38:9F:2F

Alias name: certplusclass2primaryca
Certificate fingerprints:
  MD5:  88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
  SHA1: 74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
  SHA256:
 0F:99:3C:8A:EF:97:BA:AF:56:87:14:0E:D5:9A:D1:82:1B:B4:AF:AC:F0:AA:9A:58:B5:D5:7A:33:8A:3A:FB:CB

Alias name: ttelesecglobalrootclass2ca
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:52

Alias name: accvraiz1
Certificate fingerprints:
  MD5:  D0:A0:5A:EE:05:B6:09:94:21:A1:7D:F1:B2:29:82:02
  SHA1: 93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
  SHA256:
 9A:6E:C0:12:E1:A7:DA:9D:BE:34:19:4D:47:8A:D7:C0:DB:18:22:FB:07:1D:F1:29:81:49:6E:D1:04:38:41:13

Alias name: digicerthighassuranceevrootca
Certificate fingerprints:
  MD5:  D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
  SHA1: 5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
  SHA256:
 74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:CF

Alias name: amzninternalinfoseccag3
Certificate fingerprints:
  MD5:  E9:34:94:02:BA:BB:31:6B:22:E6:2B:A9:C4:F0:26:04
  SHA1: B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
  SHA256:
 81:03:0B:C7:E2:54:DA:7B:F8:B7:45:DB:DD:41:15:89:B5:A3:81:86:FB:4B:29:77:1F:84:0A:18:D9:67:6D:68

Alias name: cia-crt-g3-02-ca
Certificate fingerprints:
  MD5:  FD:B9:23:FD:D3:EB:2D:3E:57:EF:56:FF:DB:D3:E4:B9
  SHA1: 96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
  SHA256:
 93:F1:72:FB:BA:43:31:5C:06:EE:0F:9F:04:89:B8:F6:88:BC:75:15:3C:BE:B4:80:AC:A7:14:3A:F6:FC:4A:C1

Alias name: entrustrootcertificationauthorityec1
Certificate fingerprints:
  MD5:  B6:7E:1D:F0:58:C5:49:6C:24:3B:3D:ED:98:18:ED:BC
  SHA1: 20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
  SHA256:
 02:ED:0E:B2:8C:14:DA:45:16:5C:56:67:91:70:0D:64:51:D7:FB:56:F0:B2:AB:1D:3B:8E:B0:70:E5:6E:DF:F5

Alias name: securitycommunicationrootca
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6C

Alias name: globalsignca
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:99

Alias name: trustcenterclass2caii
Certificate fingerprints:
  MD5:  CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
  SHA1: AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
  SHA256:
 E6:B8:F8:76:64:85:F8:07:AE:7F:8D:AC:16:70:46:1F:07:C0:A1:3E:EF:3A:1F:F7:17:53:8D:7A:BA:D3:91:B4

Alias name: camerfirmachambersofcommerceroot
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C3

Alias name: geotrustprimarycag3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D4

Alias name: geotrustprimarycag2
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:66
```

```
Alias name: hongkongpostrootca1
Certificate fingerprints:
  MD5:  A8:0D:6F:39:78:B9:43:6D:77:42:6D:98:5A:CC:23:CA
  SHA1: D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
  SHA256:
 F9:E6:7D:33:6C:51:00:2A:C0:54:C6:32:02:2D:66:DD:A2:E7:E3:FF:F1:0A:D0:61:ED:31:D8:BB:B4:10:CF:B2

Alias name: affirmtrustpremiumeccca
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:23

Alias name: hellenicacademicandresearchinstitutionsrootca2015
Certificate fingerprints:
  MD5:  CA:FF:E2:DB:03:D9:CB:4B:E9:0F:AD:84:FD:7B:18:CE
  SHA1: 01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
  SHA256:
 A0:40:92:9A:02:CE:53:B4:AC:F4:F2:FF:C6:98:1C:E4:49:6F:75:5E:6D:45:FE:0B:2A:69:2B:CD:52:52:3F:36
```

Reducing messaging costs with basic ingest

Basic Ingest enables you to securely send device data to the AWS services supported by AWS IoT rule actions (p. 359) without incurring messaging costs. Basic Ingest optimizes data flow by removing the publish/subscribe message broker from the ingestion path, so it's more cost effective.

To use Basic Ingest, you send messages from your devices or applications with topic names that start with `$aws/rules/rule-name` as their first three levels, where `rule-name` is the name of your AWS IoT rule to trigger.

You can use an existing rule with Basic Ingest simply by adding the Basic Ingest prefix (`$aws/rules/rule-name`) to the message topic by which you normally trigger the rule. For example, if you have a rule named `BuildingManager` that is triggered by messages with topics like `Buildings/Building5/Floor2/Room201/Lights` (`"sql": "SELECT * FROM 'Buildings/#'"`), you can trigger the same rule with Basic Ingest by sending a message with topic `$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`.

Be aware that:

- Your devices and rules cannot subscribe to Basic Ingest reserved topics. For more information, see Reserved topics (p. 87).
- If you need a publish/subscribe broker to distribute messages to multiple subscribers (for example, to deliver messages to other devices and the rules engine), you should continue to use the AWS IoT message broker to handle the message distribution. Just publish your messages on topics other than Basic Ingest topics.

Using basic ingest

Make sure your device or application is using a policy (p. 235) that has publish permissions on `$aws/rules/*`. Or you can specify permission for individual rules with `$aws/rules/rule-name/*` in the policy. Otherwise, your devices and applications can continue to use their existing connections with AWS IoT Core.

When the message reaches the rules engine, there is no difference in execution or error handling between rules triggered from Basic Ingest and those triggered through message broker subscriptions.

You can, of course, create rules for use with Basic Ingest. Keep in mind the following:

- The initial prefix of a Basic Ingest topic (`$aws/rules/`*`rule-name`*) isn't available to the topic(Decimal) (p. 479) function.
- If you define a rule that is triggered only with Basic Ingest, the `FROM` clause is optional in the `sql` field of the `rule` definition. It's still required if the rule is also triggered by other messages that must be sent through the message broker (for example, because those other messages must be distributed to multiple subscribers). For more information, see AWS IoT SQL reference (p. 430).
- The first three levels of the Basic Ingest topic (`$aws/rules/`*`rule-name`*) are not counted toward the eight segment length limit or toward the 256 total character limit for a topic. Otherwise, the same restrictions apply as documented in AWS IoT Limits.
- If a message is received with a Basic Ingest topic that specifies an inactive rule or a rule that doesn't exist, an error log is created in an Amazon CloudWatch log to help you with debugging. For more information, see Rules engine log entries (p. 338). A `RuleNotFound` metric is indicated and you can create alarms on this metric. For more information, see Rule Metrics in Rule metrics (p. 322).
- You can still publish with QoS 1 on Basic Ingest topics. You receive a PUBACK after the message is successfully delivered to the rules engine. Receiving a PUBACK does not mean that your rule actions were completed successfully. You can configure an error action to handle errors during action execution. See Error handling (error action) (p. 401).

AWS IoT SQL reference

In AWS IoT, rules are defined using an SQL-like syntax. SQL statements are composed of three types of clauses:

SELECT

Required. Extracts information from the payload of an incoming message and performs transformations on the information. The messages to use are identified by the topic filter (p. 86) specified in the FROM clause.

The SELECT clause supports Data types (p. 433), Operators (p. 437), Functions (p. 442), Literals (p. 483), Case statements (p. 484), JSON extensions (p. 484), Substitution templates (p. 485), Nested object queries (p. 487), and Binary payloads (p. 488).

FROM

The MQTT message topic filter (p. 86) that identifies the messages to extract data from. The rule is triggered for each message sent to an MQTT topic that matches the topic filter specified here. Required for rules that are triggered by messages that pass through the message broker. Optional for rules that are only triggered using the Basic Ingest (p. 429) feature.

WHERE

(Optional) Adds conditional logic that determines whether the actions specified by a rule are carried out.

The WHERE clause supports Data types (p. 433), Operators (p. 437), Functions (p. 442), Literals (p. 483), Case statements (p. 484), JSON extensions (p. 484), Substitution templates (p. 485), and Nested object queries (p. 487).

An example SQL statement looks like this:

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

An example MQTT message (also called an incoming payload) looks like this:

```
{
    "color":"red",
    "temperature":100
}
```

If this message is published on the `'topic/subtopic'` topic, the rule is triggered and the SQL statement is evaluated. The SQL statement extracts the value of the `color` property if the `"temperature"` property is greater than 50. The WHERE clause specifies the condition `temperature > 50`. The `AS` keyword renames the `"color"` property to `"rgb"`. The result (also called an *outgoing payload*) looks like this:

```
{
    "rgb":"red"
}
```

This data is then forwarded to the rule's action, which sends the data for more processing. For more information about rule actions, see AWS IoT rule actions (p. 359).

SELECT clause

The AWS IoT SELECT clause is essentially the same as the ANSI SQL SELECT clause, with some minor differences.

The SELECT clause supports Data types (p. 433), Operators (p. 437), Functions (p. 442), Literals (p. 483), Case statements (p. 484), JSON extensions (p. 484), Substitution templates (p. 485), Nested object queries (p. 487), and Binary payloads (p. 488).

You can use the SELECT clause to extract information from incoming MQTT messages. You can also use `SELECT *` to retrieve the entire incoming message payload. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

If the payload is a JSON object, you can reference keys in the object. Your outgoing payload contains the key-value pair. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

You can use the AS keyword to rename keys. For example:

```
Incoming payload published on topic 'topic/subtopic':{"color":"red", "temperature":50}
SQL:SELECT color AS my_color FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red"}
```

You can select multiple items by separating them with a comma. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT color as my_color, temperature as fahrenheit FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red","fahrenheit":50}
```

You can select multiple items including '*' to add items to the incoming payload. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT *, 15 as speed FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50, "speed":15}
```

You can use the `"VALUE"` keyword to produce outgoing payloads that are not JSON objects. With SQL version `2015-10-08`, you can select only one item. With SQL version `2016-03-23` or later, you can also select an array to output as a top-level object.

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

You can use `'.'` syntax to drill into nested JSON objects in the incoming payload. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":
{"red":255,"green":0,"blue":0}, "temperature":50}
SQL: SELECT color.red as red_value FROM 'topic/subtopic'
Outgoing payload: {"red_value":255}
```

For information about how to use JSON object and property names that include reserved characters, such as numbers or the hyphen (minus) character, see JSON extensions (p. 484)

You can use functions (see Functions (p. 442)) to transform the incoming payload. You can use parentheses for grouping. For example:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM 'topic/
subtopic'
Outgoing payload: {"celsius":10,"my_color":"RED"}
```

FROM clause

The FROM clause subscribes your rule to a topic (p. 85) or topic filter (p. 86). You must enclose the topic or topic filter in single quotes ('). The rule is triggered for each message sent to an MQTT topic that matches the topic filter specified here. A topic filter allows you to subscribe to a group of similar topics.

Example:

Incoming payload published on topic `'topic/subtopic'`: `{temperature: 50}`

Incoming payload published on topic `'topic/subtopic-2'`: `{temperature: 50}`

SQL: `"SELECT temperature AS t FROM 'topic/subtopic'"`.

The rule is subscribed to `'topic/subtopic'`, so the incoming payload is passed to the rule. The outgoing payload, passed to the rule actions, is: `{t: 50}`. The rule is not subscribed to `'topic/subtopic-2'`, so the rule is not triggered for the message published on `'topic/subtopic-2'`.

# Wildcard Example:

You can use the '#' (multi-level) wildcard character to match one or more particular path elements:

Incoming payload published on topic `'topic/subtopic'`: `{temperature: 50}`.

Incoming payload published on topic `'topic/subtopic-2'`: `{temperature: 60}`.

Incoming payload published on topic `'topic/subtopic-3/details'`: `{temperature: 70}`.

Incoming payload published on topic `'topic-2/subtopic-x'`: `{temperature: 80}`.

SQL: `"SELECT temperature AS t FROM 'topic/#'"`.

The rule is subscribed to any topic that begins with `'topic'`, so it is executed three times, sending outgoing payloads of `{t: 50}` (for topic/subtopic), `{t: 60}` (for topic/subtopic-2), and `{t: 70}` (for topic/subtopic-3/details) to its actions. It is not subscribed to `'topic-2/subtopic-x'`, so the rule is not triggered for the `{temperature: 80}` message.

+ Wildcard Example:

You can use the '+' (single-level) wildcard character to match any one particular path element:

Incoming payload published on topic `'topic/subtopic'`: `{temperature: 50}`.

Incoming payload published on topic `'topic/subtopic-2'`: `{temperature: 60}`.

Incoming payload published on topic `'topic/subtopic-3/details'`: `{temperature: 70}`.

Incoming payload published on topic `'topic-2/subtopic-x'`: `{temperature: 80}`.

SQL: `"SELECT temperature AS t FROM 'topic/+'"`.

The rule is subscribed to all topics with two path elements where the first element is `'topic'`. The rule is executed for the messages sent to `'topic/subtopic'` and `'topic/subtopic-2'`, but not `'topic/subtopic-3/details'` (it has more levels than the topic filter) or `'topic-2/subtopic-x'` (it does not start with `topic`).

WHERE clause

The WHERE clause determines if the actions specified by a rule are carried out. If the WHERE clause evaluates to true, the rule actions are performed. Otherwise, the rule actions are not performed.

The WHERE clause supports Data types (p. 433), Operators (p. 437), Functions (p. 442), Literals (p. 483), Case statements (p. 484), JSON extensions (p. 484), Substitution templates (p. 485), and Nested object queries (p. 487).

Example:

Incoming payload published on `topic/subtopic`: `{"color":"red", "temperature":40}`.

SQL: `SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'`.

In this case, the rule would be triggered, but the actions specified by the rule would not be performed. There would be no outgoing payload.

You can use functions and operators in the WHERE clause. However, you cannot reference any aliases created with the AS keyword in the SELECT. The WHERE clause is evaluated first, to determine if SELECT is evaluated.

Data types

The AWS IoT rules engine supports all JSON data types.

Supported data types

Type	Meaning
`Int`	A discrete `Int`. 34 digits maximum.
`Decimal`	A `Decimal` with a precision of 34 digits, with a minimum non-zero magnitude of 1E-999 and a maximum magnitude 9.999...E999. **Note** Some functions return `Decimal` values with double precision rather than 34-digit precision. With SQL V2 (2016-03-23), numeric values that are whole numbers, such as `10.0`, are processed as an `Int` value (`10`) instead of the expected `Decimal` value (`10.0`). To reliably process whole number numeric values as `Decimal` values, use SQL V1 (2015-10-08) for the rule query statement.
`Boolean`	`True` or `False`.
`String`	A UTF-8 string.
`Array`	A series of values that don't have to have the same type.
`Object`	A JSON value consisting of a key and a value. Keys must be strings. Values can be any type.
`Null`	`Null` as defined by JSON. It's an actual value that represents the absence of a value. You can explicitly create a `Null` value by using the `Null` keyword in your SQL statement. For example: `"SELECT NULL AS n FROM 'topic/subtopic'"`

`Undefined`	Not a value. This isn't explicitly representable in JSON except by omitting the value. For example, in the object `{"foo": null}`, the key "foo" returns NULL, but the key "bar" returns `Undefined`. Internally, the SQL language treats `Undefined` as a value, but it isn't representable in JSON, so when serialized to JSON, the results are `Undefined`.
	
	 `{"foo":null, "bar":undefined}`
	
	 is serialized to JSON as:
	
	 `{"foo":null}`
	
	 Similarly, `Undefined` is converted to an empty string when serialized by itself. Functions called with invalid arguments (for example, wrong types,

Type	Meaning
	wrong number of arguments, and so on) return `Undefined`.

Conversions

The following table lists the results when a value of one type is converted to another type (when a value of the incorrect type is given to a function). For example, if the absolute value function "abs" (which expects an `Int` or `Decimal`) is given a `String`, it attempts to convert the `String` to a `Decimal`, following these rules. In this case, 'abs("-5.123")' is treated as 'abs(-5.123)'.

Note
There are no attempted conversions to `Array`, `Object`, `Null`, or `Undefined`.

To decimal

Argument type	Result
`Int`	A `Decimal` with no decimal point.
`Decimal`	The source value.
`Boolean`	`Undefined`. (You can explicitly use the cast function to transform true = 1.0, false = 0.0.)
`String`	The SQL engine tries to parse the string as a `Decimal`. AWS IoT attempts to parse strings matching the regular expression:`^-?\d+(\.\d+)?((?i)E-?\d+)?$`. "0", "-1.2", "5E-12" are all examples of strings that are converted automatically to `Decimals`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Null`.
Undefined	`Undefined`.

To int

Argument type	Result
`Int`	The source value.
`Decimal`	The source value rounded to the nearest `Int`.
`Boolean`	`Undefined`. (You can explicitly use the cast function to transform true = 1.0, false = 0.0.)
`String`	The SQL engine tries to parse the string as a `Decimal`. AWS IoT attempts to parse strings matching the regular expression:`^-?\d+(\.\d+)?((?i)E-?\d+)?$`. "0", "-1.2", "5E-12" are all examples of strings that are converted automatically to `Decimals`. AWS IoT attempts to convert the `String` to a `Decimal`, and then

Argument type	Result
	truncates the decimal places of that `Decimal` to make an `Int`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Null`.
Undefined	`Undefined`.

To Boolean

Argument type	Result
`Int`	`Undefined`. (You can explicitly use the `cast` function to transform 0 = False, any_nonzero_value = True.)
`Decimal`	`Undefined`. (You can explicitly use the cast function to transform 0 = False, any_nonzero_value = True.)
`Boolean`	The original value.
`String`	"true"=True and "false"=False (case insensitive). Other string values are `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

To string

Argument type	Result
`Int`	A string representation of the `Int` in standard notation.
`Decimal`	A string representing the `Decimal` value, possibly in scientific notation.
`Boolean`	"true" or "false". All lowercase.
`String`	The original value.
Array	The `Array` serialized to JSON. The resultant string is a comma-separated list, enclosed in square brackets. A `String` is quoted. A `Decimal`, `Int`, `Boolean`, and `Null` is not.
Object	The object serialized to JSON. The resultant string is a comma-separated list of key-value pairs and

Argument type	Result
	begins and ends with curly braces. A `String` is quoted. A `Decimal`, `Int`, `Boolean`, and `Null` is not.
Null	`Undefined`.
Undefined	Undefined.

Operators

The following operators can be used in SELECT and WHERE clauses.

AND operator

Returns a `Boolean` result. Performs a logical AND operation. Returns true if left and right operands are true. Otherwise, returns false. `Boolean` operands or case insensitive "true" or "false" string operands are required.

Syntax: `expression AND expression`.

AND operator

Left operand	Right operand	Output
`Boolean`	`Boolean`	`Boolean`. True if both operands are true. Otherwise, false.
`String/Boolean`	`String/Boolean`	If all strings are "true" or "false" (case insensitive), they are converted to `Boolean` and processed normally as `boolean AND boolean`.
Other value	Other value	`Undefined`.

OR operator

Returns a `Boolean` result. Performs a logical OR operation. Returns true if either the left or the right operands are true. Otherwise, returns false. `Boolean` operands or case insensitive "true" or "false" string operands are required.

Syntax: `expression OR expression`.

OR operator

Left operand	Right operand	Output
`Boolean`	`Boolean`	`Boolean`. True if either operand is true. Otherwise, false.
`String/Boolean`	`String/Boolean`	If all strings are "true" or "false" (case insensitive), they are converted to Booleans and processed normally as `boolean OR boolean`.
Other value	Other value	`Undefined`.

NOT operator

Returns a `Boolean` result. Performs a logical NOT operation. Returns true if the operand is false. Otherwise, returns true. A `Boolean` operand or case insensitive "true" or "false" string operand is required.

Syntax: `NOT` *`expression`*.

NOT operator

Operand	Output
`Boolean`	`Boolean`. True if operand is false. Otherwise, true.
`String`	If string is "true" or "false" (case insensitive), it is converted to the corresponding boolean value, and the opposite value is returned.
Other value	`Undefined`.

> operator

Returns a `Boolean` result. Returns true if the left operand is greater than the right operand. Both operands are converted to a `Decimal`, and then compared.

Syntax: *`expression`* `>` *`expression`*.

> operator

Left operand	Right operand	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. True if the left operand is greater than the right operand. Otherwise, false.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings can be converted to `Decimal`, then `Boolean`. Returns true if the left operand is greater than the right operand. Otherwise, false.
Other value	`Undefined`.	`Undefined`.

>= operator

Returns a `Boolean` result. Returns true if the left operand is greater than or equal to the right operand. Both operands are converted to a `Decimal`, and then compared.

Syntax: *`expression`* `>=` *`expression`*.

>= operator

Left operand	Right operand	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. True if the left operand is greater than or equal to the right operand. Otherwise, false.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings can be converted to `Decimal`, then `Boolean`. Returns true if the left operand is greater than or equal to the right operand. Otherwise, false.

Left operand	Right operand	Output
Other value	`Undefined`.	`Undefined`.

< operator

Returns a `Boolean` result. Returns true if the left operand is less than the right operand. Both operands are converted to a `Decimal`, and then compared.

Syntax: `expression < expression`.

< operator

Left operand	Right operand	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. True if the left operand is less than the right operand. Otherwise, false.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings can be converted to `Decimal`, then `Boolean`. Returns true if the left operand is less than the right operand. Otherwise, false.
Other value	`Undefined`	`Undefined`

<= operator

Returns a `Boolean` result. Returns true if the left operand is less than or equal to the right operand. Both operands are converted to a `Decimal`, and then compared.

Syntax: `expression <= expression`.

<= operator

Left operand	Right operand	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. True if the left operand is less than or equal to the right operand. Otherwise, false.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings can be converted to `Decimal`, then `Boolean`. Returns true if the left operand is less than or equal to the right operand. Otherwise, false.
Other value	`Undefined`	`Undefined`

<> operator

Returns a `Boolean` result. Returns true if both left and right operands are not equal. Otherwise, returns false.

Syntax: `expression <> expression`.

<> operator

Left operand	Right operand	Output
`Int`	`Int`	True if left operand is not equal to right operand. Otherwise, false.
`Decimal`	`Decimal`	True if left operand is not equal to right operand. Otherwise, false.`Int` is converted to `Decimal` before being compared.
`String`	`String`	True if left operand is not equal to right operand. Otherwise, false.
Array	Array	True if the items in each operand are not equal and not in the same order. Otherwise, false
Object	Object	True if the keys and values of each operand are not equal. Otherwise, false. The order of keys/values is unimportant.
Null	Null	False.
Any value	`Undefined`	Undefined.
`Undefined`	Any value	Undefined.
Mismatched type	Mismatched type	True.

= operator

Returns a `Boolean` result. Returns true if both left and right operands are equal. Otherwise, returns false.

Syntax: `expression = expression`.

= operator

Left operand	Right operand	Output
`Int`	`Int`	True if left operand is equal to right operand. Otherwise, false.
`Decimal`	`Decimal`	True if left operand is equal to right operand. Otherwise, false.`Int` is converted to `Decimal` before being compared.
`String`	`String`	True if left operand is equal to right operand. Otherwise, false.
Array	Array	True if the items in each operand are equal and in the same order. Otherwise, false.
Object	Object	True if the keys and values of each operand are equal. Otherwise, false. The order of keys/values is unimportant.
Any value	`Undefined`	`Undefined`.
`Undefined`	Any value	`Undefined`.
Mismatched type	Mismatched type	False.

+ operator

The "+" is an overloaded operator. It can be used for string concatenation or addition.

Syntax: `expression + expression`.

+ operator

Left operand	Right operand	Output
`String`	Any value	Converts the right operand to a string and concatenates it to the end of the left operand.
Any value	`String`	Converts the left operand to a string and concatenates the right operand to the end of the converted left operand.
`Int`	`Int`	`Int` value. Adds operands together.
`Int/Decimal`	`Int/Decimal`	`Decimal` value. Adds operands together.
Other value	Other value	`Undefined`.

- operator

Subtracts the right operand from the left operand.

Syntax: `expression - expression`.

- operator

Left operand	Right operand	Output
`Int`	`Int`	`Int` value. Subtracts right operand from left operand.
`Int/Decimal`	`Int/Decimal`	`Decimal` value. Subtracts right operand from left operand.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert to decimals correctly, a `Decimal` value is returned. Subtracts right operand from left operand. Otherwise, returns `Undefined`.
Other value	Other value	`Undefined`.
Other value	Other value	`Undefined`.

* operator

Multiplies the left operand by the right operand.

Syntax: `expression * expression`.

*** operator**

Left operand	Right operand	Output
`Int`	`Int`	`Int` value. Multiplies the left operand by the right operand.
`Int/Decimal`	`Int/Decimal`	`Decimal` value. Multiplies the left operand by the right operand.

Left operand	Right operand	Output
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert to decimals correctly, a `Decimal` value is returned. Multiplies the left operand by the right operand. Otherwise, returns `Undefined`.
Other value	Other value	`Undefined`.

/ operator

Divides the left operand by the right operand.

Syntax: `expression / expression`.

/ operator

Left operand	Right operand	Output
`Int`	`Int`	`Int` value. Divides the left operand by the right operand.
`Int/Decimal`	`Int/Decimal`	`Decimal` value. Divides the left operand by the right operand.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert to decimals correctly, a `Decimal` value is returned. Divides the left operand by the right operand. Otherwise, returns `Undefined`.
Other value	Other value	`Undefined`.

% operator

Returns the remainder from dividing the left operand by the right operand.

Syntax: `expression % expression`.

% operator

Left operand	Right operand	Output
`Int`	`Int`	`Int` value. Returns the remainder from dividing the left operand by the right operand.
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert to decimals correctly, a `Decimal` value is returned. Returns the remainder from dividing the left operand by the right operand. Otherwise, `Undefined`.
Other value	Other value	`Undefined`.

Functions

You can use the following built-in functions in the SELECT or WHERE clauses of your SQL expressions.

abs(Decimal)

Returns the absolute value of a number. Supported by SQL version 2015-10-08 and later.

Example: `abs(-5)` returns 5.

Argument type	Result
`Int`	`Int`, the absolute value of the argument.
`Decimal`	`Decimal`, the absolute value of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal`. The result is the absolute value of the argument. If the string cannot be converted, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

accountid()

Returns the ID of the account that owns this rule as a `String`. Supported by SQL version 2015-10-08 and later.

Example:

`accountid()` = "123456789012"

acos(Decimal)

Returns the inverse cosine of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `acos(0)` = 1.5707963267948966

Argument type	Result
`Int`	`Decimal` (with double precision), the inverse cosine of the argument. Imaginary results are returned as `Undefined`.
`Decimal`	`Decimal` (with double precision), the inverse cosine of the argument. Imaginary results are returned as `Undefined`.
`Boolean`	`Undefined`.
`String`	`Decimal`, the inverse cosine of the argument. If the string cannot be converted, the result is `Undefined`. Imaginary results are returned as `Undefined`.
Array	`Undefined`.
Object	`Undefined`.

Argument type	Result
Null	`Undefined`.
Undefined	`Undefined`.

asin(Decimal)

Returns the inverse sine of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `asin(0)` = 0.0

Argument type	Result
`Int`	`Decimal` (with double precision), the inverse sine of the argument. Imaginary results are returned as `Undefined`.
`Decimal`	`Decimal` (with double precision), the inverse sine of the argument. Imaginary results are returned as `Undefined`.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the inverse sine of the argument. If the string cannot be converted, the result is `Undefined`. Imaginary results are returned as `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

atan(Decimal)

Returns the inverse tangent of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `atan(0)` = 0.0

Argument type	Result
`Int`	`Decimal` (with double precision), the inverse tangent of the argument. Imaginary results are returned as `Undefined`.
`Decimal`	`Decimal` (with double precision), the inverse tangent of the argument. Imaginary results are returned as `Undefined`.
`Boolean`	`Undefined`.

Argument type	Result
`String`	`Decimal`, the inverse tangent of the argument. If the string cannot be converted, the result is `Undefined`. Imaginary results are returned as `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

atan2(Decimal, Decimal)

Returns the angle, in radians, between the positive x-axis and the (x, y) point defined in the two arguments. The angle is positive for counter-clockwise angles (upper half-plane, y > 0), and negative for clockwise angles (lower half-plane, y < 0). `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `atan2(1, 0)` = 1.5707963267948966

Argument type	Argument type	Result
`Int/Decimal`	`Int/Decimal`	`Decimal` (with doubl x-axis and the specifi
`Int/Decimal/String`	`Int/Decimal/String`	`Decimal`, the inverse string cannot be conv
Other value	Other value	`Undefined`.

aws_lambda(functionArn, inputJson)

Calls the specified Lambda function passing `inputJson` to the Lambda function and returns the JSON generated by the Lambda function.

Arguments

Argument	Description
`functionArn`	The ARN of the Lambda function to call. The Lambda function must return JSON data.
`inputJson`	The JSON input passed to the Lambda function.

You must grant AWS IoT `lambda:InvokeFunction` permissions to invoke the specified Lambda function. The following example shows how to grant the `lambda:InvokeFunction` permission using the AWS CLI:

```
aws lambda add-permission --function-name "function_name"
--region "region"
--principal iot.amazonaws.com
--source-arn arn:aws:iot:us-east-1:account_id:rule/rule_name
```

```
--source-account "account_id"
--statement-id "unique_id"
--action "lambda:InvokeFunction"
```

The following are the arguments for the **add-permission** command:

--function-name

Name of the Lambda function. You add a new permission to update the function's resource policy.

--region

The AWS Region of your account.

--principal

The principal who is getting the permission. This should be `iot.amazonaws.com` to allow AWS IoT permission to call a Lambda function.

--source-arn

The ARN of the rule. You can use the **get-topic-rule** AWS CLI command to get the ARN of a rule.

--source-account

The AWS account where the rule is defined.

--statement-id

A unique statement identifier.

--action

The Lambda action that you want to allow in this statement. To allow AWS IoT to invoke a Lambda function, specify `lambda:InvokeFunction`.

Important

If you add a permission for an AWS IoT principal without providing the source ARN, any AWS account that creates a rule with your Lambda action can trigger rules to invoke your Lambda function from AWS IoT. For more information, see Lambda Permission Model.

Given a JSON message payload like:

```
{
    "attribute1": 21,
    "attribute2": "value"
}
```

The `aws_lambda` function can be used to call Lambda function as follows.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

If you want to pass the full MQTT message payload, you can specify the JSON payload using '*', such as the following example.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

`payload.inner.element` selects data from message published on topic 'topic/subtopic'.

`some.value` selects data from the output that is generated by the Lambda function.

Note
The rules engine limits the execution duration of Lambda functions. Lambda function calls from rules should be completed within 2000 milliseconds.

bitand(Int, Int)

Performs a bitwise AND on the bit representations of the two `Int`(-converted) arguments. Supported by SQL version 2015-10-08 and later.

Example: `bitand(13, 5)` = 5

Argument type	Argument type	Result
`Int`	`Int`	`Int`, a bitwise AND o
`Int/Decimal`	`Int/Decimal`	`Int`, a bitwise AND o numbers are roundec the arguments canno is `Undefined`.
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, a bitwise AND o are converted to deci the nearest `Int`. If th `Undefined`.
Other value	Other value	`Undefined`.

bitor(Int, Int)

Performs a bitwise OR of the bit representations of the two arguments. Supported by SQL version 2015-10-08 and later.

Example: `bitor(8, 5)` = 13

Argument type	Argument type	Result
`Int`	`Int`	`Int`, the bitwise OR c
`Int/Decimal`	`Int/Decimal`	`Int`, the bitwise OR c numbers are roundec conversion fails, the r
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, the bitwise OR c converted to decimal `Int`. If the conversior
Other value	Other value	`Undefined`.

bitxor(Int, Int)

Performs a bitwise XOR on the bit representations of the two `Int`(-converted) arguments. Supported by SQL version 2015-10-08 and later.

Example:`bitor(13, 5)` = 8

Argument type	Argument type	Result
`Int`	`Int`	`Int`, a bitwise XOR o
`Int/Decimal`	`Int/Decimal`	`Int`, a bitwise XOR o numbers are rounde
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, a bitwise XOR o converted to decima `Int`. If any conversio
Other value	Other value	`Undefined`.

bitnot(Int)

Performs a bitwise NOT on the bit representations of the `Int`(-converted) argument. Supported by SQL version 2015-10-08 and later.

Example: `bitnot(13)` = 2

Argument type	Result
`Int`	`Int`, a bitwise NOT of the argument.
`Decimal`	`Int`, a bitwise NOT of the argument. The `Decimal` value is rounded down to the nearest `Int`.
`String`	`Int`, a bitwise NOT of the argument. Strings are converted to decimals and rounded down to the nearest `Int`. If any conversion fails, the result is `Undefined`.
Other value	Other value.

cast()

Converts a value from one data type to another. Cast behaves mostly like the standard conversions, with the addition of the ability to cast numbers to or from Booleans. If AWS IoT cannot determine how to cast one type to another, the result is `Undefined`. Supported by SQL version 2015-10-08 and later. Format: cast(*`value`* as *`type`*).

Example:

`cast(true as Int)` = 1

The following keywords might appear after "as" when calling `cast`:

For SQL version 2015-10-08 and 2016-03-23

Keyword	Result
`String`	Casts value to `String`.
Nvarchar	Casts value to `String`.
Text	Casts value to `String`.

Keyword	Result
Ntext	Casts value to `String`.
varchar	Casts value to `String`.
`Int`	Casts value to `Int`.
Integer	Casts value to `Int`.

Additionally, for SQL version 2016-03-23

Keyword	Result
`Decimal`	Casts value to `Decimal`.
Bool	Casts value to `Boolean`.
`Boolean`	Casts value to `Boolean`.

Casting rules:

Cast to decimal

Argument type	Result
`Int`	A `Decimal` with no decimal point.
`Decimal`	The source value. **Note** With SQL V2 (2016-03-23), numeric values that are whole numbers, such as `10.0`, return an `Int` value (`10`) instead of the expected `Decimal` value (`10.0`). To reliably cast whole number numeric values as `Decimal` values, use SQL V1 (2015-10-08) for the rule query statement.
`Boolean`	true = 1.0, false = 0.0.
`String`	Tries to parse the string as a `Decimal`. AWS IoT attempts to parse strings matching the regex: ^-?\d+(\.\d+)?((?i)E-?\d+)?$. "0", "-1.2", "5E-12" are all examples of strings that are converted automatically to decimals.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

Cast to int

Argument type	Result
`Int`	The source value.

Argument type	Result
`Decimal`	The source value, rounded down to the nearest `Int`.
`Boolean`	true = 1.0, false = 0.0.
`String`	Tries to parse the string as a `Decimal`. AWS IoT attempts to parse strings matching the regex: ^-?\d+(\.\d+)?((?i)E-?\d+)?$. "0", "-1.2", "5E-12" are all examples of strings that are converted automatically to decimals. AWS IoT attempts to convert the string to a `Decimal` and round down to the nearest `Int`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

Cast to `Boolean`

Argument type	Result
`Int`	0 = False, any_nonzero_value = True.
`Decimal`	0 = False, any_nonzero_value = True.
`Boolean`	The source value.
`String`	"true" = True and "false" = False (case insensitive). Other string values = `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

Cast to string

Argument type	Result
`Int`	A string representation of the `Int`, in standard notation.
`Decimal`	A string representing the `Decimal` value, possibly in scientific notation.
`Boolean`	"true" or "false", all lowercase.
`String`	"true"=True and "false"=False (case-insensitive). Other string values = `Undefined`.
Array	The array serialized to JSON. The result string is a comma-separated list enclosed in square brackets. `String` is quoted. `Decimal`, `Int`, and `Boolean` are not.

Argument type	Result
Object	The object serialized to JSON. The JSON string is a comma-separated list of key-value pairs and begins and ends with curly braces. `String` is quoted. `Decimal`, `Int`, `Boolean`, and `Null` are not.
Null	`Undefined`.
Undefined	`Undefined`.

ceil(Decimal)

Rounds the given `Decimal` up to the nearest `Int`. Supported by SQL version 2015-10-08 and later.

Examples:

`ceil(1.2)` = 2

`ceil(-1.2)` = -1

Argument type	Result
`Int`	`Int`, the argument value.
`Decimal`	`Int`, the `Decimal` value rounded up to the nearest `Int`.
`String`	`Int`. The string is converted to `Decimal` and rounded up to the nearest `Int`. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Other value	`Undefined`.

chr(String)

Returns the ASCII character that corresponds to the given `Int` argument. Supported by SQL version 2015-10-08 and later.

Examples:

`chr(65)` = "A".

`chr(49)` = "1".

Argument type	Result
`Int`	The character corresponding to the specified ASCII value. If the argument is not a valid ASCII value, the result is `Undefined`.
`Decimal`	The character corresponding to the specified ASCII value. The `Decimal` argument is rounded down to the nearest `Int`. If the argument is not a valid ASCII value, the result is `Undefined`.

Argument type	Result
`Boolean`	`Undefined`.
`String`	If the `String` can be converted to a `Decimal`, it is rounded down to the nearest `Int`. If the argument is not a valid ASCII value, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Other value	`Undefined`.

clientid()

Returns the ID of the MQTT client sending the message, or `n/a` if the message wasn't sent over MQTT. Supported by SQL version 2015-10-08 and later.

Example:

`clientid()` = "123456789012"

concat()

Concatenates arrays or strings. This function accepts any number of arguments and returns a `String` or an `Array`. Supported by SQL version 2015-10-08 and later.

Examples:

`concat()` = `Undefined`.

`concat(1)` = "1".

`concat([1, 2, 3], 4)` = [1, 2, 3, 4].

`concat([1, 2, 3], "hello")` = [1, 2, 3, "hello"]

`concat("con", "cat")` = "concat"

`concat(1, "hello")` = "1hello"

`concat("he","is","man")` = "heisman"

`concat([1, 2, 3], "hello", [4, 5, 6])` = [1, 2, 3, "hello", 4, 5, 6]

Number of arguments	Result
0	`Undefined`.
1	The argument is returned unmodified.
2+	If any argument is an `Array`, the result is a single array containing all of the arguments. If no arguments are arrays, and at least one argument is a `String`, the result is the concatenation of the `String` representations of all

Number of arguments	Result
	the arguments. Arguments are converted to strings using the standard conversions listed above. .

cos(Decimal)

Returns the cosine of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example:

`cos(0)` = 1.

Argument type	Result
`Int`	`Decimal` (with double precision), the cosine of the argument. Imaginary results are returned as `Undefined`.
`Decimal`	`Decimal` (with double precision), the cosine of the argument. Imaginary results are returned as `Undefined`.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the cosine of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`. Imaginary results are returned as `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

cosh(Decimal)

Returns the hyperbolic cosine of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `cosh(2.3)` = 5.037220649268761.

Argument type	Result
`Int`	`Decimal` (with double precision), the hyperbolic cosine of the argument. Imaginary results are returned as `Undefined`.
`Decimal`	`Decimal` (with double precision), the hyperbolic cosine of the argument. Imaginary results are returned as `Undefined`.

Argument type	Result
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the hyperbolic cosine of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`. Imaginary results are returned as `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

decode(value, decodingScheme)

Use the `decode` function to decode an encoded value. If the decoded string is a JSON document, an addressable object is returned. Otherwise, the decoded string is returned as a string. The function returns NULL if the string cannot be decoded.

Supported by SQL version 2016-03-23 and later.

value

A string value or any of the valid expressions, as defined in AWS IoT SQL reference (p. 430), that return a string.

decodingScheme

A literal string representing the scheme used to decode the value. Currently, only `'base64'` is supported.

Example

In this example, the message payload includes an encoded value.

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
}
```

The `decode` function in this SQL statement, decodes the value in the message payload.

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

Decoding the `encoded_temp` value results in the following valid JSON document, which allows the SELECT statement to read the temperature value.

```
{ "temperature": 33 }
```

The result of the SELECT statement in this example is shown here.

```
{ "temp": 33 }
```

If the decoded value was not a valid JSON document, the decoded value would be returned as a string.

encode(value, encodingScheme)

Use the `encode` function to encode the payload, which potentially might be non-JSON data, into its string representation based on the encoding scheme. Supported by SQL version 2016-03-23 and later.

value

Any of the valid expressions, as defined in AWS IoT SQL reference (p. 430). You can specify * to encode the entire payload, regardless of whether it's in JSON format. If you supply an expression, the result of the evaluation is converted to a string before it is encoded.

encodingScheme

A literal string representing the encoding scheme you want to use. Currently, only `'base64'` is supported.

endswith(String, String)

Returns a `Boolean` indicating whether the first `String` argument ends with the second `String` argument. If either argument is `Null` or `Undefined`, the result is `Undefined`. Supported by SQL version 2015-10-08 and later.

Example: `endswith("cat","at")` = true.

Argument type 1	Argument type 2	Result
`String`	`String`	True if the first argun Otherwise, false.
Other value	Other value	Both arguments are c standard conversion r ends in the second ar either argument is `Nu` `Undefined`.

exp(Decimal)

Returns e raised to the `Decimal` argument. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `exp(1)` = e.

Argument type	Result
`Int`	`Decimal` (with double precision), e ^ argument.
`Decimal`	`Decimal` (with double precision), e ^ argument.
`String`	`Decimal` (with double precision), e ^ argument. If the `String` cannot be converted to a `Decimal`, the result is `Undefined`.
Other value	`Undefined`.

floor(Decimal)

Rounds the given `Decimal` down to the nearest `Int`. Supported by SQL version 2015-10-08 and later.

Examples:

`floor(1.2)` = 1

`floor(-1.2)` = -2

Argument type	Result
`Int`	`Int`, the argument value.
`Decimal`	`Int`, the `Decimal` value rounded down to the nearest `Int`.
`String`	`Int`. The string is converted to `Decimal` and rounded down to the nearest `Int`. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Other value	`Undefined`.

get

Extracts a value from a collection-like type (Array, String, Object). No conversion is applied to the first argument. Conversion applies as documented in the table to the second argument. Supported by SQL version 2015-10-08 and later.

Examples:

`get(["a", "b", "c"], 1)` = "b"

`get({"a":"b"}, "a")` = "b"

`get("abc", 1)` = "b"

Argument type 1	Argument type 2	Result
Array	Any Type (converted to `Int`)	The item at the 0-bas by the second argum conversion is unsucce index is outside the b array.length), the resu
String	Any Type (converted to `Int`)	The character at the (by the second argum conversion is unsucce index is outside the b string.length), the res
Object	`String` (no conversion is applied)	The value stored in th corresponding to the argument.

Argument type 1	Argument type 2	Result
Other value	Any value	`Undefined`.

get_dynamodb(tableName, partitionKeyName, partitionKeyValue, sortKeyName, sortKeyValue, roleArn)

Retrieves data from a DynamoDB table. `get_dynamodb()` allows you to query a DynamoDB table while a rule is evaluated. You can filter or augment message payloads using data retrieved from DynamoDB. Supported by SQL version 2016-03-23 and later.

`get_dynamodb()` takes the following parameters:

tableName

The name of the DynamoDB table to query.

partitionKeyName

The name of the partition key. For more information, see DynamoDB Keys.

partitionKeyValue

The value of the partition key used to identify a record. For more information, see DynamoDB Keys.

sortKeyName

(Optional) The name of the sort key. This parameter is required only if the DynamoDB table queried uses a composite key. For more information, see DynamoDB Keys.

sortKeyValue

(Optional) The value of the sort key. This parameter is required only if the DynamoDB table queried uses a composite key. For more information, see DynamoDB Keys.

roleArn

The ARN of an IAM role that grants access to the DynamoDB table. The rules engine assumes this role to access the DynamoDB table on your behalf. Avoid using an overly permissive role. Grant the role only those permissions required by the rule. The following is an example policy that grants access to one DynamoDB table.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:aws-region:account-id:table/table-name"
        }
    ]
}}
```

As an example of how you can use `get_dynamodb()`, say you have a DynamoDB table that contains device ID and location information for all of your devices connected to AWS IoT. The following SELECT statement uses the `get_dynamodb()` function to retrieve the location for the specified device ID:

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

Note

- You can call `get_dynamodb()` a maximum of one time per SQL statement. Calling `get_dynamodb()` multiple times in a single SQL statement causes the rule to terminate without invoking any actions.
- If `get_dynamodb()` returns more than 8 KB of data, the rule's action may not be invoked.

get_secret(secretId, secretType, key, roleArn)

Retrieves the value of the encrypted `SecretString` or `SecretBinary` field of the current version of a secret in AWS Secrets Manager. For more information about creating and maintaining secrets, see CreateSecret, UpdateSecret, and PutSecretValue.

`get_secret()` takes the following parameters:

secretId

String: The Amazon Resource Name (ARN) or the friendly name of the secret to retrieve.

secretType

String: The secret type. Valid values: `SecretString` | `SecretBinary`.

SecretString

- For secrets that you create as JSON objects by using the APIs, the AWS CLI, or the AWS Secrets Manager console:
  - If you specify a value for the `key` parameter, this function returns the value of the specified key.
  - If you don't specify a value for the `key` parameter, this function returns the entire JSON object.
- For secrets that you create as non-JSON objects by using the APIs or the AWS CLI:
  - If you specify a value for the `key` parameter, this function fails with an exception.
  - If you don't specify a value for the `key` parameter, this function returns the contents of the secret.

SecretBinary

- If you specify a value for the `key` parameter, this function fails with an exception.
- If you don't specify a value for the `key` parameter, this function returns the secret value as a base64-encoded UTF-8 string.

key

(Optional) String: The key name inside a JSON object stored in the `SecretString` field of a secret. Use this value when you want to retrieve only the value of a key stored in a secret instead of the entire JSON object.

If you specify a value for this parameter and the secret doesn't contain a JSON object inside its `SecretString` field, this function fails with an exception.

roleArn

String: A role ARN with `secretsmanager:GetSecretValue` and `secretsmanager:DescribeSecret` permissions.

Note

This function always returns the current version of the secret (the version with the `AWSCURRENT` tag). The AWS IoT rules engine caches each secret for up to 15 minutes. As a result, the rules

engine can take up to 15 minutes to update a secret. This means that if you retrieve a secret up to 15 minutes after an update with AWS Secrets Manager, this function might return the older version.
This function is not metered and is free to use, but AWS Secrets Manager charges apply. Because of the secret caching mechanism, the rules engine occasionally calls AWS Secrets Manager. Because the rules engine is a fully distributed service, you might see multiple Secrets Manager API calls from the rules engine during the 15-minute caching window.

Examples:

You can use the `get_secret` function in an authentication header in an HTTPS rule action, as in the following API key authentication example.

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

For more information about the HTTPS rule action, see the section called "HTTPS" (p. 374).

get_thing_shadow(thingName, shadowName, roleARN)

Returns the specified shadow of the specified thing. Supported by SQL version 2016-03-23 and later.

thingName

String: The name of the thing whose shadow you want to retrieve.

shadowName

(Optional) String: The name of the shadow. This parameter is required only when referencing named shadows.

roleArn

String: A role ARN with `iot:GetThingShadow` permission.

Examples:

When used with a named shadow, provide the `shadowName` parameter.

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

When used with an unnamed shadow, omit the `shadowName` parameter.

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

Hashing functions

AWS IoT provides the following hashing functions:

- md2
- md5
- sha1
- sha224
- sha256
- sha384
- sha512

All hash functions expect one string argument. The result is the hashed value of that string. Standard string conversions apply to non-string arguments. All hash functions are supported by SQL version 2015-10-08 and later.

Examples:

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(String, String)

Returns the first index (0-based) of the second argument as a substring in the first argument. Both arguments are expected as strings. Arguments that are not strings are subjected to standard string conversion rules. This function does not apply to arrays, only to strings. Supported by SQL version 2016-03-23 and later.

Examples:

`indexof("abcd", "bc")` = 1

isNull()

Returns true if the argument is the `Null` value. Supported by SQL version 2015-10-08 and later.

Examples:

`isNull(5)` = false.

`isNull(Null)` = true.

Argument type	**Result**
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	true

Argument type	Result
`Undefined`	false

isUndefined()

Returns true if the argument is `Undefined`. Supported by SQL version 2016-03-23 and later.

Examples:

`isUndefined(5)` = false.

`isUndefined(floor([1,2,3]))` = true.

Argument type	Result
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	false
`Undefined`	true

length(String)

Returns the number of characters in the provided string. Standard conversion rules apply to non-`String` arguments. Supported by SQL version 2016-03-23 and later.

Examples:

`length("hi")` = 2

`length(false)` = 5

ln(Decimal)

Returns the natural logarithm of the argument. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `ln(e)` = 1.

Argument type	Result
`Int`	`Decimal` (with double precision), the natural log of the argument.

Argument type	Result
`Decimal`	`Decimal` (with double precision), the natural log of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the natural log of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

log(Decimal)

Returns the base 10 logarithm of the argument. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `log(100)` = 2.0.

Argument type	Result
`Int`	`Decimal` (with double precision), the base 10 log of the argument.
`Decimal`	`Decimal` (with double precision), the base 10 log of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the base 10 log of the argument. If the `String` cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

lower(String)

Returns the lowercase version of the given `String`. Non-string arguments are converted to strings using the standard conversion rules. Supported by SQL version 2015-10-08 and later.

Examples:

`lower("HELLO")` = "hello".

`lower(["HELLO"])` = "[\"hello\"]".

lpad(String, Int)

Returns the `String` argument, padded on the left side with the number of spaces specified by the second argument. The `Int` argument must be between 0 and 1000. If the provided value is outside of this valid range, the argument is set to the nearest valid value (0 or 1000). Supported by SQL version 2015-10-08 and later.

Examples:

`lpad("hello", 2)` = "`  hello`".

`lpad(1, 3)` = "`   1`"

Argument type 1	Argument type 2	Result
`String`	`Int`	`String`, the providec with a number of spa
`String`	`Decimal`	The `Decimal` argume `Int` and the `String` specified number of s
`String`	`String`	The second argumen is rounded down to t padded with the spec the second argument result is `Undefined`.
Other value	`Int/Decimal/String`	The first value is conv standard conversions applied on that `Stri` result is `Undefined`.
Any value	Other value	`Undefined`.

ltrim(String)

Removes all leading white space (tabs and spaces) from the provided `String`. Supported by SQL version 2015-10-08 and later.

Example:

`Ltrim(" h i ")` = "hi ".

Argument type	Result
`Int`	The `String` representation of the `Int` with all leading white space removed.
`Decimal`	The `String` representation of the `Decimal` with all leading white space removed.
`Boolean`	The `String` representation of the boolean ("true" or "false") with all leading white space removed.
`String`	The argument with all leading white space removed.

Argument type	Result
Array	The `String` representation of the `Array` (using standard conversion rules) with all leading white space removed.
Object	The `String` representation of the Object (using standard conversion rules) with all leading white space removed.
Null	`Undefined`.
Undefined	`Undefined`.

machinelearning_predict(modelId)

Use the `machinelearning_predict` function to make predictions using the data from an MQTT message based on an Amazon Machine Learning (Amazon ML) model. Supported by SQL version 2015-10-08 and later. The arguments for the `machinelearning_predict` function are:

modelId

The ID of the model against which to run the prediction. The real-time endpoint of the model must be enabled.

roleArn

The IAM role that has a policy with `machinelearning:Predict` and `machinelearning:GetMLModel` permissions and allows access to the model against which the prediction is run.

record

The data to be passed into the Amazon ML Predict API. This should be represented as a single layer JSON object. If the record is a multi-level JSON object, the record is flattened by serializing its values. For example, the following JSON:

```
{ "key1": {"innerKey1": "value1"}, "key2": 0}
```

would become:

```
{ "key1": "{\"innerKey1\": \"value1\"}", "key2": 0}
```

The function returns a JSON object with the following fields:

predictedLabel

The classification of the input based on the model.

details

Contains the following attributes:

PredictiveModelType

The model type. Valid values are REGRESSION, BINARY, MULTICLASS.

Algorithm

The algorithm used by Amazon ML to make predictions. The value must be SGD.

predictedScores

Contains the raw classification score corresponding to each label.

predictedValue

The value predicted by Amazon ML.

mod(Decimal, Decimal)

Returns the remainder of the division of the first argument by the second argument. Equivalent to remainder(Decimal, Decimal) (p. 470). You can also use "%" as an infix operator for the same modulo functionality. Supported by SQL version 2015-10-08 and later.

Example: `mod(8, 3)` = 2.

Left operand	**Right operand**	**Output**
`Int`	`Int`	`Int`, the first argume
`Int/Decimal`	`Int/Decimal`	`Decimal`, the first ar
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert t argument modulo th `Undefined`.
Other value	Other value	`Undefined`.

nanvl(AnyValue, AnyValue)

Returns the first argument if it is a valid `Decimal`. Otherwise, the second argument is returned. Supported by SQL version 2015-10-08 and later.

Example: `Nanvl(8, 3)` = 8.

Argument type 1	**Argument type 2**	**Output**
Undefined	Any value	The second argumen
Null	Any value	The second argumen
`Decimal` (NaN)	Any value	The second argumen
`Decimal` (not NaN)	Any value	The first argument.
Other value	Any value	The first argument.

newuuid()

Returns a random 16-byte UUID. Supported by SQL version 2015-10-08 and later.

Example: `newuuid()` = `123a4567-b89c-12d3-e456-789012345000`

numbytes(String)

Returns the number of bytes in the UTF-8 encoding of the provided string. Standard conversion rules apply to non-`String` arguments. Supported by SQL version 2016-03-23 and later.

Examples:

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long[, String])

Use the `parse_time` function to format a timestamp into a human-readable date/time format. Supported by SQL version 2016-03-23 and later. To convert a timestamp string into milliseconds, see time_to_epoch(String, String) (p. 478).

The `parse_time` function expects the following arguments:

pattern

(String) A date/time pattern that follows the ISO 8601 standard format. Specifically, the function supports Joda-Time formats.

timestamp

(Long) The time to be formatted in milliseconds since Unix epoch. See function timestamp() (p. 479).

timezone

(String) The time zone of the formatted date/time. The default is "UTC". The function supports Joda-Time time zones. This argument is optional.

Examples:

When this message is published to the topic 'A/B', the payload `{"ts": "1970.01.01 AD at 21:46:40 CST"}` is sent to the S3 bucket:

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time("yyyy.MM.dd G 'at' HH:mm:ss z", 100000000, "America/
Belize" ) as ts FROM 'A/B'",

        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

When this message is published to the topic 'A/B', a payload similar to `{"ts": "2017.06.09 AD at 17:19:46 UTC"}` (but with the current date/time) is sent to the S3 bucket:

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time("yyyy.MM.dd G 'at' HH:mm:ss z", timestamp() ) as ts FROM
 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

`parse_time()` can also be used as a substitution template. For example, when this message is published to the topic 'A/B', the payload is sent to the S3 bucket with key = "2017":

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT * FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": BUCKET_NAME,
                    "key": "${parse_time("yyyy", timestamp(), "UTC")}"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

power(Decimal, Decimal)

Returns the first argument raised to the second argument. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later. Supported by SQL version 2015-10-08 and later.

Example: `power(2, 5)` = 32.0.

Argument type 1	Argument type 2	Output
`Int/Decimal`	`Int/Decimal`	A `Decimal` (with dou raised to the second a
`Int/Decimal/String`	`Int/Decimal/String`	A `Decimal` (with dou raised to the second a are converted to deci converted to `Decima`
Other value	Other value	`Undefined`.

principal()

Returns the principal that the device uses for authentication, based on how the triggering message was published. The following table describes the principal returned for each publishing method and protocol.

How the message is published	Protocol	Credential type
MQTT client	MQTT	X.509 device certifica
AWS IoT console MQTT client	MQTT	IAM user or role
AWS CLI	HTTP	IAM user or role
AWS IoT Device SDK	MQTT	X.509 device certifica
AWS IoT Device SDK	MQTT over WebSocket	IAM user or role

The following examples show the different types of values that `principal()` can return:

- X.509 certificate thumbprint: `ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373`
- IAM role ID and session name: `ABCD1EFG3HIJK2LMNOP5:my-session-name`
- Returns a user ID: `ABCD1EFG3HIJK2LMNOP5`

rand()

Returns a pseudorandom, uniformly distributed double between 0.0 and 1.0. Supported by SQL version 2015-10-08 and later.

Example:

`rand()` = 0.8231909191640703

regexp_matches(String, String)

Returns true if the string (first argument) contains a match for the regular expression (second argument).

Example:

`regexp_matches("aaaa", "a{2,}")` = true.

`regexp_matches("aaaa", "b")` = false.

First argument:

Argument type	Result
`Int`	The `String` representation of the `Int`.
`Decimal`	The `String` representation of the `Decimal`.
`Boolean`	The `String` representation of the boolean ("true" or "false").
`String`	The `String`.

Argument type	Result
Array	The `String` representation of the `Array` (using standard conversion rules).
Object	The `String` representation of the Object (using standard conversion rules).
Null	`Undefined`.
Undefined	`Undefined`.

Second argument:

Must be a valid regex expression. Non-string types are converted to `String` using the standard conversion rules. Depending on the type, the resultant string might not be a valid regular expression. If the (converted) argument is not valid regex, the result is `Undefined`.

regexp_replace(String, String, String)

Replaces all occurrences of the second argument (regular expression) in the first argument with the third argument. Reference capture groups with "$". Supported by SQL version 2015-10-08 and later.

Example:

`regexp_replace("abcd", "bc", "x")` = "axd".

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac".

First argument:

Argument type	Result
`Int`	The `String` representation of the `Int`.
`Decimal`	The `String` representation of the `Decimal`.
`Boolean`	The `String` representation of the boolean ("true" or "false").
`String`	The source value.
Array	The `String` representation of the `Array` (using standard conversion rules).
Object	The `String` representation of the Object (using standard conversion rules).
Null	`Undefined`.
Undefined	`Undefined`.

Second argument:

Must be a valid regex expression. Non-string types are converted to `String` using the standard conversion rules. Depending on the type, the resultant string might not be a valid regular expression. If the (converted) argument is not a valid regex expression, the result is `Undefined`.

Third argument:

Must be a valid regex replacement string. (Can reference capture groups.) Non-string types are converted to `String` using the standard conversion rules. If the (converted) argument is not a valid regex replacement string, the result is `Undefined`.

regexp_substr(String, String)

Finds the first match of the second parameter (regex) in the first parameter. Reference capture groups with "$". Supported by SQL version 2015-10-08 and later.

Example:

`regexp_substr("hihihello", "hi")` = "hi"

`regexp_substr("hihihello", "(hi)*")` = "hihi"

First argument:

Argument type	Result
`Int`	The `String` representation of the `Int`.
`Decimal`	The `String` representation of the `Decimal`.
`Boolean`	The `String` representation of the boolean ("true" or "false").
`String`	The `String` argument.
Array	The `String` representation of the `Array` (using standard conversion rules).
Object	The `String` representation of the Object (using standard conversion rules).
Null	`Undefined`.
Undefined	`Undefined`.

Second argument:

Must be a valid regex expression. Non-string types are converted to `String` using the standard conversion rules. Depending on the type, the resultant string might not be a valid regular expression. If the (converted) argument is not a valid regex expression, the result is `Undefined`.

Third argument:

Must be a valid regex replacement string. (Can reference capture groups.) Non-string types are converted to `String` using the standard conversion rules. If the argument is not a valid regex replacement string, the result is `Undefined`.

remainder(Decimal, Decimal)

Returns the remainder of the division of the first argument by the second argument. Equivalent to mod(Decimal, Decimal) (p. 465). You can also use "%" as an infix operator for the same modulo functionality. Supported by SQL version 2015-10-08 and later.

Example: `remainder(8, 3)` = 2.

Left operand	Right operand	Output
`Int`	`Int`	`Int`, the first argume
`Int/Decimal`	`Int/Decimal`	`Decimal`, the first ar
`String/Int/Decimal`	`String/Int/Decimal`	If all strings convert t argument modulo th `Undefined`.
Other value	Other value	`Undefined`.

replace(String, String, String)

Replaces all occurrences of the second argument in the first argument with the third argument. Supported by SQL version 2015-10-08 and later.

Example:

`replace("abcd", "bc", "x")` = `"axd"`.

`replace("abcdabcd", "b", "x")` = `"axcdaxcd"`.

All arguments

Argument type	Result
`Int`	The `String` representation of the `Int`.
`Decimal`	The `String` representation of the `Decimal`.
`Boolean`	The `String` representation of the boolean ("true" or "false").
`String`	The source value.
Array	The `String` representation of the `Array` (using standard conversion rules).
Object	The `String` representation of the Object (using standard conversion rules).
Null	`Undefined`.
Undefined	`Undefined`.

rpad(String, Int)

Returns the string argument, padded on the right side with the number of spaces specified in the second argument. The `Int` argument must be between 0 and 1000. If the provided value is outside of this valid range, the argument is set to the nearest valid value (0 or 1000). Supported by SQL version 2015-10-08 and later.

Examples:

`rpad("hello", 2)` = `"hello  "`.

`rpad(1, 3)` = `"1   "`.

Argument type 1	Argument type 2	Result
`String`	`Int`	The `String` is padded on the right side with a number of spaces equal to the provided `Int`.
`String`	`Decimal`	The `Decimal` argument is rounded down to the nearest `Int` and the string is padded on the right side with a number of spaces equal to the provided `Int`.
`String`	`String`	The second argument is converted to a `Decimal`, which is rounded down to the nearest

Argument type 1	Argument type 2	Result
		`Int`. The `String` is padded on the right side with a number of spaces equal to the `Int` value.
Other value	`Int/Decimal/String`	The first value is converted to a `String` using the standard conversions, and the rpad function is applied on that `String`. If it cannot be converted, the result is `Undefined`.
Any value	Other value	`Undefined`.

round(Decimal)

Rounds the given `Decimal` to the nearest `Int`. If the `Decimal` is equidistant from two `Int` values (for example, 0.5), the `Decimal` is rounded up. Supported by SQL version 2015-10-08 and later.

Example: `Round(1.2)` = 1.

`Round(1.5)` = 2.

`Round(1.7)` = 2.

`Round(-1.1)` = -1.

`Round(-1.5)` = -2.

Argument type	Result
`Int`	The argument.
`Decimal`	`Decimal` is rounded down to the nearest `Int`.
`String`	`Decimal` is rounded down to the nearest `Int`. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Other value	`Undefined`.

rtrim(String)

Removes all trailing white space (tabs and spaces) from the provided `String`. Supported by SQL version 2015-10-08 and later.

Examples:

`rtrim(" h i ")` = " h i"

Argument type	Result
`Int`	The `String` representation of the `Int`.
`Decimal`	The `String` representation of the `Decimal`.
`Boolean`	The `String` representation of the boolean ("true" or "false").
Array	The `String` representation of the `Array` (using standard conversion rules).
Object	The `String` representation of the Object (using standard conversion rules).
Null	`Undefined`.
Undefined	`Undefined`

sign(Decimal)

Returns the sign of the given number. When the sign of the argument is positive, 1 is returned. When the sign of the argument is negative, -1 is returned. If the argument is 0, 0 is returned. Supported by SQL version 2015-10-08 and later.

Examples:

`sign(-7)` = -1.

`sign(0)` = 0.

`sign(13)` = 1.

Argument type	Result
`Int`	`Int`, the sign of the `Int` value.
`Decimal`	`Int`, the sign of the `Decimal` value.
`String`	`Int`, the sign of the `Decimal` value. The string is converted to a `Decimal` value, and the sign of the `Decimal` value is returned. If the `String` cannot be converted to a `Decimal`, the result is `Undefined`. Supported by SQL version 2015-10-08 and later.
Other value	`Undefined`.

sin(Decimal)

Returns the sine of a number in radians. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `sin(0)` = 0.0

Argument type	Result
`Int`	`Decimal` (with double precision), the sine of the argument.
`Decimal`	`Decimal` (with double precision), the sine of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the sine of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
`Undefined`	`Undefined`.

sinh(Decimal)

Returns the hyperbolic sine of a number. `Decimal` values are rounded to double precision before function application. The result is a `Decimal` value of double precision. Supported by SQL version 2015-10-08 and later.

Example: `sinh(2.3)` = 4.936961805545957

Argument type	Result
`Int`	`Decimal` (with double precision), the hyperbolic sine of the argument.
`Decimal`	`Decimal` (with double precision), the hyperbolic sine of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the hyperbolic sine of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

substring(String, Int[, Int])

Expects a `String` followed by one or two `Int` values. For a `String` and a single `Int` argument, this function returns the substring of the provided `String` from the provided `Int` index (0-based, inclusive) to the end of the `String`. For a `String` and two `Int` arguments, this function returns the substring of the provided `String` from the first `Int` index argument (0-based, inclusive) to the second `Int` index argument (0-based, exclusive). Indices that are less than zero are set to zero. Indices that are greater than the `String` length are set to the `String` length. For the three argument version, if the first index is greater than (or equal to) the second index, the result is the empty `String`.

If the arguments provided are not (*`String`*, *`Int`*), or (*`String`*, *`Int`*, *`Int`*), the standard conversions are applied to the arguments to attempt to convert them into the correct types. If the types cannot be converted, the result of the function is `Undefined`. Supported by SQL version 2015-10-08 and later.

Examples:

`substring("012345", 0)` = "012345".

`substring("012345", 2)` = "2345".

`substring("012345", 2.745)` = "2345".

`substring(123, 2)` = "3".

`substring("012345", -1)` = "012345".

`substring(true, 1.2)` = "rue".

`substring(false, -2.411E247)` = "false".

`substring("012345", 1, 3)` = "12".

`substring("012345", -50, 50)` = "012345".

`substring("012345", 3, 1)` = "".

sql_version()

Returns the SQL version specified in this rule. Supported by SQL version 2015-10-08 and later.

Example:

`sql_version()` = "2016-03-23"

sqrt(Decimal)

Returns the square root of a number. `Decimal` arguments are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `sqrt(9)` = 3.0.

Argument type	Result
`Int`	The square root of the argument.
`Decimal`	The square root of the argument.
`Boolean`	`Undefined`.
`String`	The square root of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

startswith(String, String)

Returns `Boolean`, whether the first string argument starts with the second string argument. If either argument is `Null` or `Undefined`, the result is `Undefined`. Supported by SQL version 2015-10-08 and later.

Example:

`startswith("ranger","ran")` = true

Argument type 1	Argument type 2	Result
`String`	`String`	Whether the first stri
Other value	Other value	Both arguments are c standard conversion r starts with the secon or `Undefined`, the re

tan(Decimal)

Returns the tangent of a number in radians. `Decimal` values are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `tan(3)` = -0.1425465430742778

Argument type	Result
`Int`	`Decimal` (with double precision), the tangent of the argument.
`Decimal`	`Decimal` (with double precision), the tangent of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the tangent of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

tanh(Decimal)

Returns the hyperbolic tangent of a number in radians. `Decimal` values are rounded to double precision before function application. Supported by SQL version 2015-10-08 and later.

Example: `tanh(2.3)` = 0.9800963962661914

Argument type	Result
`Int`	`Decimal` (with double precision), the hyperbolic tangent of the argument.
`Decimal`	`Decimal` (with double precision), the hyperbolic tangent of the argument.
`Boolean`	`Undefined`.
`String`	`Decimal` (with double precision), the hyperbolic tangent of the argument. If the string cannot be converted to a `Decimal`, the result is `Undefined`.
Array	`Undefined`.
Object	`Undefined`.
Null	`Undefined`.
Undefined	`Undefined`.

time_to_epoch(String, String)

Use the `time_to_epoch` function to convert a timestamp string into a number of milliseconds in Unix epoch time. Supported by SQL version 2016-03-23 and later. To convert milliseconds to a formatted timestamp string, see parse_time(String, Long[, String]) (p. 466).

The `time_to_epoch` function expects the following arguments:

timestamp

: (String) The timestamp string to be converted to milliseconds since Unix epoch. If the timestamp string doesn't specify a timezone, the function uses the UTC timezone.

pattern

: (String) A date/time pattern that follows the ISO 8601 standard format. Specifically, the function supports Joda-Time formats.

Examples:

`time_to_epoch("2020-04-03 09:45:18 UTC+01:00", "yyyy-MM-dd HH:mm:ss z")` = 1585903518000

`time_to_epoch("18 December 2015", "dd MMMM yyyy")` = 1450396800000

`time_to_epoch("2007-12-03 10:15:30.592 America/Los_Angeles", "yyyy-MM-dd HH:mm:ss.SSS z")` = 1196705730592

timestamp()

Returns the current timestamp in milliseconds from 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, as observed by the AWS IoT rules engine. Supported by SQL version 2015-10-08 and later.

Example: `timestamp()` = `1481825251155`

topic(Decimal)

Returns the topic to which the message that triggered the rule was sent. If no parameter is specified, the entire topic is returned. The `Decimal` parameter is used to specify a specific topic segment, with 1 designating the first segment. For the topic `foo/bar/baz`, topic(1) returns `foo`, topic(2) returns `bar`, and so on. Supported by SQL version 2015-10-08 and later.

Examples:

`topic()` = "things/myThings/thingOne"

`topic(1)` = "things"

When Basic Ingest (p. 429) is used, the initial prefix of the topic (`$aws/rules/`*`rule-name`*) is not available to the topic() function. For example, given the topic:

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2"

traceid()

Returns the trace ID (UUID) of the MQTT message, or `Undefined` if the message wasn't sent over MQTT. Supported by SQL version 2015-10-08 and later.

Example:

`traceid()` = "12345678-1234-1234-1234-123456789012"

transform(String, Object, Array)

Returns an array of objects that contains the result of the specified transformation of the `Object` parameter on the `Array` parameter.

Supported by SQL version 2016-03-23 and later.

String

The transformation mode to use. Refer to the following table for the supported transformation modes and how they create the `Result` from the `Object` and `Array` parameters.

Object

An object that contains the attributes to apply to each element of the `Array`.

Array

An array of objects into which the attributes of `Object` are applied.

Each object in this Array corresponds to an object in the function's response. Each object in the function's response contains the attributes present in the original object and the attributes provided by `Object` as determined by the transformation mode specified in `String`.

`String` parameter	**`Object` parameter**	**`Array` parameter**	**Result**
`enrichArray`	Object	Array of objects	An Array of objects in which each object contains the attributes of an element from the `Array` parameter and the attributes of the `Object` parameter.
Any other value	Any value	Any value	Undefined

Note
The array returned by this function is limited to 128 KiB.

Example:

In this example, the following message is published to the MQTT topic `A/B`.

```
{
  "foo": "example",
  "bar": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
  }
}
```

This SQL statement for a topic rule action uses the **transform()**. In this example, the `Object` for this transform function is the `foo` attribute from the message payload and the `Array` is the two objects of the `bar.c` array.

```
select transform('enrichArray', foo, bar.c) as example from 'A/B'
```

With the preceding message, the SQL statement evaluates to the following response.

```
{
  "example": [
    {
      "x": {
        "someInt": 5,
        "someString": "hello"
      },
      "y": true,
      "foo": "example"
    },
    {
      "x": {
        "someInt": 10,
        "someString": "world"
      },
      "y": false,
      "foo": "example"
    }
  ]
}
```

Nested SELECT clauses can also use this function to select multiple attributes. In this example, the `Object` for this transform function is the object returned by the SELECT statement, which contains the `a` and `b` elements of the message's `bar` object. The `Array` parameter consists of the two objects from the `bar.c` array in the original message.

```
select value transform('enrichArray', (select a, b from bar), (select value c from bar))
 from 'A/B'
```

With the preceding message, the SQL statement evaluates to the following response.

```
[
  {
    "x": {
      "someInt": 5,
      "someString": "hello"
    },
    "y": true,
    "a": "first attribute",
    "b": "second attribute"
  },
  {
    "x": {
      "someInt": 10,
      "someString": "world"
    },
    "y": false,
    "a": "first attribute",
    "b": "second attribute"
  }
]
```

The array returned in this response could be used with topic rule actions that support `batchMode`.

trim(String)

Removes all leading and trailing white space from the provided `String`. Supported by SQL version 2015-10-08 and later.

Example:

`Trim(" hi ")` = "hi"

Argument type	**Result**
`Int`	The `String` representation of the `Int` with all leading and trailing white space removed.
`Decimal`	The `String` representation of the `Decimal` with all leading and trailing white space removed.
`Boolean`	The `String` representation of the `Boolean` ("true" or "false") with all leading and trailing white space removed.
`String`	The `String` with all leading and trailing white space removed.
Array	The `String` representation of the `Array` using standard conversion rules.
Object	The `String` representation of the Object using standard conversion rules.
Null	`Undefined`.
Undefined	`Undefined`.

trunc(Decimal, Int)

Truncates the first argument to the number of `Decimal` places specified by the second argument. If the second argument is less than zero, it is set to zero. If the second argument is greater than 34, it is set to 34. Trailing zeroes are stripped from the result. Supported by SQL version 2015-10-08 and later.

Examples:

`trunc(2.3, 0)` = 2.

`trunc(2.3123, 2)` = 2.31.

`trunc(2.888, 2)` = 2.88.

`trunc(2.00, 5)` = 2.

Argument type 1	Argument type 2	Result
`Int`	`Int`	The source value.
`Int/Decimal`	`Int/Decimal`	The first argument is by the second argum `Int`, is rounded dow
`Int/Decimal/String`	`Int/Decimal`	The first argument is by the second argum `Int`, is rounded dow converted to a `Deci` fails, the result is `Und`
Other value		`Undefined`.

upper(String)

Returns the uppercase version of the given `String`. Non-`String` arguments are converted to `String` using the standard conversion rules. Supported by SQL version 2015-10-08 and later.

Examples:

`upper("hello")` = "HELLO"

`upper(["hello"])` = "[\"HELLO\"]"

Literals

You can directly specify literal objects in the SELECT and WHERE clauses of your rule SQL, which can be useful for passing information.

> **Note**
> Literals are available only when using SQL version 2016-03-23 or later.

JSON object syntax is used (key-value pairs, comma-separated, where keys are strings and values are JSON values, wrapped in curly brackets {}). For example:

Incoming payload published on topic `topic/subtopic`: `{"lat_long": [47.606,-122.332]}`

SQL statement: `SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'`

The resulting outgoing payload would be: `{"lat_long": {"latitude":47.606,"longitude":-122.332}}`.

You can also directly specify arrays in the SELECT and WHERE clauses of your rule SQL, which allows you to group information. JSON syntax is used (wrap comma-separated items in square brackets [] to create an array literal). For example:

Incoming payload published on topic `topic/subtopic`: `{"lat": 47.696, "long": -122.332}`

SQL statement: `SELECT [lat,long] as lat_long FROM 'topic/subtopic'`

The resulting output payload would be: `{"lat_long": [47.606,-122.332]}`.

Case statements

Case statements can be used for branching execution, like a switch statement, or if/else statements.

Syntax:

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

The expression `v` is evaluated and matched for equality against each `t[i]` expression. If a match is found, the corresponding `r[i]` expression becomes the result of the case statement. If there is more than one possible match, the first match is selected. If there are no matches, the else statement's `re` is used as the result. If there is no match and no else statement, the result of the case statement is `Undefined`. For example:

Incoming payload published on topic `topic/subtopic`: `{"color":"yellow"}`

SQL statement: `SELECT CASE color WHEN 'green' THEN 'go' WHEN 'yellow' THEN 'caution' WHEN 'red' THEN 'stop' ELSE 'you are not at a stop light' END as instructions FROM 'topic/subtopic'`

The resulting output payload would be: `{"instructions":"caution"}`.

Case statements require at least one WHEN clause. An ELSE clause is not required.

Note
If v is `Undefined`, the result of the case statement is `Undefined`.

JSON extensions

You can use the following extensions to ANSI SQL syntax to make it easier to work with nested JSON objects.

"." Operator

This operator accesses members in embedded JSON objects and functions identically to ANSI SQL and JavaScript. For example:

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

selects the value of the `bar` property in the `foo` object from the following message payload sent to the `topic/subtopic` topic.

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

If a JSON property name includes a hyphen character or numeric characters, the simple 'dot' notation will not work. Instead, you must use the get function (p. 456) to extract the property's value.

In this example the following message is sent to the `iot/rules` topic.

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

Normally, the value of `my-key` would be identified as in this query.

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

However, because the property name `my-key` contains a hyphen and `item2` contains a numeric character, the get function (p. 456) must be used as the following query shows.

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

`*` Operator

This functions in the same way as the `*` wildcard in ANSI SQL. It's used in the SELECT clause only and creates a new JSON object containing the message data. If the message payload is not in JSON format, `*` returns the entire message payload as raw bytes. For example:

```
SELECT * FROM 'topic/subtopic'
```

Applying a Function to an Attribute Value

The following is an example JSON payload that might be published by a device:

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
        "longitude" : -122.3359976
    }
}
```

The following example applies a function to an attribute value in a JSON payload:

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

The result of this query is the following JSON object:

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

Substitution templates

You can use a substitution template to augment the JSON data returned when a rule is triggered and AWS IoT performs an action. The syntax for a substitution template is `${`*expression*`}`, where *expression* can be any expression supported by AWS IoT in SELECT clauses, WHERE clauses, and AWS IoT rule actions (p. 359). This expression can be plugged into an action field on a rule, allowing you to dynamically configure an action. In effect, this feature substitutes a piece of information in an action. This includes functions, operators, and information present in the original message payload.

Important
Because an expression in a substitution template is evaluated separately from the "SELECT ..." statement, you cannot reference an alias created using the AS clause. You can reference only information present in the original payload, functions (p. 442), and operators (p. 437).

For more information about supported expressions, see AWS IoT SQL reference (p. 430).

The following rule actions support substitution templates. Each action supports different fields that can be substituted.

- CloudWatch alarms (p. 365)
- CloudWatch Logs (p. 366)
- CloudWatch metrics (p. 367)
- DynamoDB (p. 369)
- DynamoDBv2 (p. 371)
- Elasticsearch (p. 373)
- HTTPS (p. 374)
- IoT Analytics (p. 377)
- IoT Events (p. 378)
- IoT SiteWise (p. 380)
- Kinesis Data Streams (p. 385)
- Kinesis Data Firehose (p. 383)
- Lambda (p. 387)
- Republish (p. 389)
- S3 (p. 390)
- SNS (p. 392)
- SQS (p. 394)
- Step Functions (p. 395)
- Timestream (p. 396)

Substitution templates appear in the action parameters within a rule:

```
{
    "sql": "SELECT *, timestamp() AS timestamp FROM 'my/iot/topic'",
    "ruleDisabled": false,
    "actions": [{
        "republish": {
            "topic": "${topic()}/republish",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

If this rule is triggered by the following JSON published to `my/iot/topic`:

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

Then this rule publishes the following JSON to `my/iot/topic/republish`, which AWS IoT substitutes from `${topic()}/republish`:

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

Nested object queries

You can use nested SELECT clauses to query for attributes within arrays and inner JSON objects. Supported by SQL version 2016-03-23 and later.

Consider the following MQTT message:

```
{
    "e": [
        { "n": "temperature", "u": "Cel", "t": 1234, "v": 22.5 },
        { "n": "light", "u": "lm", "t": 1235, "v": 135 },
        { "n": "acidity", "u": "pH", "t": 1235, "v": 7 }
    ]
}
```

Example

You can convert values to a new array with the following rule.

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

The rule generates the following output.

```
{
    "sensors": [
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

Using the same MQTT message, you can also query a specific value within a nested object with the following rule.

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

The rule generates the following output.

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

You can also flatten the output with a more complicated rule.

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

The rule generates the following output.

```
{
    "temperature": 22.5
}
```

Working with binary payloads

When the message payload should be handled as raw binary data, rather than a JSON object, use the * operator to refer to it in a `SELECT` clause. This works for non-JSON payloads with some rule actions, such as the S3 action.

To use * to refer to the message payload as raw binary data, follow these rules:

1. The SQL statement and templates must not refer to JSON names other than *.
2. The `SELECT` statement must have * as the only item, or must have only functions. See the following example.

   ```
   SELECT * FROM 'topic/subtopic'
   ```

   ```
   SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'
   ```

For rule actions that don't support binary payload input, such as Lambda action, you must decode binary payloads. The Lambda rule action can receive binary data if it's base64 encoded and in a JSON payload. You can do this by changing the rule to:

```
SELECT encode(*, 'base64') AS data FROM 'my_topic'
```

Binary payload examples

You can use the following `SELECT` clause with binary payloads because it doesn't refer to any JSON names.

```
SELECT * FROM 'topic/subtopic'
```

You cannot use the following SELECT with binary payloads because it refers to device_type in the WHERE clause.

```
SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
```

You cannot use the following SELECT with binary payloads because it violates rule #2.

```
SELECT *, timestamp() AS timestamp FROM 'topic/subtopic'
```

You can use the following SELECT with binary payloads because it complies with rule #1 or rule #2.

```
SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0
```

You cannot use the following AWS IoT rule with binary payloads because it violates rule #1.

```
{
    "sql": "SELECT * FROM 'topic/subtopic'"
    "actions": [{
        "republish": {
            "topic":"device/${device_id}"
         }
     }]
}
```

SQL versions

The AWS IoT rules engine uses an SQL-like syntax to select data from MQTT messages. The SQL statements are interpreted based on an SQL version specified with the awsIotSqlVersion property in a JSON document that describes the rule. For more information about the structure of JSON rule documents, see Creating a Rule (p. 355). The awsIotSqlVersion property lets you specify which version of the AWS IoT SQL rules engine that you want to use. When a new version is deployed, you can continue to use an earlier version or change your rule to use the new version. Your current rules continue to use the version with which they were created.

The following JSON example shows you how to specify the SQL version using the awsIotSqlVersion property.

```
{
    "sql": "expression",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "republish": {
            "topic": "my-mqtt-topic",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

AWS IoT currently supports the following SQL versions:

- `2015-10-08` – The original SQL version built on 2015-10-08.
- `2016-03-23` – The SQL version built on 2016-03-23.
- `beta` – The most recent beta SQL version. If you use this version, it might introduce breaking changes to your rules.

What's new in the 2016-03-23 SQL rules engine version

- Fixes for selecting nested JSON objects.
- Fixes for array queries.
- Intra-object query support. For more information, see Nested object queries (p. 487).
- Support to output an array as a top-level object.
- Addition of the `encode(value, encodingScheme)` function, which can be applied on JSON and non-JSON format data. For more information, see the encode function (p. 455).

Output an `Array` as a top-level object

This feature allows a rule to return an array as a top-level object. For example, given the following MQTT message:

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

And the following rule:

```
SELECT VALUE arr FROM 'topic'
```

The rule generates the following output.

```
[1,2,3,4]
```

AWS IoT Device Shadow service

The AWS IoT Device Shadow service adds shadows to AWS IoT thing objects. Shadows can make a device's state available to apps and other services whether the device is connected to AWS IoT or not. AWS IoT thing objects can have multiple named shadows so that your IoT solution has more options for connecting your devices to other apps and services.

AWS IoT thing objects do not have any named shadows until they are created explicitly; however, an unnamed, classic shadow is created for a thing when the thing is created. Shadows can be created, updated, and deleted by using the AWS IoT console. Devices, other web clients, and services can create, update, and delete shadows by using MQTT and the reserved MQTT topics (p. 97), HTTP using the Device Shadow REST API (p. 515), and the AWS IoT CLI. Because shadows are stored by AWS in the cloud, they can collect and report device state data from apps and other cloud services whether the device is connected or not.

Topics

Using shadows

Shadows provide a reliable data store for devices, apps, and other cloud services to share data. They enable devices, apps, and other cloud services to connect and disconnect without losing a device's state.

While devices, apps, and other cloud services are connected to AWS IoT, they can access and control the current state of a device through its shadows. For example, an app can request a change in a device's state by updating a shadow. AWS IoT publishes a message that indicates the change to the device. The device receives this message, updates its state to match, and publishes a message with its updated state. The Device Shadow service reflects this updated state in the corresponding shadow. The app can subscribe to the shadow's update or it can query the shadow for its current state.

When a device goes offline, an app can still communicate with AWS IoT and the device's shadows. When the device reconnects, it receives the current state of its shadows so that it can update its state to match that of its shadows, and then publish a message with its updated state. Likewise, when an app goes offline and the device state changes while it's offline, the device keeps the shadow updated so the app can query the shadows for its current state when it reconnects.

Choosing to use named or unnamed shadows

The Device Shadow service supports named and unnamed, classic shadows, as have been used in the past. A thing object can have multiple named shadows, and no more than one unnamed, classic shadow. A thing object can have both named and unnamed shadows at the same time; however, the API used to access each is slightly different, so it might be more efficient to decide which type of shadow would work best for your solution and use that type only. For more information about the API to access the shadows, see Shadow topics (p. 97).

With named shadows, you can create different views of a thing object’s state. For example, you could divide a thing object with many properties into shadows with logical groups of properties, each identified by its shadow name. You could also limit access to properties by grouping them into different shadows and using policies to control access. Named shadows, however, do not support fleet indexing (p. 684).

The classic, unnamed shadows are simpler, but somewhat more limited than the named shadows. Each AWS IoT thing object can have only one unnamed shadow. If you expect your IoT solution to have a limited need for shadow data, this might be how you want to get started using shadows. However, if you think you might want to add additional shadows in the future, consider using named shadows from the start.

Accessing shadows

Every shadow has a reserved MQTT topic (p. 97) and HTTP URL (p. 515) that supports the `get`, `update`, and `delete` actions on the shadow.

Shadows use JSON shadow documents (p. 527) to store and retrieve data. A shadow’s document contains a state property that describes these aspects of the device’s state:

- `desired`

  Apps specify the desired states of device properties by updating the `desired` object.
- `reported`

  Devices report their current state in the `reported` object.
- `delta`

  AWS IoT reports differences between the desired and the reported state in the `delta` object.

The data stored in a shadow is determined by the state property of the update action's message body. Subsequent update actions can modify the values of an existing data object, and also add and delete keys and other elements in the shadow’s state object. For more information about accessing shadows, see Using shadows in devices (p. 494) and Using shadows in apps and services (p. 497).

Important
Permission to make update requests should be limited to trusted apps and devices. This prevents the shadow's state property from being changed unexpectedly; otherwise, the devices and apps that use the shadow should be designed to expect the keys in the state property to change.

Using shadows in devices, apps, and other cloud services

Using shadows in devices, apps, and other cloud services requires consistency and coordination between all of these. The AWS IoT Device Shadow service stores the shadow state, sends messages when the shadow state changes, and responds to messages that change its state. The devices, apps, and other cloud services in your IoT solution must manage their state and keep it consistent with the device shadow's state.

The shadow state data is dynamic and can be altered by the devices, apps, and other cloud services with permission to access the shadow. For this reason, it is important to consider how each device, app, and other cloud service will interact with the shadow. For example:

- *Devices* should write only to the `reported` property of the shadow state when communicating state data to the shadow.

- *Apps and other cloud services* should write only to the `desired` property when communicating state change requests to the device through the shadow.

Important
The data contained in a shadow data object is independent from that of other shadows and other thing object properties, such as a thing's attributes and the content of MQTT messages that a thing object's device might publish. A device can, however, report the same data in different MQTT topics and shadows if necessary.
A device that supports multiple shadows must maintain the consistency of the data that it reports in the different shadows.

Message order

There is no guarantee that messages from the AWS IoT service will arrive at the device in any specific order. The following scenario shows what happens in this case.

Initial state document:

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

Update 1:

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

Update 2:

```
{
  "state": {
    "desired": {
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

Final state document:

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

This results in two delta messages:

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
  "timestamp": 123456779
}
```

The device might receive these messages out of order. Because the state in these messages is cumulative, a device can safely discard any messages that contain a version number older than the one it is tracking. If the device receives the delta for version 12 before version 11, it can safely discard the version 11 message.

Trim shadow messages

To reduce the size of shadow messages sent to your device, define a rule that selects only the fields your device needs then republishes the message on an MQTT topic to which your device is listening.

The rule is specified in JSON and should look like the following:

```
{
  "sql": "SELECT state, version FROM '$aws/things/+/shadow/update/delta'",
  "ruleDisabled": false,
  "actions": [
    {
      "republish": {
        "topic": "${topic(3)}/delta",
        "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
      }
    }
  ]
}
```

The SELECT statement determines which fields from the message will be republished to the specified topic. A "+" wild card is used to match all shadow names. The rule specifies that all matching messages should be republished to the specified topic. In this case, the `"topic()"` function is used to specify the topic on which to republish. `topic(3)` evaluates to the thing name in the original topic. For more information about creating rules, see Rules for AWS IoT (p. 352).

Using shadows in devices

This section describes device communications with shadows using MQTT messages, the preferred method for devices to communicate with the AWS IoT Device Shadow service.

Shadow communications emulate a request/response model using the publish/subscribe communication model of MQTT. Every shadow action consists of a request topic, a successful response topic (`accepted`), and an error response topic (`rejected`).

If you want apps and services to be able to determine whether a device is connected, see Detecting a device is connected (p. 498).

Important
Because MQTT uses a publish/subscribe communication model, you should subscribe to the response topics *before* you publish a request topic. If you don't, you might not receive the response to the request that you publish.
If you use an AWS IoT Device SDK (p. 990) to call the Device Shadow service APIs, this is handled for you.

The examples in this section use an abbreviated form of the topic where the *`ShadowTopicPrefix`* can refer to either a named or an unnamed shadow, as described in this table.

Shadows can be named or unnamed (classic). The topics used by each differ only in the topic prefix. This table shows the topic prefix used by each shadow type.

`ShadowTopicPrefix` **value**	**Shadow type**
`$aws/things/`*`thingName`*`/shadow`	Unnamed (classic) shadow
`$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*	Named shadow

Important
Make sure that your app's or service's use of the shadows is consistent and supported by the corresponding implementations in your devices. For example, consider how shadows are created, updated, and deleted. Also consider how updates are handled in the device and the apps or services that access the device through a shadow. Your design should be clear about how the device's state is updated and reported and how your apps and services interact with the device and its shadows.

To create a complete topic, select the *`ShadowTopicPrefix`* for the type of shadow to which you want to refer, replace *`thingName`*, and *`shadowName`* if applicable, with their corresponding values, and then append that with the topic stub as shown in the following table. Remember that topics are case sensitive.

See Shadow topics (p. 97) for more information about the reserved topics for shadows.

Initializing the device on first connection to AWS IoT

After a device registers with AWS IoT, it should subscribe to these MQTT messages for the shadows that it supports.

Topic	**Meaning**	**Action a device should take when this topic is received**
`ShadowTopicPrefix``/delete/accepted`	The `delete` request was accepted and AWS IoT deleted the shadow.	The actions necessary to accommodate the deleted shadow, such as stop publishing updates.

Topic	Meaning	Action a device should take when this topic is received
ShadowTopicPrefix/delete/rejected	The `delete` request was rejected by AWS IoT and the shadow was not deleted. The message body contains the error information.	Respond to the error message in the message body.
ShadowTopicPrefix/get/accepted	The `get` request was accepted by AWS IoT, and the message body contains the current shadow document.	The actions necessary to process the state document in the message body.
ShadowTopicPrefix/get/rejected	The `get` request was rejected by AWS IoT, and the message body contains the error information.	Respond to the error message in the message body.
ShadowTopicPrefix/update/accepted	The `update` request was accepted by AWS IoT, and the message body contains the current shadow document.	Confirm the updated data in the message body matches the device state.
ShadowTopicPrefix/update/rejected	The `update` request was rejected by AWS IoT, and the message body contains the error information.	Respond to the error message in the message body.
ShadowTopicPrefix/update/delta	The shadow document was updated by a request to AWS IoT, and the message body contains the changes requested.	Update the device's state to match the desired state in the message body.
ShadowTopicPrefix/update/documents	An update to the shadow was recently completed, and the message body contains the current shadow document.	Confirm the updated state in the message body matches the device's state.

After subscribing to the messages in the preceding table for each shadow, the device should test to see if the shadows that it supports have already been created by publishing a `/get` topic to each shadow. If a `/get/accepted` message is received, the message body contains the shadow document, which the device can use to initialize its state. If a `/get/rejected` message is received, the shadow should be created by publishing an `/update` message with the current device state.

Processing messages while the device is connected to AWS IoT

While a device is connected to AWS IoT, it can receive **/update/delta** messages and should keep the device state matched to the changes in its shadows by:

1. Reading all **/update/delta** messages received and synchronizing the device state to match.
2. Publishing an **/update** message with a `reported` message body that has the device’s current state, whenever the device's state changes.

While a device is connected, it should publish these messages when indicated.

Indication	Topic	Payload
The device's state has changed.	*ShadowTopicPrefix*/update	A shadow document with the `reported` property.
The device might not be synchronized with the shadow.	*ShadowTopicPrefix*/get	(empty)
An action on the device indicates that a shadow will no longer be supported by the device, such as when the device is being remove or replaced	*ShadowTopicPrefix*/delete	(empty)

Processing messages when the device reconnects to AWS IoT

When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by:

1. Reading all **/update/delta** messages received and synchronizing the device state to match.
2. Publishing an **/update** message with a `reported` message body that has the device’s current state.

Using shadows in apps and services

This section describes how an app or service interacts with the AWS IoT Device Shadow service. This example assumes the app or service is interacting only with the shadow and, through the shadow, the device. This example doesn't include any management actions, such as creating or deleting shadows.

This example uses the AWS IoT Device Shadow service's REST API to interact with shadows. Unlike the example used in Using shadows in devices (p. 494), which uses a publish/subscribe communications model, this example uses the request/response communications model of the REST API. This means the app or service must make a request before it can receive a response from AWS IoT. A disadvantage of this model, however, is that it does not support notifications. If your app or service requires timely notifications of device state changes, consider the MQTT or MQTT over WSS protocols, which support the publish/subscribe communication model, as described in Using shadows in devices (p. 494).

Important
Make sure that your app's or service's use of the shadows is consistent with and supported by the corresponding implementations in your devices. Consider, for example, how shadows are created, updated, and deleted, and how updates are handled in the device and the apps or services that access the shadow. Your design should clearly specify how the device's state is updated and reported, and how your apps and services interact with the device and its shadows.

The REST API's URL for a named shadows is:

```
https://endpoint/things/thingName/shadow?name=shadowName
```

and for an unnamed shadow:

```
https://endpoint/things/thingName/shadow
```

where:

endpoint

The endpoint returned by the CLI command:

```
aws iot describe-endpoint --endpoint-type IOT:Data-ATS
```

thingName

The name of the thing object to which the shadow belongs

shadowName

The name of the named shadow. This parameter is not used with unnamed shadows.

Initializing the app or service on connection to AWS IoT

When the app first connects to AWS IoT, it should send an HTTP GET request to the URLs of the shadows it uses to get the current state of the shadows it's using. This allows it to sync the app or service to the shadow.

Processing state changes while the app or service is connected to AWS IoT

While the app or service is connected to AWS IoT, it can query the current state periodically by sending an HTTP GET request on the URLs of the shadows it uses.

When an end user interacts with the app or service to change the state of the device, the app or service can send an HTTP POST request to the URLs of the shadows it uses to update the `desired` state of the shadow. This request returns the change that was accepted, but you might have to poll the shadow by making HTTP GET requests until the device has updated the shadow with its new state.

Detecting a device is connected

To determine if a device is currently connected, include a `connected` property in the shadow document and use an MQTT Last Will and Testament (LWT) message to set the `connected` property to `false` if a device is disconnected due to an error.

Note
MQTT LWT messages sent to AWS IoT reserved topics (topics that begin with $) are ignored by the AWS IoT Device Shadow service. However, they are processed by subscribed clients and by the AWS IoT rules engine, so you will need to create an LWT message that is sent to a non-reserved topic and a rule that republishes the MQTT LWT message as a shadow update message to the shadow's reserved update topic, *`ShadowTopicPrefix`*`/update`.

To send the Device Shadow service an LWT message

1. Create a rule that republishes the MQTT LWT message on the reserved topic. The following example is a rule that listens for a messages on the `my/things/myLightBulb/update` topic and republishes it to `$aws/things/myLightBulb/shadow/update`.

```
{
    "rule": {
    "ruleDisabled": false,
    "sql": "SELECT * FROM 'my/things/myLightBulb/update'",
    "description": "Turn my/things/ into $aws/things/",
    "actions": [{
        "republish": {
            "topic": "$$aws/things/myLightBulb/shadow/update",
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
        }
    }]
    }
}
```

2. When the device connects to AWS IoT, it registers an LWT message to a non-reserved topic for the republish rule to recognize. In this example, that topic is `my/things/myLightBulb/update` and it sets the connected property to `false`.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. After connecting, the device publishes a message on its shadow update topic, `$aws/things/myLightBulb/shadow/update`, to report its current state, which includes setting its `connected` property to `true`.

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
    }
}
```

4. Before the device disconnects gracefully, it publishes a message on its shadow update topic, `$aws/things/myLightBulb/shadow/update`, to report its latest state, which include setting its `connected` property to `false`.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. If the device disconnects due to an error, the AWS IoT message broker publishes the device's LWT message on behalf of the device. The republish rule detects this message and publishes the shadow update message to update the `connected` property of the device shadow.

Simulating Device Shadow service communications

This topic demonstrates how the Device Shadow service acts as an intermediary and allows devices and apps to use a shadow to update, store, and retrieve a device's state.

To demonstrate the interaction described in this topic, and to explore it further, you'll need an AWS account and a system on which you can run the AWS CLI. If you don't have these, you can still see the interaction in the code examples.

In this example, the AWS IoT console represents the device. The AWS CLI represents the app or service that accesses the device by way of the shadow. The AWS CLI interface is very similar to the API that an app might use to communicate with AWS IoT. The device in this example is a smart light bulb and the app displays the light bulb's state and can change the light bulb's state.

Setting up the simulation

These procedures initialize the simulation by opening the AWS IoT console, which simulates your device, and the command line window that simulates your app.

To set up your simulation environment

1. Create an AWS account or, if you already have one to use for this simulation, you can skip this step.

   You'll need an AWS account to run the examples from this topic on your own. If you don't have an AWS account, create one, as described in Set up your AWS account (p. 18).
2. Open the AWS IoT console, and in the left menu, choose **Test** to open the **MQTT client**.
3. In another window, open a terminal window on a system that has the AWS CLI installed on it.

You should have two windows open: one with the AWS IoT console on the **Test** page, and one with a command line prompt.

Initialize the device

In this simulation, we'll be working with a thing object named, *mySimulatedThing*, and its shadow named, *simShadow1*.

In the console, subscribe to these MQTT topics. These topics are the responses to the `get`, `update`, and `delete` actions so that your device will be ready to receive the responses after it publishes an action.

- `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/rejected`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta`
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/documents`

Do this procedure for each of the MQTT topics in the preceding list.

To subscribe to an MQTT topic in the MQTT client

1. In the **MQTT client**, choose **Subscribe to topic** under **Subscriptions**
2. In **Subscription topic**, enter the topic that you want to subscribe to.

   For this simulation, copy a topic from the preceding list and paste it into **Subscription topic**. Enter only one topic at a time.
3. Choose **Subscribe to topic**.
4. Confirm the topics appear in the left column of the **MQTT client**.

   At this point, your simulated device is ready to receive the topics as they are published by AWS IoT.
5. After a device has initialized itself and subscribed to the response topics, it should query for the shadows it supports. This simulation supports only one shadow, the shadow that supports a thing object named, *mySimulatedThing*, named, *simShadow1*.

To get the current shadow state from the MQTT client

1. In the **MQTT client**, choose **Publish to a topic** under **Subscriptions**
2. Under **Publish**, enter this topic:

```
$aws/things/mySimulatedThing/shadow/name/simShadow1/get
```

3. Delete any content from the message body window below where you entered the topic to get.
4. Choose **Publish to topic** to publish the request.

6. If you receive a message in the `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`, topic and the `code` is `404`, such as in this example, the shadow has not been created, so we'll create it next.

```
{
  "code": 404,
  "message": "No shadow exists with name: 'simShadow1'"
}
```

To create a shadow with the current status of the device

1. In the **MQTT client**, choose **Publish to a topic** under **Subscriptions**
2. Under **Publish**, enter this topic:

```
$aws/things/mySimulatedThing/shadow/name/simShadow1/update
```

3. In the message body window below where you entered the topic, enter this shadow document to show the device is reporting its ID and its current color in RGB values.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

4. Choose **Publish to topic** to publish the request.
5. Observe which of the topics receives the response message.
6. If you receive a message in the `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted` topic, the shadow was created and the message body contains the current shadow document, such as the example in Step 7.
7. If you receive a message in the `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected` topic, review the error in the message body.

7. If you received a message in the `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`, topic, the shadow already exists and the message body has the current shadow state, such as in this example. With this, you could set your device or confirm that it matches the shadow state.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591140517,
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

Send an update from the app

This section uses the AWS CLI to demonstrate how an app can interact with a shadow.

To get the current state of the shadow using the AWS CLI

1. From the command line, enter this command.

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1  /
dev/stdout
```

2. Because the shadow exists and had been initialized by the device to reflect its current state, it should return the following shadow document.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

The app can use this response to initialize its representation of the device state.

If the app updates the state, such as when an end user changes the color of our smart light bulb to yellow, the app would send an **update-thing-shadow** command. This command corresponds to the `UpdateThingShadow` REST API.

To update a shadow from an app

1. From the command line, enter this command.

   AWS CLI v2.x

   ```
   aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 \
       --cli-binary-format raw-in-base64-out \
       --payload '{"state":{"desired":{"ColorRGB":
   [255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
   ```

   AWS CLI v1.x

   ```
   aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 \
       --payload '{"state":{"desired":{"ColorRGB":
   [255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
   ```

2. If successful, this command should return the following shadow document.

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
```

```
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    }
  },
  "version": 4,
  "timestamp": 1591141596,
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

Respond to update in device

Returning to the **MQTT client** in the AWS console, you should see the messages that AWS IoT published to reflect the update command issued in the previous section.

To view the update messages in the MQTT client

1. In the **MQTT client**, choose **$aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta** in the **Subscriptions** column. If the topic name is truncated, you can pause on it to see the full topic.
2. In the **$aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta** topic log, you should see a `/delta` message similar as this one.

```
{
  "version": 4,
  "timestamp": 1591141596,
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

Your device would process the contents of this message to set the device state to match the `desired` state in the message.

After the device updates the state to match the `desired` state in the message, it must send the new reported state back to AWS IoT by publishing an update message. This procedure simulates this in the **MQTT client**.

To update the shadow from the device

1. In the **MQTT client**, choose **Publish to a topic**.
2. In the message body window of the **Publish** section, enter this updated shadow document, which describes the current state of the device.

```
{
  "state": {
    "reported": {
      "ColorRGB": [255,255,0]
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

3. In the **Publish** section, in the topic field above the message body window, enter the shadow's topic followed by the `/update` action.

```
$aws/things/mySimulatedThing/shadow/name/simShadow1/update
```

4. Choose **Publish to topic** to publish the updated device state.
5. If the message was successfully received by AWS IoT, you should see a new response in the **$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted** message log in the **MQTT client** with the current state of the shadow, such as this example.

```
{
  "state": {
    "reported": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "reported": {
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

A successful update to the reported state of the device also causes AWS IoT to send a comprehensive description of the shadow state in a message to the topic, such as this message body that resulted from the shadow update performed by the device in the preceding procedure.

```
{
  "previous": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          128,
          128,
          128
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          }
        ]
      }
    },
    "version": 4
  },
  "current": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          255,
          255,
          0
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          }
        ]
      }
    },
    "version": 5
  },
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

Observe the update in the app

The app can now query the shadow for the current state as reported by the device.

To get the current state of the shadow using the AWS CLI

1. From the command line, enter this command.

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1  /
   dev/stdout
   ```

2. Because the shadow has just been updated by the device to reflect its current state, it should return the following shadow document.

   ```
   {
     "state": {
   ```

```
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    },
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    },
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591143269
}
```

Going beyond the simulation

Experiment with the interaction between the AWS CLI (representing the app) and the console (representing the device) to model your IoT solution.

Interacting with shadows

This topic describes the messages associated with each of the three methods that AWS IoT provides for working with shadows. These methods include the following:

`UPDATE`

Creates a shadow if it doesn't exist, or updates the contents of an existing shadow with the state information provided in the message body. AWS IoT records a timestamp with each update to indicate when the state was last updated. When the shadow's state changes, AWS IoT sends `/delta` messages to all MQTT subscribers with the difference between the `desired` and the `reported` states. Devices or apps that receive a `/delta` message can perform actions based on the difference. For example, a device can update its state to the desired state, or an app can update its UI to reflect the device's state change.

`GET`

Retrieves a current shadow document that contains the complete state of the shadow, including metadata.

`DELETE`

Deletes the shadow and all of its content. You can't restore a deleted shadow, but you can create a new shadow with the same name. Note that deleting a shadow does not reset its version number to 0.

Protocol support

AWS IoT supports MQTT and a REST API over HTTPS protocols to interact with shadows. AWS IoT provides a set of reserved request and response topics for MQTT publish and subscribe actions. Devices and apps should subscribe to the response topics before publishing a request topic for information about how AWS IoT handled the request. For more information, see Device Shadow MQTT topics (p. 519) and Device Shadow REST API (p. 515).

Requesting and reporting state

When designing your IoT solution using AWS IoT and shadows, you should determine the apps or devices that will request changes and those that will implement them. Typically, a device implements and reports changes back to the shadow and apps and services respond to and request changes in the shadow. Your solution could be different, but the examples in this topic assume that the client app or service requests changes in the shadow and the device performs the changes and reports them back to the shadow.

Updating a shadow

Your app or service can update a shadow's state by using the UpdateThingShadow (p. 516) API or by publishing to the /update (p. 521) topic. Updates affect only the fields specified in the request.

Updating a shadow when a client requests a state change

When a client requests a state change in a shadow by using the MQTT protocol

1. The client should have a current shadow document so that it can identify the properties to change. See the /get action for how to obtain the current shadow document.
2. The client subscribes to these MQTT topics:
   - `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*`/update/accepted`
   - `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*`/update/rejected`
   - `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*`/update/delta`
   - `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*`/update/documents`

3. The client publishes a `$aws/things/thingName/shadow/name/shadowName/update` request topic with a state document that contains the desired state of the shadow. Only the properties to change need to be included in the document. This is an example of a document with the desired state.

```
{
  "state": {
    "desired": {
      "color": {
        "r": 10
      },
      "engine": "ON"
    }
  }
}
```

4. If the update request is valid, AWS IoT updates the desired state in the shadow and publishes messages on these topics:
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`

   The `/update/accepted` message contains an /accepted response state document (p. 528) shadow document, and the `/update/delta` message contains a /delta response state document (p. 528) shadow document.
5. If the update request is not valid, AWS IoT publishes a message with the `$aws/things/thingName/shadow/name/shadowName/update/rejected` topic with an Error response document (p. 531) shadow document that describes the error.

When a client requests a state change in a shadow by using the API

1. The client calls the `UpdateThingShadow (p. 516)` API with a Request state document (p. 527) state document as its message body.
2. If the request was valid, AWS IoT returns an HTTP success response code and an /accepted response state document (p. 528) shadow document as its response message body.

   AWS IoT will also publish an MQTT message to the `$aws/things/thingName/shadow/name/shadowName/update/delta` topic with a /delta response state document (p. 528) shadow document for any devices or clients that subscribe to it.
3. If the request was not valid, AWS IoT returns an HTTP error response code an Error response document (p. 531) as its response message body.

When the device receives the `/desired` state on the `/update/delta` topic, it makes the desired changes in the device. It then sends a message to the `/update` topic to report its current state to the shadow.

Updating a shadow when a device reports its current state

When a device reports its current state to the shadow by using the MQTT protocol

1. The device should subscribe to these MQTT topics before updating the shadow:
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`

- $aws/things/*thingName*/shadow/name/*shadowName*/update/documents

2. The device reports its current state by publishing a message to the $aws/things/*thingName*/shadow/name/*shadowName*/update topic that reports the current state, such as in this example.

```
{
    "state": {
        "reported" : {
            "color" : { "r" : 10 },
            "engine" : "ON"
        }
    }
}
```

3. If AWS IoT accepts the update, it publishes a message to the $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted topics with an /accepted response state document (p. 528) shadow document.
4. If the update request is not valid, AWS IoT publishes a message with the $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected topic with an Error response document (p. 531) shadow document that describes the error.

When a device reports its current state to the shadow by using the API

1. The device calls the UpdateThingShadow (p. 516) API with a Request state document (p. 527) state document as its message body.
2. If the request was valid, AWS IoT updates the shadow and returns an HTTP success response code with an /accepted response state document (p. 528) shadow document as its response message body.

   AWS IoT will also publish an MQTT message to the $aws/things/*thingName*/shadow/name/*shadowName*/update/delta topic with a /delta response state document (p. 528) shadow document for any devices or clients that subscribe to it.
3. If the request was not valid, AWS IoT returns an HTTP error response code an Error response document (p. 531) as its response message body.

Optimistic locking

You can use the state document version to ensure you are updating the most recent version of a device's shadow document. When you supply a version with an update request, the service rejects the request with an HTTP 409 conflict response code if the current version of the state document does not match the version supplied.

For example:

Initial document:

```
{
  "state": {
    "desired": {
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

Update: (version doesn't match; this request will be rejected)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

Result:

```
{
  "code": 409,
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

Update: (version matches; this request will be accepted)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

Final state:

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
}
```

Retrieving a shadow document

You can retrieve a shadow document by using the GetThingShadow (p. 515) API or by subscribing and publishing to the /get (p. 520) topic. This retrieves a complete shadow document, including any delta between the `desired` and `reported` states. The procedure for this task is the same whether the device or a client is making the request.

To retrieve a shadow document by using the MQTT protocol

1. The device or client should subscribe to these MQTT topics before updating the shadow:

   - `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*`/get/accepted`

  - `$aws/things/thingName/shadow/name/shadowName/get/rejected`
2. The device or client publishes a message to the `$aws/things/thingName/shadow/name/shadowName/get` topic with an empty message body.
3. If the request is successful, AWS IoT publishes a message to the `$aws/things/thingName/shadow/name/shadowName/get/accepted` topic with a /accepted response state document (p. 528) in the message body.
4. If the request was not valid, AWS IoT publishes a message to the `$aws/things/thingName/shadow/name/shadowName/get/rejected` topic with an Error response document (p. 531) in the message body.

To retrieve a shadow document by using a REST API

1. The device or client call the `GetThingShadow (p. 515)` API with an empty message body.
2. If the request is valid, AWS IoT returns an HTTP success response code with an /accepted response state document (p. 528) shadow document as its response message body.
3. If the request is not valid, AWS IoT returns an HTTP error response code an Error response document (p. 531) as its response message body.

Deleting shadow data

There are two ways to delete shadow data: you can delete specific properties in the shadow document and you can delete the shadow completely.

- To delete specific properties from a shadow, update the shadow; however set the value of the properties that you want to delete to `null`. Fields with a value of `null` are removed from the shadow document.
- To delete the entire shadow, use the DeleteThingShadow (p. 517) API or publish to the /delete (p. 525) topic.

Note that deleting a shadow does not reset its version number to 0.

Deleting a property from a shadow document

To delete a property from a shadow by using the MQTT protocol

1. The device or client should have a current shadow document so that it can identify the properties to change. See Retrieving a shadow document (p. 512) for information on how to obtain the current shadow document.
2. The device or client subscribes to these MQTT topics:
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
3. The device or client publishes a `$aws/things/thingName/shadow/name/shadowName/update` request topic with a state document that assigns `null` values to the properties of the shadow to delete. Only the properties to change need to be included in the document. This is an example of a document that deletes the `engine` property.

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
}
```

4. If the update request is valid, AWS IoT deletes the specified properties in the shadow and publishes a messages with the `$aws/things/thingName/shadow/name/shadowName/update/accepted` topic with an /accepted response state document (p. 528) shadow document in the message body.
5. If the update request is not valid, AWS IoT publishes a message with the `$aws/things/thingName/shadow/name/shadowName/update/rejected` topic with an Error response document (p. 531) shadow document that describes the error.

To delete a property from a shadow by using the REST API

1. The device or client calls the `UpdateThingShadow (p. 516)` API with a Request state document (p. 527) that assigns `null` values to the properties of the shadow to delete. Include only the properties that you want to delete in the document. This is an example of a document that deletes the `engine` property.

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
}
```

2. If the request was valid, AWS IoT returns an HTTP success response code and an /accepted response state document (p. 528) shadow document as its response message body.
3. If the request was not valid, AWS IoT returns an HTTP error response code an Error response document (p. 531) as its response message body.

Deleting a shadow

Note
Setting the device's shadow state to `null` does not delete the shadow. The shadow version will be incremented on the next update.
Deleting a device's shadow does not delete the thing object. Deleting a thing object does not delete the corresponding device's shadow.
Deleting a shadow does not reset its version number to 0.

To delete a shadow by using the MQTT protocol

1. The device or client subscribes to these MQTT topics:
   - `$aws/things/thingName/shadow/name/shadowName/delete/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/delete/rejected`
2. The device or client publishes a `$aws/things/thingName/shadow/name/shadowName/delete` with an empty message buffer.
3. If the delete request is valid, AWS IoT deletes the shadow and publishes a messages with the `$aws/things/thingName/shadow/name/shadowName/delete/accepted` topic and an abbreviated /accepted response state document (p. 528) shadow document in the message body. This is an example of the accepted delete message:

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

4. If the update request is not valid, AWS IoT publishes a message with the `$aws/things/thingName/shadow/name/shadowName/delete/rejected` topic with an Error response document (p. 531) shadow document that describes the error.

To delete a shadow by using the REST API

1. The device or client calls the `DeleteThingShadow (p. 517)` API with an empty message buffer.
2. If the request was valid, AWS IoT returns an HTTP success response code and an /accepted response state document (p. 528) and an abbreviated /accepted response state document (p. 528) shadow document in the message body. This is an example of the accepted delete message:

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. If the request was not valid, AWS IoT returns an HTTP error response code an Error response document (p. 531) as its response message body.

Device Shadow REST API

A shadow exposes the following URI for updating state information:

```
https://endpoint/things/thingName/shadow
```

The endpoint is specific to your AWS account. To find your endpoint, you can:

- Use the describe-endpoint command from the AWS CLI.
- Use the AWS IoT console settings. In **Settings**, the endpoint is listed under **Custom endpoint**
- Use the AWS IoT console thing details. Open **Manage**. Under **Manage**, choose **Things** and, in the list of things, select and open a thing. In the left nav of the Thing detail page, choose **Interact** and view the endpoint URI in the **HTTPS** section of the page.

The format of the endpoint is as follows:

```
identifier.iot.region.amazonaws.com
```

The shadow REST API follows the same HTTPS protocols/port mappings as described in Device communication protocols (p. 76).

API actions

- GetThingShadow (p. 515)
- UpdateThingShadow (p. 516)
- DeleteThingShadow (p. 517)
- ListNamedShadowsForThing (p. 518)

GetThingShadow

Gets the shadow for the specified thing.

The response state document includes the delta between the `desired` and `reported` states.

Request

The request includes the standard HTTP headers plus the following URI:

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

The `name` query parameter is not required for unnamed (classic) shadows.

Response

Upon success, the response includes the standard HTTP headers plus the following code and body:

```
HTTP 200
Response Body: response state document
```

For more information, see Example Response State Document (p. 528).

Authorization

Retrieving a shadow requires a policy that allows the caller to perform the `iot:GetThingShadow` action. The Device Shadow service accepts two forms of authentication: Signature Version 4 with IAM credentials or TLS mutual authentication with a client certificate.

The following is an example policy that allows a caller to retrieve a device's shadow:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:GetThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

UpdateThingShadow

Updates the shadow for the specified thing.

Updates affect only the fields specified in the request state document. Any field with a value of `null` is removed from the device's shadow.

Request

The request includes the standard HTTP headers plus the following URI and body:

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

The `name` query parameter is not required for unnamed (classic) shadows.

For more information, see Example Request State Document (p. 527).

Response

Upon success, the response includes the standard HTTP headers plus the following code and body:

```
HTTP 200
Response body: response state document
```

For more information, see Example Response State Document (p. 528).

Authorization

Updating a shadow requires a policy that allows the caller to perform the `iot:UpdateThingShadow` action. The Device Shadow service accepts two forms of authentication: Signature Version 4 with IAM credentials or TLS mutual authentication with a client certificate.

The following is an example policy that allows a caller to update a device's shadow:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:UpdateThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

DeleteThingShadow

Deletes the shadow for the specified thing.

Request

The request includes the standard HTTP headers plus the following URI:

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

The `name` query parameter is not required for unnamed (classic) shadows.

Response

Upon success, the response includes the standard HTTP headers plus the following code and body:

```
HTTP 200
Response body: Empty response state document
```

Note that deleting a shadow does not reset its version number to 0.

Authorization

Deleting a device's shadow requires a policy that allows the caller to perform the `iot:DeleteThingShadow` action. The Device Shadow service accepts two forms of authentication: Signature Version 4 with IAM credentials or TLS mutual authentication with a client certificate.

The following is an example policy that allows a caller to delete a device's shadow:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:DeleteThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

ListNamedShadowsForThing

Lists the shadows for the specified thing.

Request

The request includes the standard HTTP headers plus the following URI:

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

The token to retrieve the next set of results.

This value is returned on paged results and is used in the call that returns the next page.

pageSize

The number of shadow names to return in each call. See also `nextToken`.

thingName

The name of the thing for which to list the named shadows.

Response

Upon success, the response includes the standard HTTP headers plus the following response code and a Shadow name list response document (p. 531)

Note
The unnamed (classic) shadow does not appear in this list.

```
HTTP 200
Response body: Shadow name list document
```

Authorization

Deleting a device's shadow requires a policy that allows the caller to perform the `iot:ListNamedShadowsForThing` action. The Device Shadow service accepts two forms of authentication: Signature Version 4 with IAM credentials or TLS mutual authentication with a client certificate.

The following is an example policy that allows a caller to list a thing's named shadows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:ListNamedShadowsForThing",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

Device Shadow MQTT topics

The Device Shadow service uses reserved MQTT topics to enable devices and apps to get, update, or delete the state information for a device (shadow).

Publishing and subscribing on shadow topics requires topic-based authorization. AWS IoT reserves the right to add new topics to the existing topic structure. For this reason, we recommend that you avoid wild card subscriptions to shadow topics. For example, avoid subscribing to topic filters like `$aws/things/thingName/shadow/#` because the number of topics that match this topic filter might increase as AWS IoT introduces new shadow topics. For examples of the messages published on these topics see Interacting with shadows (p. 508).

Shadows can be named or unnamed (classic). The topics used by each differ only in the topic prefix. This table shows the topic prefix used by each shadow type.

ShadowTopicPrefix **value**	**Shadow type**
`$aws/things/`*`thingName`*`/shadow`	Unnamed (classic) shadow
`$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*	Named shadow

To create a complete topic, select the *`ShadowTopicPrefix`* for the type of shadow to which you want to refer, replace *`thingName`*, and *`shadowName`* if applicable, with their corresponding values, and then append that with the topic stub as shown in the following sections.

The following are the MQTT topics used for interacting with shadows.

Topics

/get

Publish an empty message to this topic to get the device's shadow:

```
ShadowTopicPrefix/get
```

AWS IoT responds by publishing to either /get/accepted (p. 520) or /get/rejected (p. 521).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get"
      ]
    }
  ]
}
```

/get/accepted

AWS IoT publishes a response shadow document to this topic when returning the device's shadow:

```
ShadowTopicPrefix/get/accepted
```

For more information, see Response state documents (p. 528).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get/accepted"
      ]
    }
  ]
}
```

/get/rejected

AWS IoT publishes an error response document to this topic when it can't return the device's shadow:

```
ShadowTopicPrefix/get/rejected
```

For more information, see Error response document (p. 531).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/rejected"
      ]
    },
    {
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get/rejected"
      ]
    }
  ]
}
```

/update

Publish a request state document to this topic to update the device's shadow:

```
ShadowTopicPrefix/update
```

The message body contains a partial request state document (p. 527).

A client attempting to update the state of a device would send a JSON request state document with the `desired` property such as this:

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

A device updating its shadow would send a JSON request state document with the `reported` property, such as this:

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT responds by publishing to either /update/accepted (p. 523) or /update/rejected (p. 524).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update"
      ]
    }
  ]
}
```

/update/delta

AWS IoT publishes a response state document to this topic when it accepts a change for the device's shadow, and the request state document contains different values for `desired` and `reported` states:

```
ShadowTopicPrefix/update/delta
```

The message buffer contains a /delta response state document (p. 528).

Message body details

- A message published on `update/delta` includes only the desired attributes that differ between the `desired` and `reported` sections. It contains all of these attributes, regardless of whether these attributes were contained in the current update message or were already stored in AWS IoT. Attributes that do not differ between the `desired` and `reported` sections are not included.
- If an attribute is in the `reported` section but has no equivalent in the `desired` section, it is not included.
- If an attribute is in the `desired` section but has no equivalent in the `reported` section, it is included.
- If an attribute is deleted from the `reported` section but still exists in the `desired` section, it is included.

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/delta"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/delta"
      ]
    }
  ]
}
```

/update/accepted

AWS IoT publishes a response state document to this topic when it accepts a change for the device's shadow:

```
ShadowTopicPrefix/update/accepted
```

The message buffer contains a /accepted response state document (p. 528).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/accepted"
      ]
    }
  ]
}
```

/update/documents

AWS IoT publishes a state document to this topic whenever an update to the shadow is successfully performed:

```
ShadowTopicPrefix/update/documents
```

The message body contains a /documents response state document (p. 529).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
documents"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/documents"
      ]
    }
  ]
}
```

/update/rejected

AWS IoT publishes an error response document to this topic when it rejects a change for the device's shadow:

```
ShadowTopicPrefix/update/rejected
```

The message body contains an Error response document (p. 531).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/rejected"
      ]
    }
  ]
}
```

/delete

To delete a device's shadow, publish an empty message to the delete topic:

```
ShadowTopicPrefix/shadow/delete
```

The content of the message is ignored.

Note that deleting a shadow does not reset its version number to 0.

AWS IoT responds by publishing to either /delete/accepted (p. 525) or /delete/rejected (p. 526).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete"
      ]
    }
  ]
}
```

/delete/accepted

AWS IoT publishes a message to this topic when a device's shadow is deleted:

```
ShadowTopicPrefix/delete/accepted
```

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/delete/
accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete/accepted"
      ]
    }
  ]
}
```

/delete/rejected

AWS IoT publishes an error response document to this topic when it can't delete the device's shadow:

```
ShadowTopicPrefix/delete/rejected
```

The message body contains an Error response document (p. 531).

Example policy

The following is an example of the required policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/delete/
rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete/rejected"
      ]
    }
  ]
}
```

Device Shadow service documents

The Device Shadow service respects all rules of the JSON specification. Values, objects, and arrays are stored in the device's shadow document.

Contents

Shadow document examples

The Device Shadow service uses these documents in UPDATE, GET, and DELETE operations using the REST API (p. 515) or MQTT Pub/Sub Messages (p. 519).

Examples

Request state document

A request state document has the following format:

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state` — Updates affect only the fields specified. Typically, you'll use either the `desired` or the `reported` property, but not both in the same request.
  - `desired` — The state properties and values requested to be updated in the device.
  - `reported` — The state properties and values reported by the device.
- `clientToken` — If used, you can match the request and corresponding response by the client token.
- `version` — If used, the Device Shadow service processes the update only if the specified version matches the latest version it has.

Response state documents

Response state documents have the following format depending on the response type.

/accepted response state document

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/delta response state document

```
{
    "state": {
        "attribute1": integer2,
        "attribute2": "string2",
        ...
        "attributeN": boolean2
        }
    },
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/documents response state document

```
{
  "previous" : {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version-1
  },
  "current": {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version
  },
  "timestamp": timestamp,
  "clientToken": "token"
}
```

Response state document properties

- previous — After a successful update, contains the state of the object before the update.
- current — After a successful update, contains the state of the object after the update.
- state
  - reported — Present only if a thing reported any data in the reported section and contains only fields that were in the request state document.
  - desired — Present only if a device reported any data in the desired section and contains only fields that were in the request state document.
  - delta — Present only if the desired data differs from the shadow's current reported data.
- metadata — Contains the timestamps for each attribute in the desired and reported sections so that you can determine when the state was updated.
- timestamp — The Epoch date and time the response was generated by AWS IoT.
- clientToken — Present only if a client token was used when publishing valid JSON to the /update topic.
- version — The current version of the document for the device's shadow shared in AWS IoT. It is increased by one over the previous version of the document.

Error response document

An error response document has the following format:

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- `code` — An HTTP response code that indicates the type of error.
- `message` — A text message that provides additional information.
- `timestamp` — The date and time the response was generated by AWS IoT. This property is not present in all error response documents.
- `clientToken` — Present only if a client token was used in the published message.

For more information, see Device Shadow error messages (p. 535).

Shadow name list response document

A shadow name list response document has the following format:

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- `results` — The array of shadow names.
- `nextToken` — The token value to use in paged requests to get the next page in the sequence. This property is not present when there are no more shadow names to return.
- `timestamp` — The date and time the response was generated by AWS IoT.

Document properties

A device's shadow document has the following properties:

`state`

`desired`

The desired state of the device. Apps can write to this portion of the document to update the state of a device directly without having to connect to it.

`reported`

The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state.

`metadata`

Information about the data stored in the `state` section of the document. This includes timestamps, in Epoch time, for each attribute in the `state` section, which enables you to determine when they were updated.

Note
Metadata do not contribute to the document size for service limits or pricing. For more information, see AWS IoT Service Limits.

`timestamp`

Indicates when the message was sent by AWS IoT. By using the timestamp in the message and the timestamps for individual attributes in the `desired` or `reported` section, a device can determine a property's age, even if the device doesn't have an internal clock.

`clientToken`

A string unique to the device that enables you to associate responses with requests in an MQTT environment.

`version`

The document version. Every time the document is updated, this version number is incremented. Used to ensure the version of the document being updated is the most recent.

For more information, see Shadow document examples (p. 527).

Delta state

Delta state is a virtual type of state that contains the difference between the `desired` and `reported` states. Fields in the `desired` section that are not in the `reported` section are included in the delta. Fields that are in the `reported` section and not in the `desired` section are not included in the delta. The delta contains metadata, and its values are equal to the metadata in the `desired` field. For example:

```
{
  "state": {
    "desired": {
      "color": "RED",
      "state": "STOP"
    },
    "reported": {
      "color": "GREEN",
      "engine": "ON"
    },
    "delta": {
      "color": "RED",
      "state": "STOP"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
      "state": {
        "timestamp": 12345
      },
      "reported": {
        "color": {
          "timestamp": 12345
        },
        "engine": {
          "timestamp": 12345
        }
      },
      "delta": {
        "color": {
          "timestamp": 12345
        },
        "state": {
          "timestamp": 12345
        }
      }
    },
    "version": 17,
    "timestamp": 123456789
  }
}
```

When nested objects differ, the delta contains the path all the way to the root.

```
{
  "state": {
    "desired": {
      "lights": {
        "color": {
          "r": 255,
          "g": 255,
          "b": 255
        }
      }
    },
    "reported": {
      "lights": {
        "color": {
          "r": 255,
          "g": 0,
          "b": 255
        }
      }
    },
    "delta": {
      "lights": {
        "color": {
          "g": 255
        }
      }
    }
  },
  "version": 18,
  "timestamp": 123456789
}
```

The Device Shadow service calculates the delta by iterating through each field in the `desired` state and comparing it to the `reported` state.

Arrays are treated like values. If an array in the `desired` section doesn't match the array in the `reported` section, then the entire desired array is copied into the delta.

Versioning shadow documents

The Device Shadow service supports versioning on every update message, both request and response. This means that with every update of a shadow, the version of the JSON document is incremented. This ensures two things:

- A client can receive an error if it attempts to overwrite a shadow using an older version number. The client is informed it must resync before it can update a device's shadow.
- A client can decide not to act on a received message if the message has a lower version than the version stored by the client.

A client can bypass version matching by not including a version in the shadow document.

Client tokens in shadow documents

You can use a client token with MQTT-based messaging to verify the same client token is contained in a request and request response. This ensures the response and request are associated.

Note
The client token can be no longer than 64 bytes. A client token that is longer than 64 bytes causes a 400 (Bad Request) response and an *Invalid clientToken* error message.

Empty shadow document properties

The `reported` and `desired` properties in a shadow document can be empty or omitted when they don't apply to the current shadow state. For example, a shadow document contains a `desired` property only if it has a desired state. The following is a valid example of a state document with no `desired` property:

```
{
    "reported" : { "temp": 55 }
}
```

The `reported` property can also be empty, such as if the shadow has not been updated by the device:

```
{
    "desired" : { "color" : "RED" }
}
```

If an update causes the `desired` or `reported` properties to become null, it is removed from the document. The following shows how to remove the `desired` property by setting it to `null`. You might do this when a device updates its state, for example.

```
{
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

A shadow document can also have neither `desired` or `reported` properties, making the shadow document empty. This is an example of an empty, yet valid shadow document.

```
{
}
```

Array values in shadow documents

Shadows support arrays, but treat them as normal values in that an update to an array replaces the whole array. It is not possible to update part of an array.

Initial state:

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

Update:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

Final state:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

Arrays can't have null values. For example, the following array is not valid and will be rejected.

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

Device Shadow error messages

The Device Shadow service publishes a message on the error topic (over MQTT) when an attempt to change the state document fails. This message is only emitted as a response to a publish request on one of the reserved $aws topics. If the client updates the document using the REST API, then it receives the HTTP error code as part of its response, and no MQTT error messages are emitted.

HTTP error code	Error messages
400 (Bad Request)	• Invalid JSON • Missing required node: state • State node must be an object • Desired node must be an object • Reported node must be an object • Invalid version • Invalid clientToken **Note** A client token that is longer than 64 bytes will cause this response. • JSON contains too many levels of nesting; maximum is 6 • State contains an invalid node
401 (Unauthorized)	• Unauthorized
403 (Forbidden)	• Forbidden

HTTP error code	Error messages
404 (Not Found)	• Thing not found • No shadow exists with name: *`shadowName`*
409 (Conflict)	• Version conflict
413 (Payload Too Large)	• The payload exceeds the maximum size allowed
415 (Unsupported Media Type)	• Unsupported documented encoding; supported encoding is UTF-8
429 (Too Many Requests)	• The Device Shadow service will generate this error message when there are more than 10 in-flight requests on a single connection.
500 (Internal Server Error)	• Internal service failure

Jobs

AWS IoT jobs can be used to define a set of remote operations that are sent to and executed on one or more devices connected to AWS IoT.

> **Tip**
> For job document examples, see the jobs-agent.js example in the AWS IoT SDK for JavaScript.

Jobs key concepts

job

A job is a remote operation that is sent to and executed on one or more devices connected to AWS IoT. For example, you can define a job that instructs a set of devices to download and install application or firmware updates, reboot, rotate certificates, or perform remote troubleshooting operations.

job document

To create a job, you must first create a job document that is a description of the remote operations to be performed by the devices.

Job documents are UTF-8 encoded JSON documents and should contain information that your devices need to perform a job. A job document contains one or more URLs where the device can download an update or some other data. The job document can be stored in an Amazon S3 bucket, or be included inline with the command that creates the job.

target

When you create a job, you specify a list of targets that are the devices that should perform the operations. The targets can be things or thing groups (p. 179) or both. The AWS IoT Jobs service sends a message to each target to inform it that a job is available.

job execution

A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target. The Jobs service provides commands to track the progress of a job execution on a target and the progress of a job across all targets.

snapshot job

By default, a job is sent to all targets that you specify when you create the job. After those targets complete the job (or report that they are unable to do so), the job is complete.

continuous job

A continuous job is sent to all targets that you specify when you create the job. It continues to run and is sent to any new devices (things) that are added to the target group. For example, a continuous job can be used to onboard or upgrade devices as they are added to a group. You can make a job continuous by setting an optional parameter when you create the job.

rollouts

You can specify how quickly targets are notified of a pending job execution. This allows you to create a staged rollout to better manage updates, reboots, and other operations.

The following field can be added to the `CreateJob` request to specify the maximum number of job targets to inform per minute. This example sets a static rollout rate.

```
    "jobExecutionRolloutConfig": {
        "maximumPerMinute": "integer"
    }
```

You can also set a variable rollout rate with the `exponentialRate` field. The following example creates a rollout that has an exponential rate.

```
  "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": integer,
         "incrementFactor": integer,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": integer, // Set one or the other
            "numberOfSucceededThings": integer // of these two values.
         },
         "maximumPerMinute": integer
    }
}
```

For more information about configuring job rollouts, see Job Rollout and Abort Configuration.

abort

You can create a set of conditions to abort rollouts when criteria that you specify have been met. For more information, see Job Rollout and Abort Configuration.

presigned URLs

To allow a device secure, time-limited access to data beyond that included in the job document itself, you can use presigned Amazon S3 URLs. You can place your data in an Amazon S3 bucket and add a placeholder link to the data in the job document. When the Jobs service receives a request for the job document, it parses the job document looking for placeholder links and it replaces them with presigned Amazon S3 URLs.

The placeholder link is of the following form:

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

where *`bucket`* is your bucket name and *`key`* is the object in the bucket to which you are linking.

In the Beijing and Ningxia Regions, presigned URLs work only if the resource owner has an ICP license. For more information, see Amazon Simple Storage Service in the *Getting Started with AWS Services in China* documentation.

timeouts

Job timeouts make it possible to be notified whenever a job execution gets stuck in the `IN_PROGRESS` state for an unexpectedly long period of time. There are two types of timers: in-progress timers and step timers.

When you create a job, you can set a value for the `inProgressTimeoutInMinutes` property of the optional TimeoutConfig object. The in-progress timer can't be updated and applies to all job executions for the job. Whenever a job execution remains in the `IN_PROGRESS` status for longer than this interval, the job execution fails and switches to the terminal `TIMED_OUT` status. AWS IoT also publishes an MQTT notification.

You can also set a step timer for a job execution by setting a value for `stepTimeoutInMinutes` when you call UpdateJobExecution. The step timer applies only to the job execution that you update. You can set a new value for this timer each time you update a job execution. You can also create a step timer when you call StartNextPendingJobExecution. If the job execution remains in the `IN_PROGRESS` status for longer than the step timer interval, it fails and switches to the terminal `TIMED_OUT` status. The step timer has no effect on the in-progress timer that you set when you create a job.

The following diagram and description illustrate the ways in which in-progress timeouts and step timeouts interact with each other.

Job Creation: `CreateJob` sets an in-progress timer that expires in twenty minutes. This timer applies to all job executions and can't be updated.

12:00 PM: The job execution starts and switches to `IN_PROGRESS` status. The in-progress timer starts to run.

12:05 PM: `UpdateJobExecution` creates a step timer with a value of 7 minutes. If a new step timer isn't created, the job execution times out at 12:12 PM.

12:10 PM: `UpdateJobExecution` creates a new step timer with a value of 5 minutes. The previous step timer is discarded. If a new step timer isn't created, the job execution times out at 12:15 PM.

12:13 PM: `UpdateJobExecution` creates a new step timer with a value of 9 minutes. The job execution times out at 12:20 because the in-progress timer expires at 12:20. The step timer can't exceed the absolute bound created by the in-progress timer.

`UpdateJobExecution` can also discard a step timer that has already been created by creating a new step timer with a value of -1.

Managing jobs

You can use the AWS IoT console, the Jobs HTTPS API, the AWS Command Line Interface, or the AWS SDKs to create and manage jobs. For more information, see Job management and control API (p. 561), AWS CLI Command Reference: iot or AWS SDKs and Tools.

The primary purpose of jobs is to notify devices of a software or firmware update. When sending code to devices, the best practice is to sign the code file. This allows devices to detect if the code has been modified in transit. The instructions in the following section are written with the assumption that you want to code-sign the software update you are sending to your devices.

For more information, see What Is Code Signing for AWS IoT?

Before you create a job, you must create a job document. If you are using Code-signing for AWS IoT, you must upload your job document to a versioned Amazon S3 bucket. For more information about creating an Amazon S3 bucket and uploading files to it, see Getting Started with Amazon Simple Storage Service in the *Amazon S3 Getting Started Guide*.

Your job document can contain a presigned Amazon S3 URL that points to your code file (or other file). Presigned Amazon S3 URLs are valid for a limited amount of time and so are not generated until a device requests a job document. Because the presigned URL has not been created when you are creating the job document, you put a placeholder URL in your job document instead.

A placeholder URL looks like the following: `${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/<bucket>/<code file>}` where *bucket* is the Amazon S3 bucket that contains the code file and *code file* is the Amazon S3 key of the code file.

When a device requests the job document, AWS IoT generates the presigned URL and replaces the placeholder URL with the presigned URL. Your job document is then sent to the device.

When you create a job that uses presigned Amazon S3 URLs, you must provide an IAM role that grants permission to download files from the Amazon S3 bucket where the data or updates are stored. The role must also grant permission for AWS IoT to assume the role.

You can specify an optional timeout for the presigned URL. For more information, see CreateJob (p. 569).

To grant the jobs service permission to assume your role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search for your role and choose it.
4. On the **Trust Relationships** tab, choose **Edit Trust Relationship**.
5. On the **Edit Trust Relationship** page, replace the policy document with the following JSON:

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

6. Choose **Update Trust Policy**.
7. If your job uses a job document that is an Amazon S3 object, choose **Permissions** and with the following JSON, add a policy that grants permission to download files from your Amazon S3 bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
    ]
}
```

Creating and managing jobs (console)

If you are using Code-signing for AWS IoT, you must add two placeholder URLs in your job document:

A placeholder for the code file should look like this: `${aws:iot:s3-presigned-url:https://s3.amazonaws.com/<my-s3-bucket>/<my-code-file>}`.

> **Note**
> Currently, versions are not supported for presigned URL placeholders for jobs. If you update your code file and copy it to the same Amazon S3 location, you must create a new signature and then reference the new signature version in your job document.

A placeholder for the signature should look like this: `${aws:iot:code-sign-signature:s3://<region>.<my-s3-bucket>/<my-code-file>@<code-file-version-id>}`.

To create a job

1. Browse to the AWS IoT console.
2. In the navigation pane, choose **Manage**, and then choose **Jobs**.
3. Choose **Create a job**.
4. Choose **Create a custom job**.
5. Enter an alphanumeric ID for your job and an optional description.

   > **Note**
   > We do not recommend using personally identifiable information in your job IDs or descriptions.
6. Select the device or device groups that you want to update.
7. Under **Add a job file**, choose **Select**, and then select your job document.
8. Choose **Sign image for me**. If you are not code signing your update, you can skip this step.
9. Create or choose a code-signing profile. If you are not code signing your update, you can skip this step.
10. Under **Pre-sign resource URLs**, choose **I want to pre-sign my URLs and have configured my job file**. If you are not code signing your update, you can skip this step.
11. Choose a role and an expiry time for the presigned URL.
12. Under **Job type**, choose the appropriate option for your update, and then choose **Next**.
13. Specify values for any advanced configurations, and then choose **Create**.

After you create the job, the console generates a JSON signature and places it in your job document.

You can use the AWS IoT console to view the status, cancel, or delete a job.

1. Browse to the AWS IoT console.
2. In the navigation pane, choose **Manage**, and then choose **Jobs**.

Creating and managing jobs (CLI)

This section describes how to create and manage jobs.

Create jobs

You use the **CreateJob** command to create an AWS IoT job. The job is queued for execution on the targets (things or thing groups) that you specify. To create an AWS IoT job, you need a job document that can be included in the body of the request or as a link to an Amazon S3 document. If the job includes downloading files using presigned Amazon S3 URLs, you need an IAM role ARN that has permission to download the file and grants permission to the AWS IoT Jobs service to assume the role.

Code-signing with jobs

If you are using Code-signing for AWS IoT, you must start a code-signing job and include the output in your job document. Use the start-signing-job command to create a code-signing job. `start-signing-job` returns a job ID. Use the **describe-signing-job** command to get the Amazon S3 location where the signature is stored. You can then download the signature from Amazon S3. For more information about code signing jobs, see Code-signing for AWS IoT.

Your job document must contain a presigned URL placeholder for your code file and the JSON signature output placed in an Amazon S3 bucket using the **start-signing-job** command, enclosed in a `codesign` element:

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/image}",
    "codesign": {
        "rawPayloadSize": <image-file-size>,
        "signature": <signature>,
        "signatureAlgorithm": <signature-algorithm>,
        "payloadLocation": {
            "s3": {
                "bucketName": <my-s3-bucket>,
                "key": <my-code-file>,
                "version": <code-file-version-id>
            }
        }
    }
}
```

Creating a job with a job document

The following command shows how to create a job using a job document (*`job-document.json`*) stored in an Amazon S3 bucket (*`jobBucket`*) and a role with permission to download files from Amazon S3 (*`S3DownloadRole`*).

```
aws iot create-job  \
      --job-id 010  \
      --targets arn:aws:iot:us-east-1:123456789012:thing/thingOne  \
      --document-source https://s3.amazonaws.com/my-s3-bucket/job-document.json  \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType\":
 \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20}, { \"action
\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings\": 200,
 \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

The job is executed on *`thingOne`*.

The optional `timeout-config` parameter specifies the amount of time each device has to finish its execution of the job. The timer starts when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.

The in-progress timer can't be updated and applies to all job executions for the job. Whenever a job execution remains in the `IN_PROGRESS` state for longer than this interval, the job execution fails and switches to the terminal `TIMED_OUT` status. AWS IoT also publishes an MQTT notification.

For more information about creating configurations about job rollouts and aborts, see Job Rollout and Abort Configuration.

Note
Job documents that are specified as Amazon S3 files are retrieved at the time you create the job. Changing the contents of the Amazon S3 file you used as the source of your job document after you have created the job does not change what is sent to the targets of the job.

Update a job

You use the **UpdateJob** command to update a job. You can update the `description`, `presignedUrlConfig`, `jobExecutionsRolloutConfig`, `abortConfig`, and `timeoutConfig` fields of a job.

```
aws iot update-job  \
  --job-id 010  \
  --description "updated description" \
  --timeout-config inProgressTimeoutInMinutes=100 \
  --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}, \"maximumPerMinute\": 1000}}" \
  --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType\":
 \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20}, { \"action
\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings\": 200,
 \"thresholdPercentage\": 50}]}" \
  --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/S3DownloadRole\",
 \"expiresInSec\":3600}"
```

For more information, see Job Rollout and Abort Configuration.

Cancel a job

You use the **CancelJob** command to cancel a job. Canceling a job stops AWS IoT from rolling out any new job executions for the job. It also cancels any job executions that are in a `QUEUED` state. AWS IoT leaves any job executions in a terminal state untouched because the device has already completed the job. If the status of a job execution is `IN_PROGRESS`, it also remains untouched unless you use the optional `--force` parameter.

The following command shows how to cancel a job with ID 010.

```
aws iot cancel-job --job-id 010
```

The command displays the following output:

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

When you cancel a job, job executions that are in a `QUEUED` state are canceled. Job executions that are in an `IN_PROGRESS` state are canceled if you specify the optional `--force` parameter. Job executions in a terminal state are not canceled.

Warning
Canceling a job that is in the `IN_PROGRESS` state (by setting the `--force` parameter) cancels any job executions that are in progress and causes the device that is executing the job to be unable to update the job execution status. Use caution and make sure that each device executing a canceled job can recover to a valid state.

The status of a canceled job or of one of its job executions is eventually consistent. AWS IoT stops scheduling new job executions and `QUEUED` job executions for that job to devices as soon as possible. Changing the status of a job execution to `CANCELED` might take some time, depending on the number of devices and other factors.

If a job is canceled because it has met the criteria defined by an `AbortConfig` object, the service adds auto-populated values for the `comment` and `reasonCode` fields. You can create your own values for `reasonCode` when the job cancellation is user-driven.

Cancel a job execution

You use the **CancelJobExecution** command to cancel a job execution on a device. It cancels a job execution that is in a `QUEUED` state. If you want to cancel a job execution that is in progress, you must use the `--force` parameter.

The following command shows how to cancel the job execution from job 010 running on `myThing`.

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

The command displays no output.

A job execution that is in a `QUEUED` state is canceled. A job execution that is in an `IN_PROGRESS` state is canceled if you specify the optional `--force` parameter. Job executions in a terminal state cannot be canceled.

> **Warning**
> When you cancel a job execution that is in the `IN_PROGRESS` state, the device cannot update the job execution status. Use caution and ensure that the device can recover to a valid state.

If the job execution is in a terminal state or if the job execution is in an `IN_PROGRESS` state and the `--force` parameter is not set to `true`, this command causes an `InvalidStateTransitionException`.

The status of a canceled job execution is eventually consistent. Changing the status of a job execution to `CANCELED` might take some time, depending various factors.

Delete a job

You use the **DeleteJob** command to delete a job and its job executions. By default, you can only delete a job that is in a terminal state (`SUCCEEDED` or `CANCELED`). Otherwise, an exception occurs. You can delete a job in the `IN_PROGRESS` state if the `force` parameter is set to `true`.

Run the following command to delete a job:

```
aws iot delete-job --job-id 010 --force|--no-force
```

The command displays no output.

> **Warning**
> When you delete a job that is in the `IN_PROGRESS` state, the device that is executing the job cannot access job information or update the job execution status. Use caution and make sure that each device executing a job that has been deleted can recover to a valid state.

It can take some time to delete a job, depending on the number of job executions created for the job and other factors. While the job is being deleted, `DELETION_IN_PROGRESS` appears as the status of the job. An error results if you attempt to delete or cancel a job whose status is already `DELETION_IN_PROGRESS`.

Only 10 jobs can have a status of `DELETION_IN_PROGRESS` at the same time. Otherwise, a `LimitExceededException` occurs.

Get a job document

You use the **GetJobDocument** command to retrieve a job document for a job. A job document is a description of the remote operations to be performed by the devices.

Run the following command to get a job document:

```
aws iot get-job-document --job-id 010
```

The command returns the job document for the specified job:

```
{
    "document": "{\n\t\"operation\":\"install\",\n\t\"url\":\"http://amazon.com/
firmWareUpate-01\",\n\t\"data\":\"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/job-
test-bucket/datafile}\"\n}"
}
```

> **Note**
> When you use this command to retrieve a job document, placeholder URLs are not replaced by presigned Amazon S3 URLs. When a device calls the GetPendingJobExecutions (p. 616) MQTT API, the placeholder URLs are replaced by presigned Amazon S3 URLs in the job document.

List jobs

You use the **ListJobs** command to get a list of all jobs in your AWS account. Job data and job execution data are retained for a limited time. Run the following command to list all jobs in your AWS account:

```
aws iot list-jobs
```

The command returns all jobs in your account, sorted by the job status:

```
{
    "jobs": [
        {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486687079.743,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/013",
            "createdAt": 1486687079.743,
            "targetSelection": "SNAPSHOT",
            "jobId": "013"
        },
        {
            "status": "SUCCEEDED",
            "lastUpdatedAt": 1486685868.444,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/012",
            "createdAt": 1486685868.444,
            "completedAt": 148668789.690,
            "targetSelection": "SNAPSHOT",
            "jobId": "012"
        },
        {
            "status": "CANCELED",
            "lastUpdatedAt": 1486678850.575,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/011",
            "createdAt": 1486678850.575,
            "targetSelection": "SNAPSHOT",
            "jobId": "011"
        }
    ]
}
```

Describe a job

Run the **DescribeJob** command to get the status of a job. The following command shows how to describe a job:

```
$ aws iot describe-job --job-id 010
```

The command returns the status of the specified job. For example:

```
{
    "documentSource": "https://s3.amazonaws.com/job-test-bucket/job-document.json",
    "job": {
        "status": "IN_PROGRESS",
        "jobArn": "arn:aws:iot:us-east-1:123456789012:job/010",
        "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/myThing"
        ],
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfFailedThings": 0,
            "numberOfInProgressThings": 0,
            "numberOfQueuedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfTimedOutThings": 0,
            "processingTargets": [
                arn:aws:iot:us-east-1:123456789012:thing/thingOne,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupOne,
                arn:aws:iot:us-east-1:123456789012:thing/thingTwo,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupTwo
            ]
        },
        "presignedUrlConfig": {
            "expiresInSec": 60,
            "roleArn": "arn:aws:iam::123456789012:role/S3DownloadRole"
        },
        "jobId": "010",
        "lastUpdatedAt": 1486593195.006,
        "createdAt": 1486593195.006,
        "targetSelection": "SNAPSHOT",
        "jobExecutionsRolloutConfig": {
            "exponentialRate": {
                "baseRatePerMinute": integer,
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                    "numberOfNotifiedThings": integer, // Set one or the other
                    "numberOfSucceededThings": integer // of these two values.
                },
            "maximumPerMinute": integer
         }
        },
        "abortConfig": {
            "criteriaList": [
                {
                    "action": "string",
                    "failureType": "string",
                    "minNumberOfExecutedThings": integer,
                    "thresholdPercentage": integer
                }
            ]
        },
        "timeoutConfig": {
           "inProgressTimeoutInMinutes": number
          }
    }
}
```

List executions for a job

A job running on a specific device is represented by a job execution object. Run the **ListJobExecutionsForJob** command to list all job executions for a job. The following shows how to list the executions for a job:

```
aws iot list-job-executions-for-job --job-id 010
```

The command returns a list of job executions:

```
{
    "executionSummaries": [
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 1234567890
        }
    },
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingTwo",
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486593345.659,
            "queuedAt": 1486593196.378,
            "startedAt": 1486593345.659,
            "executionNumber": 4567890123
        }
    }
    ]
}
```

List job executions for a thing

Run the **ListJobExecutionsForThing** command to list all job executions running on a thing. The following shows how to list job executions for a thing:

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

The command returns a list of job executions that are running or have run on the specified thing:

```
{
    "executionSummaries": [
    {
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486687082.071,
            "queuedAt": 1486687082.071,
            "executionNumber": 9876543210
        },
        "jobId": "013"
    },
    {
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "startAt": 1486685870.729,
            "lastUpdatedAt": 1486685870.729,
            "queuedAt": 1486685870.729,
            "executionNumber": 1357924680
        },
        "jobId": "012"
    },
    {
        "jobExecutionSummary": {
            "status": "SUCCEEDED",
            "startAt": 1486678853.415,
            "lastUpdatedAt": 1486678853.415,
            "queuedAt": 1486678853.415,
            "executionNumber": 4357680912
        },
        "jobId": "011"
    },
    {
        "jobExecutionSummary": {
            "status": "CANCELED",
            "startAt": 1486593196.378,
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 2143174250
        },
        "jobId": "010"
    }
    ]
}
```

Describe job execution

Run the **DescribeJobExecution** command to get the status of a job execution. You must specify a job ID and thing name and, optionally, an execution number to identify the job execution. The following shows how to describe a job execution:

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

The command returns the JobExecution (p. 613). For example:

```
{
    "execution": {
        "jobId": "017",
        "executionNumber": 4516820379,
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "versionNumber": 123,
        "createdAt": 1489084805.285,
        "lastUpdatedAt": 1489086279.937,
        "startedAt": 1489086279.937,
        "status": "IN_PROGRESS",
        "approximateSecondsBeforeTimedOut": 100,
        "statusDetails": {
            "status": "IN_PROGRESS",
            "detailsMap": {
                "percentComplete": "10"
            }
        }
    }
}
```

Delete job execution

Run the **DeleteJobExecution** command to delete a job execution. You must specify a job ID, a thing name, and an execution number to identify the job execution. The following shows how to delete a job execution:

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

The command displays no output.

By default, the status of the job execution must be `QUEUED` or in a terminal state (`SUCCEEDED`, `FAILED`, `REJECTED`, `TIMED_OUT`, `REMOVED` or `CANCELED`). Otherwise, an error occurs. To delete a job execution with a status of `IN_PROGRESS`, you can set the `force` parameter to `true`.

Warning
When you delete a job execution with a status of `IN_PROGRESS`, the device that is executing the job cannot access job information or update the job execution status. Use caution and make sure that the device can recover to a valid state.

Devices and jobs

Device communication with jobs

Devices can communicate with the AWS IoT Jobs service through these methods:

- MQTT
- HTTP Signature Version 4
- HTTP TLS

Using the MQTT protocol

Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic. For more information about publishing and subscribing to MQTT topics, see the section called “Device communication protocols” (p. 76).

Note
You must use the correct endpoint when you communicate with the AWS IoT Jobs service through MQTT. Use the **DescribeEndpoint** command to find it. For example, if you run this command:

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

you get a result similar to the following:

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

With this method, your device uses its device-specific certificate and private key to authenticate with the AWS IoT Jobs service.

Devices can:

- Be notified when a job execution is added or removed from the list of pending job executions by subscribing to the `$aws/things/thing-name/jobs/notify` MQTT topic, where `thing-name` is the name of the thing associated with the device.
- Be notified when the next pending job execution has changed by subscribing to the `$aws/things/thing-name/jobs/notify-next` MQTT topic, where `thing-name` is the name of the thing associated with the device.
- Update the status of a job execution by calling the UpdateJobExecution (p. 629) API.
- Query the status of a job execution by calling the DescribeJobExecution (p. 625) API.
- Retrieve a list of pending job executions by calling the GetPendingJobExecutions (p. 616) API.
- Retrieve the next pending job execution by calling the DescribeJobExecution (p. 625) API with jobId `$next`.
- Get and start the next pending job execution by calling the StartNextPendingJobExecution (p. 620) API.

The AWS IoT Jobs service publishes success and failure messages on an MQTT topic. The topic is formed by appending `accepted` or `rejected` to the topic used to make the request. For example, if a request message is published on the `$aws/things/myThing/jobs/get` topic, the AWS IoT Jobs service publishes success messages on the `$aws/things/myThing/jobs/get/accepted` topic and publishes rejected messages on the `$aws/things/myThing/jobs/get/rejected` topic.

Using HTTP Signature Version 4

Communication between the AWS IoT Jobs service and your devices can occur over HTTP Signature Version 4 on port 443. This is the method used by the AWS SDKs and CLI. For more information about those tools, see AWS CLI Command Reference: iot-jobs-data or AWS SDKs and Tools and refer to the IotJobsDataPlane section for your preferred language.

Note
You must use the correct endpoint when you communicate with the AWS IoT Jobs service through HTTP Signature Version 4 or using an AWS SDK or CLI **IotJobsDataPlane** command. Use the **DescribeEndpoint** command to find it. For example, if you run this command:

```
aws iot describe-endpoint --endpoint-type iot:Jobs
```

you get a result similar to the following:

```
{
    "endpointAddress": "a1b2c3d4e5f6g7.jobs.iot.us-west-2.amazonaws.com"
}
```

With this method of communication, your device uses IAM credentials to authenticate with the AWS IoT Jobs service.

The following commands are available using this method:

- **DescribeJobExecution**

  ```
  aws iot-jobs-data describe-job-execution ...
  ```
- **GetPendingJobExecutions**

  ```
  aws iot-jobs-data get-pending-job-executions ...
  ```
- **StartNextPendingJobExecution**

  ```
  aws iot-jobs-data start-next-pending-job-execution ...
  ```
- **UpdateJobExecution**

  ```
  aws iot-jobs-data update-job-execution ...
  ```

Using HTTP TLS

Communication between the AWS IoT Jobs service and your devices can occur over HTTP TLS on port 8443 using a third-party software client that supports this protocol.

Note
You must use the correct endpoint when you communicate with the AWS IoT Jobs service through HTTP TLS. Use the **DescribeEndpoint** command to find it. For example, if you run this command:

```
aws iot describe-endpoint --endpoint-type iot:Jobs
```

you get a result similar to the following:

```
{
    "endpointAddress": "a1b2c3d4e5f6g7.jobs.iot.us-west-2.amazonaws.com"
}
```

With this method, your device uses X.509 certificate-based authentication (for example, its device-specific certificate and private key).

The following commands are available using this method:

- **DescribeJobExecution**
- **GetPendingJobExecutions**
- **StartNextPendingJobExecution**
- **UpdateJobExecution**

Programming devices to work with jobs

The examples in this section use MQTT to illustrate how a device works with the AWS IoT Jobs service. Alternatively, you could use the corresponding API or CLI commands. For these examples, we assume a device called `MyThing` subscribes to the following MQTT topics:

- `$aws/things/`*`MyThing`*`/jobs/notify` (or `$aws/things/`*`MyThing`*`/jobs/notify-next`)
- `$aws/things/`*`MyThing`*`/jobs/get/accepted`
- `$aws/things/`*`MyThing`*`/jobs/get/rejected`
- `$aws/things/`*`MyThing`*`/jobs/`*`jobId`*`/get/accepted`

- `$aws/things/`*`MyThing`*`/jobs/`*`jobId`*`/get/rejected`

If you are using Code-signing for AWS IoT your device code must verify the signature of your code file. The signature is in the job document in the `codesign` property. For more information about verifying a code file signature, see Device Agent Sample.

Device workflow

There are two ways a device can handle the jobs it is given to execute.

Option a: Get the next job

1. When a device first comes online, it should subscribe to the device's `notify-next` topic.
2. Call the DescribeJobExecution (p. 625) MQTT API with jobId `$next` to get the next job, its job document, and other details, including any state saved in `statusDetails`. If the job document has a code file signature, you must verify the signature before proceeding with processing the job request.
3. Call the UpdateJobExecution (p. 629) MQTT API to update the job status. Or, to combine this and the previous step in one call, the device can call StartNextPendingJobExecution (p. 620).
4. (Optional) You can add a step timer by setting a value for `stepTimeoutInMinutes` when you call either UpdateJobExecution (p. 629) or StartNextPendingJobExecution (p. 620).
5. Perform the actions specified by the job document using the UpdateJobExecution (p. 629) MQTT API to report on the progress of the job.
6. Continue to monitor the job execution by calling the DescribeJobExecution (p. 625) MQTT API with this jobId. If the job execution is deleted, DescribeJobExecution (p. 625) returns a `ResourceNotFoundException`.

   The device should be able to recover to a valid state if the job execution is canceled or deleted while the device is running the job.
7. Call the UpdateJobExecution (p. 629) MQTT API when finished with the job to update the job status and report success or failure.
8. Because this job's execution status has been changed to a terminal state, the next job available for execution (if any) changes. The device is notified that the next pending job execution has changed. At this point, the device should continue as described in step 2.

If the device remains online, it continues to receive a notifications of the next pending job execution, including its job execution data, when it completes a job or a new pending job execution is added. When this occurs, the device continues as described in step 2.

Option b: Pick from available jobs

1. When a device first comes online, it should subscribe to the thing's `notify` topic.
2. Call the GetPendingJobExecutions (p. 616) MQTT API to get a list of pending job executions.
3. If the list contains one or more job executions, pick one.
4. Call the DescribeJobExecution (p. 625) MQTT API to get the job document and other details, including any state saved in `statusDetails`.
5. Call the UpdateJobExecution (p. 629) MQTT API to update the job status. If the `includeJobDocument` field is set to `true` in this command, the device can skip the previous step and retrieve the job document at this point.
6. Optionally, you can add a step timer by setting a value for `stepTimeoutInMinutes` when you call UpdateJobExecution (p. 629).
7. Perform the actions specified by the job document using the UpdateJobExecution (p. 629) MQTT API to report on the progress of the job.

8. Continue to monitor the job execution by calling the DescribeJobExecution (p. 625) MQTT API with this jobId. If the job execution is canceled or deleted while the device is running the job, the device should be capable of recovering to a valid state.
9. Call the UpdateJobExecution (p. 629) MQTT API when finished with the job to update the job status and to report success or failure.

If the device remains online, it is notified of all pending job executions when a new pending job execution becomes available. When this occurs, the device can continue as described in step 2.

If the device is unable to execute the job, it should call the UpdateJobExecution (p. 629) MQTT API to update the job status to `REJECTED`.

Starting a new job

New job notification

When a new job is created, the AWS IoT Jobs service publishes a message on the `$aws/things/`*`thing-name`*`/jobs/notify` topic for each target device.

More Information(1)

The message contains the following information:

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

The device receives this message on the `'$aws/things/`*`thingName`*`/jobs/notify'` topic when the job execution is queued.

Get job information

To get more information about a job execution, the device calls the DescribeJobExecution (p. 625) MQTT API with the `includeJobDocument` field set to `true` (the default).

More Information(2)

If the request is successful, the AWS IoT Jobs service publishes a message on the `$aws/things/MyThing/jobs/0023/get/accepted` topic:

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

Note
If the request fails, the AWS IoT Jobs service publishes a message on the `$aws/things/MyThing/jobs/0023/get/rejected` topic.

The device now has the job document that it can use to perform the remote operations for the job. If the job document contains an Amazon S3 presigned URL, the device can use that URL to download any required files for the job.

Report job execution status

Update execution status

As the device is executing the job, it can call the UpdateJobExecution (p. 629) MQTT API to update the status of the job execution.

More information (3)

For example, a device can update the job execution status to `IN_PROGRESS` by publishing the following message on the `$aws/things/MyThing/jobs/0023/update` topic:

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

Jobs responds by publishing a message to the `$aws/things/MyThing/jobs/0023/update/accepted` or `$aws/things/MyThing/jobs/0023/update/rejected` topic:

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

The device can combine the two previous requests by calling StartNextPendingJobExecution (p. 620). That gets and starts the next pending job execution and allows the device to update the job execution status. This request also returns the job document when there is a job execution pending.

If the job contains a TimeoutConfig, the in-progress timer starts running. You can also set a step timer for a job execution by setting a value for `stepTimeoutInMinutes` when you call UpdateJobExecution. The step timer applies only to the job execution that you update. You can set a new value for this timer each time you update a job execution. You can also create a step timer when you call StartNextPendingJobExecution. If the job execution remains in the `IN_PROGRESS` status for longer than the step timer interval, it fails and switches to the terminal `TIMED_OUT` status. The step timer has no effect on the in-progress timer that you set when you create a job.

The `status` field can be set to `IN_PROGRESS`, `SUCCEEDED`, or `FAILED`. You cannot update the status of a job execution that is already in a terminal state.

Report execution completed

When the device is finished executing the job, it calls the UpdateJobExecution (p. 629) MQTT API. If the job was successful, set `status` to `SUCCEEDED` and, in the message payload, in `statusDetails`, add other information about the job as name-value pairs. The in-progress and step timers end when the job execution is complete.

More Information(4)

For example:

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

If the job was not successful, set `status` to `FAILED` and, in `statusDetails`, add information about the error that occurred:

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

Note
The `statusDetails` attribute can contain any number of name-value pairs.

When the AWS IoT Jobs service receives this update, it publishes a message on the `$aws/things/MyThing/jobs/notify` topic to indicate the job execution is complete:

```
{
    "timestamp":1476290692776,
    "jobs":{}
}
```

Additional jobs

Additional jobs

If there are other job executions pending for the device, they are included in the message published to `$aws/things/MyThing/jobs/notify`.

More Information(5)

For example:

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

Jobs notifications

The AWS IoT Jobs service publishes MQTT messages to reserved topics when jobs are pending or when the first job execution in the list changes. Devices can keep track of pending jobs by subscribing to these topics.

Job notifications are published to MQTT topics as JSON payloads. There are two kinds of notifications:

- A `ListNotification` contains a list of no more than 10 pending job executions. The job executions in this list have status values of either `IN_PROGRESS` or `QUEUED`. They are sorted by status (`IN_PROGRESS` job executions before `QUEUED` job executions) and then by the times when they were queued.

  A `ListNotification` is published whenever one of the following criteria is met.
  - A new job execution is queued or changes to a non-terminal status (`IN_PROGRESS` or `QUEUED`).
  - An old job execution changes to a terminal status (`FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED`, or `REMOVED`).
- A `NextNotification` contains summary information about the one job execution that is next in the queue.

  A `NextNotification` is published whenever the first job execution in the list changes.
  - A new job execution is added to the list as `QUEUED`, and it is the first one in the list.
  - The status of an existing job execution that was not the first one in the list changes from `QUEUED` to `IN_PROGRESS` and becomes the first one in the list. (This happens when there are no other `IN_PROGRESS` job executions in the list or when the job execution whose status changes from `QUEUED` to `IN_PROGRESS` was queued earlier than any other `IN_PROGRESS` job execution in the list.)
  - The status of the job execution that is first in the list changes to a terminal status and is removed from the list.

For more information about publishing and subscribing to MQTT topics, see the section called “Device communication protocols” (p. 76).

> **Note**
> Notifications are not available when you use HTTP Signature Version 4 or HTTP TLS to communicate with jobs.

Job pending

> The AWS IoT Jobs service publishes a message on an MQTT topic when a job is added to or removed from the list of pending job executions for a thing or the first job execution in the list changes:

- `$aws/things/thingName/jobs/notify`
- `$aws/things/thingName/jobs/notify-next`

More Information(6)

The messages contain the following example payloads:

`$aws/things/thingName/jobs/notify`:

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : [ {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0
    } ]
  }
}
```

`$aws/things/thingName/jobs/notify-next`:

```
{
  "timestamp" : 10011,
  "execution" : {
    "jobId" : "other-job",
    "status" : "IN_PROGRESS",
    "queuedAt" : 10009,
    "lastUpdatedAt" : 10009,
    "versionNumber" : 1,
    "executionNumber" : 1,
    "jobDocument" : {"c":"d"}
  }
}
```

Possible job execution status values are `QUEUED`, `IN_PROGRESS`, `FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED`, and `REMOVED`.

The following series of examples show the notifications that are published to each topic as job executions are created and change from one state to another.

First, one job, called `job1`, is created. This notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

This notification is published to the `jobs/notify-next` topic:

```
{
  "timestamp": 1517016948,
  "execution": {
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

When another job is created (`job2`), this notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

A notification is not published to the `jobs/notify-next` topic because the next job in the queue (`job1`) has not changed. When `job1` starts to execute, its status changes to `IN_PROGRESS`. No notifications are published because the list of jobs and the next job in the queue have not changed.

When a third job (`job3`) is added, this notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517017906,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517017472,
        "startedAt": 1517017472,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ],
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

A notification is not published to the `jobs/notify-next` topic because the next job in the queue is still `job1`.

When `job1` is complete, its status changes to `SUCCEEDED`, and this notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

At this point, `job1` has been removed from the queue, and the next job to be executed is `job2`. This notification is published to the `jobs/notify-next` topic:

```
{
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

If `job3` must begin executing before `job2` (which is not recommended), the status of `job3` can be changed to `IN_PROGRESS`. If this happens, `job2` is no longer next in the queue, and this notification is published to the `jobs/notify-next` topic:

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

No notification is published to the `jobs/notify` topic because no job has been added or removed.

If the device rejects `job2` and updates its status to `REJECTED`, this notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517189392,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
        "startedAt": 1517186779,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

If `job3` (which is still in progress) is force deleted, this notification is published to the `jobs/notify` topic:

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

At this point, the queue is empty. This notification is published to the `jobs/notify-next` topic:

```
{
  "timestamp": 1517189551
}
```

Using the AWS IoT jobs APIs

There are two categories of API used in the AWS IoT Jobs service:

- Those used for management and control of jobs.
- Those used by the devices executing those jobs.

In general, job management and control uses an HTTPS protocol API. Devices can use either an MQTT or an HTTPS protocol API. (The HTTPS API is designed for a low volume of calls typical when creating and tracking jobs. It usually opens a connection for a single request, and then closes the connection after the response is received. The MQTT API allows long polling. It is designed for large amounts of traffic that can scale to millions of devices.)

Note
Each AWS IoT Jobs HTTPS API has a corresponding command that allows you to call the API from the AWS CLI. The commands are lowercase, with hyphens between the words that make up the name of the API. For example, you can invoke the `CreateJob` API on the CLI by typing:

```
aws iot create-job ...
```

Job management and control API

The following commands are available for Job management and control in the CLI and over the HTTPS protocol.

- AssociateTargetsWithJob (p. 562)
- CancelJob (p. 564)
- CancelJobExecution (p. 566)
- CreateJob (p. 569)
- DeleteJob (p. 577)
- DeleteJobExecution (p. 580)
- DescribeJob (p. 583)
- DescribeJobExecution (p. 589)
- GetJobDocument (p. 593)
- ListJobExecutionsForJob (p. 594)
- ListJobExecutionsForThing (p. 596)
- ListJobs (p. 599)
- UpdateJob (p. 603)

Find the *`endpoint-url`* parameter for your CLI commands by using this command.

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

This command returns the following output.

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

Note
The Jobs endpoint doesn't support ALPN `z-amzn-http-ca`.

AssociateTargetsWithJob

AssociateTargetsWithJob command

Associates a group with a continuous job. For more information, see CreateJob (p. 569). The following criteria must be met:

- The job must have been created with the `targetSelection` field set to `CONTINUOUS`.
- The job status must currently be `IN_PROGRESS`.
- The total number of targets associated with a job must not exceed 100.

HTTPS (1)

Request:

```
POST /jobs/jobId/targets

{
"targets": [ "string" ],
"comment": "string"
}
```

`jobId`

The unique identifier you assigned to this job when it was created.

`targets`

A list of thing group ARNs that define the targets of the job.

`comment`

Optional. A comment string that describes why the job was associated with the targets.

Response:

```
{
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

`jobArn`

An ARN identifying the job.

`jobId`

The unique identifier you assigned to this job when it was created.

`description`

A short text description of the job.

CLI (1)

Synopsis:

```
aws iot  associate-targets-with-job \
--targets <value> \
--job-id <value> \
[--comment <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
targets	list member: TargetArn	A list of thing group ARNs that define the targets of the job.
TargetArn	string	
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
comment	string length max:2028 pattern: [^\\p{C}]+	An optional string that describes why the job was associated with the targets.

Output:

```
{
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

CLI output fields:

Name	Type	Description
jobArn	string	An ARN identifying the job.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
description	string length max:2028 pattern: [^\\p{C}]+	A short text description of the job.

CancelJob

CancelJob command

Cancels a job.

HTTPS (2)

Request:

```
PUT /jobs/jobId/cancel

{
"force": boolean,
"comment": "string",
"reasonCode": "string"
}
```

`jobId`

The unique identifier you assigned to this job when it was created.

`force`

[Optional] If `true`, job executions with status `IN_PROGRESS` and `QUEUED` are canceled. Otherwise, only job executions with status `QUEUED` are canceled. The default is `false`.

Warning
Canceling a job with a status of `IN_PROGRESS` causes a device that is executing the job to be unable to update the job execution status. Use caution and make sure that each device executing a job that is canceled is able to recover to a valid state.

`comment`

[Optional] A comment string that describes why the job was canceled.

`reasonCode`

[Optional] A reason code string that explains why the job was canceled. If a job is cancelled because it meets the conditions defined by an `abortConfig`, this field is auto-populated.

Response:

```
{
```

```
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

`jobArn`

The job ARN.

`jobId`

The unique identifier you assigned to this job when it was created.

`description`

A short text description of the job.

CLI (2)

Synopsis:

```
aws iot  cancel-job \
--job-id <value> \
[--force <value>]  \
[--comment <value>]  \
[--reasonCode <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"force": boolean,
"comment": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
force	boolean	If true, jobs with status QUEUED and IN_PROGRESS are canceled. Otherwise, only jobs with status QUEUED are canceled. **Warning** Canceling a job with a status of IN_PROGRESS causes a device that is executing the job to be unable to update

Name	Type	Description
		the job execution status. Use caution and make sure that each device executing a job that is canceled is able to recover to a valid state.
comment	string length max:2028 pattern: [^\\p{C}]+	An optional string that describes why the job was canceled.
reasonCode	string length max:128 pattern: [\p{Upper}\p{Digit}_]+	An optional string that explains why the job was canceled. If the job is canceled because it meets the criteria defined by the `abortConfig`, this field is auto-populated.

Output:

```
{
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

CLI output fields:

Name	Type	Description
jobArn	string	The job ARN.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
description	string length max:2028 pattern: [^\\p{C}]+	A short text description of the job.

CancelJobExecution

CancelJobExecution command

Cancels a job execution on a device.

HTTPS (3)

Request:

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
    ...
}
}
```

`thingName`

The name of the thing whose job execution will be canceled.

`jobId`

The unique identifier you assigned to the job when it was created.

`force`

Optional. If `true`, a job execution with a status of `IN_PROGRESS` or `QUEUED` can be canceled. Otherwise, only a job execution with status of `QUEUED` can be canceled. If you attempt to cancel a job execution with a status of `IN_PROGRESS` and you do not set `force` to `true`, an `InvalidStateTransitionException` is thrown. The default is `false`.

Warning
Canceling a job with a status of `IN_PROGRESS` causes a device that is executing the job to be unable to update the job execution status. Use caution and make sure that each device executing a job that is canceled is able to recover to a valid state.

`expectedVersion`

Optional. The expected current version of the job execution. Each time you update the job execution, its version is incremented. If the version of the job execution stored in the AWS IoT Jobs service does not match, the update is rejected with a `VersionConflictException` error, and an `ErrorResponse` that contains the current job execution status data is returned. (This makes it unnecessary to perform a separate `DescribeJobExecution` request to obtain the job execution status data.)

`statusDetails`

Optional. A collection of name-value pairs that describe the status of the job execution.

Response:

```
{
}
```

CLI (3)

Synopsis:

```
aws iot  cancel-job-execution \
--job-id <value> \
--thing-name <value> \
[--force | --no-force] \
[--expected-version <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The job to be canceled.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing whose execution of the job will be canceled.
force	boolean	Optional. If `true`, the job execution is canceled if it has status of IN_PROGRESS or QUEUED. Otherwise, the job execution is canceled only if it has status of QUEUED. However, if you attempt to cancel a job execution that has a status of IN_PROGRESS, and you do not set `--force` to `true`, an `InvalidStateTransitionException` is thrown. The default is `false`. **Warning** Canceling a job that has a status of IN_PROGRESS, causes a device that is executing the job to be unable to update the job execution status. Use caution and make sure that each device executing a job that is canceled is able to recover to a valid state.
expectedVersion	long	Optional. The expected current version of the job execution.

Name	Type	Description
	java class: java.lang.Long	Each time you update the job execution, its version is incremented. If the version of the job execution stored in the AWS IoT Jobs service does not match, the update is rejected with a `VersionMismatch` error, and an `ErrorResponse` that contains the current job execution status data is returned. (This makes it unnecessary to perform a separate `DescribeJobExecution` request to obtain the job execution status data.)
statusDetails	map key: DetailsKey value: DetailsValue	A collection of name-value pairs that describe the status of the job execution. If not specified, the `statusDetails` are unchanged.
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	

Output:

None

CreateJob

CreateJob command

Creates a job. You can provide the job document as a link to a file in an Amazon S3 bucket (`documentSource` parameter) or in the body of the request (`document` parameter).

A job can be made *continuous* by setting the optional `targetSelection` parameter to `CONTINUOUS`. (The default is `SNAPSHOT`.) A continuous job can be used to onboard or upgrade devices as they are added to a group because it continues to run and is executed on newly added things, even after the things in the group at the time the job was created have completed the job.

A job can have an optional TimeoutConfig, which sets the value of the in-progress timer. The in-progress timer can't be updated and applies to all executions of the job.

The following validations are performed on arguments to the `CreateJob` API:

- The `targets` argument must be a list of valid thing or thing group ARNs. All things and thing groups must be in your AWS account.
- The `documentSource` argument must be a valid Amazon S3 URL to a job document. Amazon S3 URLs are of the form: `https://s3.amazonaws.com/`*`bucketName`*`/`*`objectName`*.
- The document stored in the URL specified by the `documentSource` argument must be a UTF-8 encoded JSON document.
- The size of a job document is limited to 32 KB due to the limit on the size of an MQTT message(128 KB) and encryption.
- The `jobId` must be unique in your AWS account.

HTTPS (4)

Request:

```
PUT /jobs/jobId

{
"targets": [ "string" ],
"document": "string",
"documentSource": "string",
"description": "string",
"presignedUrlConfigData": {
    "roleArn": "string",
    "expiresInSec": "integer"
},
"targetSelection": "CONTINUOUS|SNAPSHOT",
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
         "thresholdPercentage": integer
      }
   ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
}
}
```

`jobId`

A job identifier, which must be unique for your AWS account. We recommend using a UUID. Alphanumeric characters, "-", and "_" can be used here.

`targets`

A list of thing or thing group ARNs that defines the targets of the job.

`document`

Optional. The job document.

`documentSource`

Optional. An Amazon S3 link to the job document.

`description`

Optional. A short text description of the job.

`presignedUrlConfigData`

Optional. Configuration information for presigned Amazon S3 URLs.

`roleArn`

The ARN of the IAM role that contains permissions to access the Amazon S3 bucket. This is the bucket that contains the data that devices download with the presigned Amazon S3 URLs. This role must also grant AWS IoT permission to assume the role. For more information, see Create jobs (p. 541).

`expiresInSec`

How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.

`targetSelection`

Optional. Specifies whether the job continues to run (CONTINUOUS) or is complete after all those things specified as targets have completed the job (SNAPSHOT). If CONTINUOUS, the job might also be scheduled to run on a thing when a change is detected in a target. For example, a job is scheduled to run on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.

`jobExecutionRolloutConfig`

Optional. Allows you to create a staged rollout of a job.

`maximumPerMinute`

The maximum number of things on which the job is sent for execution, per minute. Valid values are 1 to 1000. If not specified, the default is 1000. The actual number of things that receive the job might be less during any particular minute interval (due to system latency), but is not more than the specified value.

`exponentialRate`

Allows you to create an exponential rate of rollout for a job.

`baseRatePerMinute`

The minimum number of things that are notified of a pending job, per minute, at the start of job rollout. This parameter allows you to define the initial rate of rollout.

`incrementFactor`

The exponential factor to increase the rate of rollout for a job.

`rateIncreaseCriteria`

The criteria to initiate the increase in rate of rollout for a job. Set values for either the `numberOfNotifiedThings` or `numberOfSucceededThings`, but not both.

`numberOfNotifiedThings`

The threshold for number of notified things that initiate the increase in rate of rollout.

`numberOfSucceededThings`

The threshold for number of succeeded things that initiate the increase in rate of rollout.

`abortConfig`

Optional. Details of abort criteria to abort the job.

`criteriaList`

The list of abort criteria to define rules to abort the job.

`action`

The type of abort action to initiate a job abort.

`failureType`

The type of job execution failure to define a rule to initiate a job abort.

`minNumberOfExecutedThings`

Minimum number of executed things before evaluating an abort rule.

`thresholdPercentage`

The threshold as a percentage of the total number of executed things that initiate a job abort.

`timeoutConfig`

Optional. Specifies the amount of time each device has to finish its execution of the job. The timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.

`inProgressTimeoutInMinutes`

Specifies the amount of time, in minutes, this device has to finish execution of this job. A timer is started, or restarted, whenever this job's execution status is specified as `IN_PROGRESS` with this field populated. If the job execution status is not set to a terminal state before the timer expires, or before another job execution status update is sent with this field populated, the status is set to `TIMED_OUT`.

Response:

```
{
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

`jobArn`

The ARN of the job.

`jobId`

The unique identifier you assigned to this job.

`description`

An optional short text description of the job.

CLI (4)

Synopsis:

```
aws iot  create-job \
--job-id <value> \
--targets <value> \
[--document-source <value>] \
[--document <value>] \
[--description <value>] \
[--presigned-url-config <value>] \
[--target-selection <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
[--timeout-config <value>] \
[--document-parameters <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"targets": [
"string"
],
"documentSource": "string",
"document": "string",
"description": "string",
"presignedUrlConfig": {
"roleArn": "string",
"expiresInSec": long
},
"targetSelection": "string",
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": integer,
      "incrementFactor": integer,
      "rateIncreaseCriteria": {
         "numberOfNotifiedThings": integer, // Set one or the other
         "numberOfSucceededThings": integer // of these two values.
      },
  "maximumPerMinute": integer
  }
},
"abortConfig": {
 "criteriaList": [
    {
       "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": integer,
        "thresholdPercentage": integer
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
},
"documentParameters": {
```

```
"string": "string"
}
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	A job identifier which must be unique for your AWS account. We recommend using a UUID. Alphanumeric characters, "-" and "_" are valid for use here.
targets	list member: TargetArn	A list of things and thing groups to which the job should be sent.
TargetArn	string	
documentSource	string length max:1350 min:1	An S3 link to the job document.
document	string length max:32768	The job document.
description	string length max:2028 pattern: [^\\p{C}]+	A short text description of the job.
presignedUrlConfig	PresignedUrlConfig	Configuration information for presigned S3 URLs.
roleArn	string length max:2048 min:20	The ARN of an IAM role that grants permission to download files from the Amazon S3 bucket where the job data or updates are stored. The role must also grant permission for AWS IoT to download the files.
expiresInSec	long java class: java.lang.Long range- max:3600 min:60	How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.
targetSelection	string enum: CONTINUOUS | SNAPSHOT	Specifies whether the job continues to run (CONTINUOUS), or is complete after all those things specified as targets have completed the job (SNAPSHOT). If continuous,

Name	Type	Description
		the job can also be run on a thing when a change is detected in a target. For example, a job runs on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.
jobExecutionsRolloutConfig	JobExecutionsRolloutConfig	Allows you to create a staged rollout of the job.
maximumPerMinute	integer java class: java.lang.Integer range- max:1000 min:1	The maximum number of things that are notified of a pending job, per minute. This parameter allows you to create a staged rollout.
exponentialRate	ExponentialRolloutRate	The rate of increase for a job rollout. This parameter allows you to define an exponential rate for a job rollout.
baseRatePerMinute	java class: java.lang.Integer	The minimum number of things that will be notified of a pending job, per minute at the start of job rollout. This parameter allows you to define the initial rate of rollout.
incrementFactor	java class: java.lang.Double	The exponential factor to increase the rate of rollout for a job.
rateIncreaseCriteria	RateIncreaseCriteria	Allows you to define a criteria to initiate the increase in rate of rollout for a job. Set a value for either `numberOfNotifiedThings` or `numberOfSucceededThings`, but not both.
numberOfNotifiedThings	java class: java.lang.Double	The threshold for number of notified things that will initiate the increase in rate of rollout.
numberOfSucceededThings	java class: java.lang.Double	The threshold for number of succeeded things that will initiate the increase in rate of rollout.
abortConfig	AbortConfig	Allows you to create criteria to abort a job.
criteriaList	AbortCriteria	The list of abort criteria to define rules to abort the job.

Name	Type	Description
action	java class: java.lang.String (CANCEL)	The type of abort action to initiate a job abort.
failureType	java class: java.lang.String (FAILED | REJECTED | TIMED_OUT | ALL)	The type of job execution failure to define a rule to initiate a job abort.
minNumberOfExecutedThings	java class: java.lang.Integer)	Minimum number of executed things before evaluating an abort rule.
thresholdPercentage	java class: java.lang.Double)	The threshold as a percentage of the total number of executed things that will initiate a job abort. AWS IoT supports up to two digits after the decimal (for example, 10.9 and 10.99, but not 10.999).
timeoutConfig	TimeoutConfig	Specifies the amount of time each device has to finish its execution of the job. The timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.
inProgressTimeoutInMinutes	long	Specifies the amount of time, in minutes, this device has to finish execution of this job. A timer is started, or restarted, whenever this job's execution status is specified as `IN_PROGRESS` with this field populated. If the job execution status is not set to a terminal state before the timer expires, or before another job execution status update is sent with this field populated, the status is set to `TIMED_OUT`.
documentParameters	map key: ParameterKey value: ParameterValue	Parameters for the job document.

Name	Type	Description
ParameterKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
ParameterValue	string length max:1024 min:1 pattern: [^\\p{C}]+	

Output:

```
{
"jobArn": "string",
"jobId": "string",
"description": "string"
}
```

CLI output fields:

Name	Type	Description
jobArn	string	The job ARN.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job.
description	string length max:2028 pattern: [^\\p{C}]+	The job description.

DeleteJob

DeleteJob command

Deletes a job and its related job executions.

Deleting a job can take time, depending on the number of job executions created for the job and various other factors. While the job is being deleted, the status of the job is shown as "DELETION_IN_PROGRESS". Attempting to delete or cancel a job whose status is already "DELETION_IN_PROGRESS" results in an error.

HTTPS (5)

Request syntax:

```
DELETE /jobs/jobId?force=force
```

URI request parameters:

Name	Type	Req?	Description
jobId	JobId	yes	The ID of the job to be deleted.
force	ForceFlag	no	(Optional) When true, you can delete a job with a status of "IN_PROGRESS". Otherwise, you can only delete a job that is in a terminal state ("SUCCEEDED" or "CANCELED") or an exception occurs. The default is false. **Note** Deleting a job with a status of "IN_PROGRESS", causes a device that is executing the job to be unable to access job information or update the job execution status. Use caution and make sure that each device executing a job that is deleted is able to recover to a valid state.

Errors:

`InvalidRequestException`

The contents of the request were invalid. For example, this code is returned when an UpdateJobExecution request contains invalid status details. The message contains details about the error.

HTTP response code: 400

`InvalidStateTransitionException`

An update attempted to change the job or job execution to a state that is invalid because of its current state (for example, an attempt to change a request in state SUCCEEDED to state

IN_PROGRESS). In this case, the body of the error message also contains the executionState field.

HTTP response code: 409

`ResourceNotFoundException`

The specified resource does not exist.

HTTP response code: 404

`ThrottlingException`

The rate exceeds the limit.

HTTP response code: 429

`ServiceUnavailableException`

The service is temporarily unavailable.

HTTP response code: 503

CLI (5)

Synopsis:

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"force": boolean
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The ID of the job to be deleted.
force	boolean	(Optional) When true, you can delete a job with a status of IN_PROGRESS. Otherwise, you can only delete a job that is in a terminal state (SUCCEEDED or CANCELED) or an exception occurs. The default is false. **Note** Deleting a job with a status of IN_PROGRESS, causes a device that is

Name	Type	Description
		executing the job to be unable to access job information or update the job execution status. Use caution and make sure that each device executing a job that is deleted is able to recover to a valid state.

Output:

None

DeleteJobExecution

DeleteJobExecution command

Deletes a job execution.

HTTPS (6)

Request syntax:

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

URI request parameters:

Name	Type	Req?	Description
jobId	JobId	yes	The ID of the job whose execution will be deleted.
thingName	ThingName	yes	The name of the thing whose execution of the job will be deleted.
executionNumber	ExecutionNumber	yes	The ID of the job execution to be deleted.
force	ForceFlag	no	When true, you can delete a job execution with a status of IN_PROGRESS. Otherwise, you can only delete a job execution that is in a terminal state (SUCCEEDED, FAILED, TIMED_OUT, REJECTED, REMOVED,

Name	Type	Req?	Description
			or CANCELED) or an exception occurs. The default is false. **Note** Deleting a job execution with a status of IN_PROGRESS causes the device to be unable to access job information or update the job execution status. Use caution and make sure that the device is able to recover to a valid state.

Errors:

`InvalidRequestException`

The contents of the request were invalid. For example, this code is returned when an UpdateJobExecution request contains invalid status details. The message contains details about the error.

HTTP response code: 400

`InvalidStateTransitionException`

An update attempted to change the job execution to a state that is invalid because of the job execution's current state (for example, an attempt to change a request in state SUCCEEDED to state IN_PROGRESS). In this case, the body of the error message also contains the `executionState` field.

HTTP response code: 409

`ResourceNotFoundException`

The specified resource does not exist.

HTTP response code: 404

`ThrottlingException`

The rate exceeds the limit.

HTTP response code: 429

`ServiceUnavailableException`

The service is temporarily unavailable.

HTTP response code: 503

CLI (6)

Synopsis:

```
aws iot  delete-job-execution \
--job-id <value> \
--thing-name <value> \
--execution-number <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The ID of the job whose execution will be deleted.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing whose execution of the job will be deleted.
executionNumber	long java class: java.lang.Long	The ID of the job execution to be deleted.
force	boolean	When true, you can delete a job execution with a status of IN_PROGRESS. Otherwise, you can only delete a job execution that is in a terminal state (SUCCEEDED, FAILED, TIMED_OUT, REJECTED, REMOVED, or CANCELED) or an exception occurs. The default is false. **Note** Deleting a job execution with a status of IN_PROGRESS causes the device to be unable to access job information or update

Name	Type	Description
		the job execution status. Use caution and make sure that the device is able to recover to a valid state.

Output:

None

DescribeJob

DescribeJob command

Gets the details of the specified job.

HTTPS (7)

Request:

```
GET /jobs/jobId
```

`jobId`

The unique identifier you assigned to this job when it was created.

Response:

```
{
"documentSource": "string",
"job": Job
}
```

`documentSource`

An Amazon S3 link to the job document.

`job`

A Job (p. 609) object.

CLI (7)

Synopsis:

```
aws iot  describe-job \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
```

```
"jobId": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.

Output:

```
{
"documentSource": "string",
"job": {
"jobArn": "string",
"jobId": "string",
"targetSelection": "string",
"status": "string",
"forceCanceled": boolean,
"comment": "string",
"targets": [
  "string"
],
"description": "string",
"presignedUrlConfig": {
  "roleArn": "string",
  "expiresInSec": long
},
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
         "thresholdPercentage": integer
      }
   ]
},
"createdAt": "timestamp",
"lastUpdatedAt": "timestamp",
"completedAt": "timestamp",
"jobProcessDetails": {
  "processingTargets": [
    "string"
  ],
  "numberOfCanceledThings": "integer",
  "numberOfSucceededThings": "integer",
  "numberOfFailedThings": "integer",
  "numberOfRejectedThings": "integer",
  "numberOfQueuedThings": "integer",
  "numberOfInProgressThings": "integer",
  "numberOfRemovedThings": "integer",
  "numberOfTimedOutThings": "integer"
},
"documentParameters": {
  "string": "string"
},
"timeoutConfig": {
   "inProgressTimeoutInMinutes": number
}
}
}
```

CLI output fields:

Name	Type	Description
documentSource	string length max:1350 min:1	An Amazon S3 link to the job document.
job	Job	Information about the job.
jobArn	string	An ARN that identifies the job with format "arn:aws:iot:region:account:job/jobId".
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
targetSelection	string enum: CONTINUOUS | SNAPSHOT	Specifies whether the job continues to run (CONTINUOUS), or is complete after all those things specified as targets have completed the job (SNAPSHOT). If continuous, the job can also be run on a thing when a change is detected in a target. For example, a job runs on a device when the thing representing the device is added to a target group, even after the job was completed by all things originally in the group.
status	string enum: IN_PROGRESS | CANCELED | SUCCEEDED	The status of the job, one of `IN_PROGRESS`, `CANCELED`, or `SUCCEEDED`.
forceCanceled	boolean java class: java.lang.Boolean	Is `true` if the job was canceled with the optional `force` parameter set to `true`.

Name	Type	Description
comment	string length max:2028 pattern: [^\\p{C}]+	If the job was updated, describes the reason for the update.
targets	list member: TargetArn	A list of AWS IoT things and thing groups to which the job should be sent.
TargetArn	string	
description	string length max:2028 pattern: [^\\p{C}]+	A short text description of the job.
presignedUrlConfig	PresignedUrlConfig	Configuration for presigned Amazon S3 URLs.
roleArn	string length max:2048 min:20	The ARN of an IAM role that grants permission to download files from the Amazon S3 bucket where the job data or updates are stored. The role must also grant permission for tAWS IoT Jobs service to download the files.
expiresInSec	long java class: java.lang.Long range- max:3600 min:60	How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.
jobExecutionsRolloutConfig	JobExecutionsRolloutConfig	Allows you to create a staged rollout of the job.
maximumPerMinute	integer java class: java.lang.Integer range- max:1000 min:1	The maximum number of things that are notified of a pending job, per minute. This parameter allows you to create a staged rollout.
exponentialRate	ExponentialRolloutRate	The rate of increase for a job rollout. This parameter allows you to define an exponential rate for a job rollout.

Name	Type	Description
baseRatePerMinute	java class: java.lang.Integer	The minimum number of things that are notified of a pending job, per minute at the start of job rollout. This parameter allows you to define the initial rate of rollout.
incrementFactor	java class: java.lang.Double	The exponential factor to increase the rate of rollout for a job.
rateIncreaseCriteria	RateIncreaseCriteria	Allows you to define a criteria to initiate the increase in rate of rollout for a job. Set a value for either `numberOfNotifiedThings` or `numberOfSucceededThings`, but not both.
numberOfNotifiedThings	java class: java.lang.Double	The threshold for number of notified things that initiate the increase in rate of rollout.
numberOfSucceededThings	java class: java.lang.Double	The threshold for number of succeeded things that initiate the increase in rate of rollout.
abortConfig	AbortConfig	Allows you to create criteria to abort a job.
criteriaList	AbortCriteria	The list of abort criteria to define rules to abort the job.
action	java class: java.lang.String (CANCEL)	The type of abort action to initiate a job abort.
failureType	java class: java.lang.String (FAILED | REJECTED | TIMED_OUT | ALL)	The type of job execution failure to define a rule to initiate a job abort.
minNumberOfExecutedThings	java class: java.lang.Integer)	Minimum number of executed things before evaluating an abort rule.
thresholdPercentage	java class: java.lang.Double)	The threshold as a percentage of the total number of executed things that initiate a job abort. AWS IoT supports up to two digits after the decimal (for example, 10.9 and 10.99, but not 10.999).

Name	Type	Description
createdAt	timestamp	The time, in seconds since the epoch, when the job was created.
lastUpdatedAt	timestamp	The time, in seconds since the epoch, when the job was last updated.
completedAt	timestamp	The time, in seconds since the epoch, when the job was completed.
jobProcessDetails	JobProcessDetails	Details about the job process.
processingTargets	list member: ProcessingTargetName java class: java.util.List	The devices on which the job is executing.
ProcessingTargetName	string	
numberOfCanceledThings	integer java class: java.lang.Integer	The number of things that canceled the job.
numberOfSucceededThings	integer java class: java.lang.Integer	The number of things that successfully completed the job.
numberOfFailedThings	integer java class: java.lang.Integer	The number of things that failed executing the job.
numberOfRejectedThings	integer java class: java.lang.Integer	The number of things that rejected the job.
numberOfQueuedThings	integer java class: java.lang.Integer	The number of things that are awaiting execution of the job.
numberOfInProgressThings	integer java class: java.lang.Integer	The number of things currently executing the job.
numberOfRemovedThings	integer java class: java.lang.Integer	The number of things that are no longer scheduled to execute the job because they have been deleted or have been removed from the group that was a target of the job.
numberOfTimedOutThings	integer java class: java.lang.Integer	The number of things whose job execution status is `TIMED_OUT`.

Name	Type	Description
documentParameters	map key: ParameterKey value: ParameterValue	The parameters specified for the job document.
ParameterKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
ParameterValue	string length max:1024 min:1 pattern: [^\\p{C}]+	
timeoutConfig	TimeoutConfig	Specifies the amount of time each device has to finish its execution of the job. A timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the timer expires, it is set to `TIMED_OUT`.
inProgressTimeoutInMinutes	long	Specifies the amount of time, in minutes, this device has to finish execution of this job. The timeout interval can be anywhere between 1 minute and 7 days (1 to 10080 minutes). The in-progress timer can't be updated and applies to all job executions for the job. Whenever a job execution remains in the `IN_PROGRESS` status for longer than this interval, the job execution fails and switches to the terminal `TIMED_OUT` status.

DescribeJobExecution

DescribeJobExecution command

Gets details of a job execution. The job's execution status must be `SUCCEEDED` or `FAILED`.

HTTPS (8)

Request:

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

`jobId`

The unique identifier you assigned to this job when it was created.

`thingName`

The thing name associated with the device the job execution is running on.

`executionNumber`

Optional. A number that is used to specify a job execution on a device. (See JobExecution (p. 613).) If not specified, the latest job execution is returned.

Response:

```
{
"execution": { JobExecution }
}
```

`execution`

A JobExecution (p. 613) object.

CLI (8)

Synopsis:

```
aws iot  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing on which the job execution is running.
executionNumber	long java class: java.lang.Long	A string (consisting of the digits "0" through "9") that is used to specify a particular

Name	Type	Description
		job execution on a particular device.

Output:

```
{
"execution": {
"approximateSecondsBeforeTimedOut": "number"
"jobId": "string",
"status": "string",
"forceCanceled": boolean,
"statusDetails": {
  "detailsMap": {
    "string": "string"
  }
},
"thingArn": "string",
"queuedAt": "timestamp",
"startedAt": "timestamp",
"lastUpdatedAt": "timestamp",
"executionNumber": long,
"versionNumber": long
}
}
```

CLI output fields:

Name	Type	Description
execution	JobExecution	Information about the job execution.
approximateSecondsBeforeTimedOut	long	The estimated number of seconds that remain before the job execution status is changed to `TIMED_OUT`. The timeout interval can be anywhere between 1 minute and 7 days (1 to 10080 minutes). The actual job execution timeout can occur up to 60 seconds later than the estimated duration. This value is not included if the job execution has reached a terminal status.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to the job when it was created.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | | TIMED_OUT |	The status of the job execution (IN_PROGRESS, QUEUED, FAILED, SUCCEEDED, TIMED_OUT, CANCELED, or REJECTED).

Name	Type	Description
	REJECTED | REMOVED | CANCELED	
forceCanceled	boolean java class: java.lang.Boolean	Is `true` if the job execution was canceled with the optional `force` parameter set to `true`.
statusDetails	JobExecutionStatusDetails	A collection of name-value pairs that describe the status of the job execution.
detailsMap	map key: DetailsKey value: DetailsValue	The job execution status.
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	
thingArn	string	The ARN of the thing on which the job execution is running.
queuedAt	timestamp	The time, in seconds since the epoch, when the job execution was queued.
startedAt	timestamp	The time, in seconds since the epoch, when the job execution started.
lastUpdatedAt	timestamp	The time, in seconds since the epoch, when the job execution was last updated.
executionNumber	long java class: java.lang.Long	A string (consisting of the digits "0" through "9") that identifies this job execution on this device. It can be used in commands that return or update job execution information.
versionNumber	long	The version of the job execution. Job execution versions are incremented each time they are updated by a device.

GetJobDocument

GetJobDocument command

Gets the job document for a job.

Note
Placeholder URLs are not replaced with presigned Amazon S3 URLs in the document returned. Presigned URLs are generated only when the AWS IoT Jobs service receives a request over MQTT.

HTTPS (9)

Request:

```
GET /jobs/jobId/job-document
```

`jobId`

The unique identifier you assigned to this job when it was created.

Response:

```
{
"document": "string"
}
```

`document`

The job document content.

CLI (9)

Synopsis:

```
aws iot  get-job-document \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.

Output:

```
{
"document": "string"
}
```

CLI output fields:

Name	Type	Description
document	string length max:32768	The job document content.

ListJobExecutionsForJob

ListExecutionsForJob command

Gets a list of job executions for a job.

HTTPS (10)

Request:

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

`jobId`

The unique identifier you assigned to this job when it was created.

`status`

Optional. A filter that lets you search for jobs that have the specified status: QUEUED, IN_PROGRESS, SUCCEEDED, FAILED, TIMED_OUT, REJECTED, REMOVED, or CANCELED.

`maxResults`

Optional. The maximum number of results to be returned per request.

`nextToken`

Optional. The token to retrieve the next set of results.

Response:

```
{
"executionSummaries": [ JobExecutionSummary ... ]
}
```

`executionSummaries`

A list of JobExecutionSummary (p. 614) objects associated with the specified job ID.

CLI (10)

Synopsis:

```
aws iot  list-job-executions-for-job \
--job-id <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The status of the job.
maxResults	integer java class: java.lang.Integer range- max:250 min:1	The maximum number of results to be returned per request.
nextToken	string	The token to retrieve the next set of results.

Output:

```
{
"executionSummaries": [
{
  "thingArn": "string",
  "jobExecutionSummary": {
    "status": "string",
    "queuedAt": "timestamp",
    "startedAt": "timestamp",
    "lastUpdatedAt": "timestamp",
    "executionNumber": long
  }
}
],
"nextToken": "string"
}
```

CLI output fields:

Name	Type	Description
executionSummaries	list member: JobExecutionSummaryForJob java class: java.util.List	A list of job execution summaries.
JobExecutionSummaryForJob	JobExecutionSummaryForJob	
thingArn	string	The ARN of the thing on which the job execution is running.
jobExecutionSummary	JobExecutionSummary	Contains a subset of information about a job execution.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The status of the job execution.
queuedAt	timestamp	The time, in seconds since the epoch, when the job execution was queued.
startedAt	timestamp	The time, in seconds since the epoch, when the job execution started.
lastUpdatedAt	timestamp	The time, in seconds since the epoch, when the job execution was last updated.
executionNumber	long java class: java.lang.Long	A string (consisting of the digits "0" through "9") that identifies this job execution on this device. It can be used later in commands that return or update job execution information.
nextToken	string	The token for the next set of results, or **null** if there are no additional results.

ListJobExecutionsForThing

ListJobExecutionsForThing command

Gets a list of job executions for a thing.

HTTPS (11)

Request:

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

`thingName`

The name of the thing for which `JobExecutions` will be listed.

`status`

An optional filter that lets you search for jobs that have the specified status: QUEUED, IN_PROGRESS, SUCCEEDED, FAILED, TIMED_OUT, REJECTED, REMOVED, or CANCELED.

`maxResults`

The maximum number of results to be returned per request.

`nextToken`

The token for the next set of results, or `null` if there are no additional results.

Response:

```
{
"executionSummaries": [ JobExecutionSummary ... ]
}
```

`executionSummaries`

A list of the JobExecutionSummary (p. 614) objects for the job executions associated with the specified thing.

CLI (11)

Synopsis:

```
aws iot  list-job-executions-for-thing \
--thing-name <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
thingName	string	The thing name.

Name	Type	Description
	length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	An optional filter that lets you search for jobs that have the specified status.
maxResults	integer java class: java.lang.Integer range- max:250 min:1	The maximum number of results to be returned per request.
nextToken	string	The token to retrieve the next set of results.

Output:

```
{
"executionSummaries": [
{
  "jobId": "string",
  "jobExecutionSummary": {
    "status": "string",
    "queuedAt": "timestamp",
    "startedAt": "timestamp",
    "lastUpdatedAt": "timestamp",
    "executionNumber": long
  }
}
],
"nextToken": "string"
}
```

CLI output fields:

Name	Type	Description
executionSummaries	list member: JobExecutionSummaryForThing java class: java.util.List	A list of job execution summaries.
JobExecutionSummaryForThing	JobExecutionSummaryForThing	
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.

Name	Type	Description
jobExecutionSummary	JobExecutionSummary	Contains a subset of information about a job execution.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT| REJECTED | REMOVED | CANCELED	The status of the job execution.
queuedAt	timestamp	The time, in seconds since the epoch, when the job execution was queued.
startedAt	timestamp	The time, in seconds since the epoch, when the job execution started.
lastUpdatedAt	timestamp	The time, in seconds since the epoch, when the job execution was last updated.
executionNumber	long java class: java.lang.Long	A string (consisting of the digits "0" through "9") that identifies this job execution on this device. It can be used later in commands that return or update job execution information.
nextToken	string	The token for the next set of results, or **null** if there are no additional results.

ListJobs

ListJobs command

Gets a list of the jobs in your AWS account.

HTTPS (12)

Request:

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thingGroup
```

`status`

Optional. A filter that lets you search for jobs that have the specified status: IN_PROGRESS, CANCELED, or SUCCEEDED.

`targetSelection`

Optional. A filter that lets you search for jobs that have the specified `targetSelection` value: CONTINUOUS or SNAPSHOT.

`thingGroupName`

Optional. A filter that lets you search for jobs that have the specified thing group name as a target.

`thingGroupId`

Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.

`maxResults`

Optional. The maximum number of results to be returned per request.

`nextToken`

Optional. The token to retrieve the next set of results.

Response:

```
{
"jobs": [ JobSummary ... ],
}
```

`jobs`

A list of JobSummary (p. 612) objects, one for each job in your AWS account that matches the specified filtering criteria.

CLI (12)

Synopsis:

```
aws iot  list-jobs \
[--status <value>] \
[--target-selection <value>] \
[--max-results <value>] \
[--next-token <value>] \
[--thing-group-name <value>] \
[--thing-group-id <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"status": "string",
"targetSelection": "string",
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
status	string enum: IN_PROGRESS | CANCELED | SUCCEEDED	An optional filter that lets you search for jobs that have the specified status.
targetSelection	string enum: CONTINUOUS | SNAPSHOT	Specifies whether the job continues to run (CONTINUOUS), or is complete after all those things specified as targets have completed the job (SNAPSHOT). If continuous, the job can also be run on a thing when a change is detected in a target. For example, a job runs on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.
maxResults	integer java class: java.lang.Integer range- max:250 min:1	The maximum number of results to return per request.
nextToken	string	The token to retrieve the next set of results.
thingGroupName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	A filter that limits the returned jobs to those for the specified group.
thingGroupId	string length max:128 min:1 pattern: [a-zA-Z0-9-]+	A filter that limits the returned jobs to those for the specified group.

Output:

```
{
"jobs": [
{
  "jobArn": "string",
  "jobId": "string",
  "thingGroupId": "string",
  "targetSelection": "string",
  "status": "string",
  "createdAt": "timestamp",
  "lastUpdatedAt": "timestamp",
  "completedAt": "timestamp"
}
],
"nextToken": "string"
}
```

CLI output fields:

Name	Type	Description
jobs	list member: JobSummary java class: java.util.List	A list of jobs.
JobSummary	JobSummary	
jobArn	string	The job ARN.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
thingGroupId	string length max:128 min:1 pattern: [a-zA-Z0-9-]+	The ID of the thing group.
targetSelection	string enum: CONTINUOUS | SNAPSHOT	Specifies whether the job continues to run (CONTINUOUS), or is complete after all those things specified as targets have completed the job (SNAPSHOT). If continuous, the job can also be run on a thing when a change is detected in a target. For example, a job runs on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.
status	string enum: IN_PROGRESS | CANCELED | SUCCEEDED	The job summary status.
createdAt	timestamp	The time, in seconds since the epoch, when the job was created.
lastUpdatedAt	timestamp	The time, in seconds since the epoch, when the job was last updated.
completedAt	timestamp	The time, in seconds since the epoch, when the job completed.

Name	Type	Description
nextToken	string	The token for the next set of results, or **null** if there are no additional results.

UpdateJob

UpdateJob command

Updates supported fields of the specified job. Updated values for `timeoutConfig` take effect for only newly in-progress executions. Currently in-progress executions continue to execute with the old timeout configuration.

HTTPS (13)

Request:

```
PATCH /jobs/jobId
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     },
  "maximumPerMinute": number
  },
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

`jobId`

A job identifier that must be unique for your AWS account. We recommend using a UUID. Alphanumeric characters, "-", and "_" can be used here.

`description`

Optional. A short text description of the job.

`presignedUrlConfigData`

Optional. Configuration information for presigned Amazon S3 URLs.

`roleArn`

The ARN of the IAM role that contains permissions to access the Amazon S3 bucket. This is the bucket that contains the data that devices download with the presigned Amazon S3 URLs. This role must also grant AWS IoT permission to assume the role. For more information, see Create jobs (p. 541).

`expiresInSec`

How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.

`jobExecutionRolloutConfig`

Optional. Allows you to create a staged rollout of a job.

`maximumPerMinute`

The maximum number of things on which the job is sent for execution, per minute. Valid values are 1 to 1000. If not specified, the default is 1000. The actual number of things that receive the job might be less during any particular minute interval (due to system latency), but are not more than the specified value.

`exponentialRate`

Allows you to create an exponential rate of rollout for a job.

`baseRatePerMinute`

The minimum number of things that are notified of a pending job, per minute at the start of job rollout. This parameter allows you to define the initial rate of rollout.

`incrementFactor`

The exponential factor to increase the rate of rollout for a job.

`rateIncreaseCriteria`

The criteria to initiate the increase in rate of rollout for a job. Set values for either the `numberOfNotifiedThings` or `numberOfSucceededThings`, but not both.

`numberOfNotifiedThings`

The threshold for number of notified things that initiate the increase in rate of rollout.

`numberOfSucceededThings`

The threshold for number of succeeded things that initiate the increase in rate of rollout.

`abortConfig`

Optional. Details of abort criteria to abort the job.

`criteriaList`

The list of abort criteria to define rules to abort the job.

`action`

The type of abort action to initiate a job abort.

`failureType`

The type of job execution failure to define a rule to initiate a job abort.

`minNumberOfExecutedThings`

Minimum number of executed things before evaluating an abort rule.

`thresholdPercentage`

The threshold as a percentage of the total number of executed things that initiate a job abort.

`timeoutConfig`

Optional. Specifies the amount of time each device has to finish its execution of the job. The timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.

`inProgressTimeoutInMinutes`

Specifies the amount of time, in minutes, this device has to finish execution of this job. A timer is started, or restarted, whenever this job's execution status is specified as `IN_PROGRESS` with this field populated. If the job execution status is not set to a terminal state before the timer expires, or before another job execution status update is sent with this field populated, the status is set to `TIMED_OUT`.

Response:

```
HTTP/1.1 200
```

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

CLI (13)

Synopsis:

```
aws iot  update-job \
--job-id <value> \
[--description <value>] \
[--presigned-url-config <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
[--timeout-config <value>] \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     }
  },
  "maximumPerMinute": number
},
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
      }
   ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	A job identifier that must be unique for your AWS account. We recommend using a UUID. Alphanumeric characters, "-" and "_" are valid for use here.
description	string length max:2028 pattern: [^\\p{C}]+	A short text description of the job.
presignedUrlConfig	PresignedUrlConfig	Configuration information for presigned S3 URLs.
roleArn	string length max:2048 min:20	The ARN of an IAM role that grants permission to download files from the S3 bucket where the job data or updates are stored. The role must also grant permission for AWS IoT to download the files.
expiresInSec	long java class: java.lang.Long range- max:3600 min:60	How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.
jobExecutionsRolloutConfig	JobExecutionsRolloutConfig	Allows you to create a staged rollout of the job.
maximumPerMinute	integer java class: java.lang.Integer range- max:1000 min:1	The maximum number of things that are notified of a pending job, per minute. This parameter allows you to create a staged rollout.
exponentialRate	ExponentialRolloutRate	The rate of increase for a job rollout. This parameter allows you to define an exponential rate for a job rollout.

Name	Type	Description
baseRatePerMinute	java class: java.lang.Integer	The minimum number of things that are notified of a pending job, per minute at the start of job rollout. This parameter allows you to define the initial rate of rollout.
incrementFactor	java class: java.lang.Double	The exponential factor to increase the rate of rollout for a job.
rateIncreaseCriteria	RateIncreaseCriteria	Allows you to define a criteria to initiate the increase in rate of rollout for a job. Set a value for either `numberOfNotifiedThings` or `numberOfSucceededThings`, but not both.
numberOfNotifiedThings	java class: java.lang.Double	The threshold for number of notified things that initiate the increase in rate of rollout.
numberOfSucceededThings	java class: java.lang.Double	The threshold for number of succeeded things that initiate the increase in rate of rollout.
abortConfig	AbortConfig	Allows you to create criteria to abort a job.
criteriaList	AbortCriteria	The list of abort criteria to define rules to abort the job.
action	java class: java.lang.String (CANCEL)	The type of abort action to initiate a job abort.
failureType	java class: java.lang.String (FAILED | REJECTED | TIMED_OUT | ALL)	The type of job execution failure to define a rule to initiate a job abort.
minNumberOfExecutedThings	java class: java.lang.Integer)	Minimum number of executed things before evaluating an abort rule.
thresholdPercentage	java class: java.lang.Double)	The threshold as a percentage of the total number of executed things that initiate a job abort. AWS IoT supports up to two digits after the decimal (for example, 10.9 and 10.99, but not 10.999).

Name	Type	Description
timeoutConfig	TimeoutConfig	Specifies the amount of time each device has to finish its execution of the job. The timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.
inProgressTimeoutInMinutes	long	Specifies the amount of time, in minutes, this device has to finish execution of this job. A timer is started, or restarted, whenever this job's execution status is specified as `IN_PROGRESS` with this field populated. If the job execution status is not set to a terminal state before the timer expires, or before another job execution status update is sent with this field populated, the status is set to `TIMED_OUT`.
documentParameters	map key: ParameterKey value: ParameterValue	Parameters for the job document.
ParameterKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
ParameterValue	string length max:1024 min:1 pattern: [^\\p{C}]+	

Output:

```
HTTP/1.1 200
```

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Job management and control data types

The following data types are used by management and control applications to communicate with the AWS IoT Jobs service.

Job

Job data type

The `Job` object contains details about a job.

Syntax (1)

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "forceCanceled": boolean,
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "comment": "string",
    "targets": ["string"],
    "description": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp,
    "jobProcessDetails": {
        "processingTargets": ["string"],
        "numberOfCanceledThings": long,
        "numberOfSucceededThings": long,
        "numberOfFailedThings": long,
        "numberOfRejectedThings": long,
        "numberOfQueuedThings": long,
        "numberOfInProgressThings": long,
        "numberOfRemovedThings": long,
        "numberOfTimedOutThings": long
    },
    "presignedUrlConfig": {
        "expiresInSec": number,
        "roleArn": "string"
    },
    "jobExecutionsRolloutConfig": {
        "exponentialRate": {
           "baseRatePerMinute": integer,
           "incrementFactor": integer,
           "rateIncreaseCriteria": {
              "numberOfNotifiedThings": integer, // Set one or the other
              "numberOfSucceededThings": integer // of these two values.
           },
           "maximumPerMinute": integer
      }
    },
    "abortConfig": {
       "criteriaList": [
          {
             "action": "string",
             "failureType": "string",
             "minNumberOfExecutedThings": integer,
             "thresholdPercentage": integer
          }
       ]
    },
    "timeoutConfig": {
        "inProgressTimeoutInMinutes": long
    }
}
```

Description (1)

`jobArn`

An ARN identifying the job with the format "arn:aws:iot:*region*:*account*:job/*jobId*".

`jobId`

The unique identifier you assigned to this job when it was created.

`status`

The status of the job, one of `IN_PROGRESS`, `CANCELED`, or `SUCCEEDED`.

`targetSelection`

Specifies whether the job continues to run (CONTINUOUS) or is complete after those things specified as targets have completed the job (SNAPSHOT). If CONTINUOUS, the job might also be run on a thing when a change is detected in a target. For example, a job runs on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.

`comment`

If the job was updated, describes the reason for the update.

`targets`

A list of AWS IoT things and thing groups to which the job should be sent.

`description`

A short text description of the job.

`createdAt`

The time, in seconds since the epoch, when the job was created.

`lastUpdatedAt`

The time, in seconds since the epoch, when the job was last updated.

`completedAt`

The time, in seconds since the epoch, when the job was completed.

`jobProcessDetails`

Details about the job process:

`processingTargets`

A list of AWS IoT things and thing groups that are currently executing the job.

`numberOfCanceledThings`

The number of AWS IoT things that canceled the job.

`numberOfSucceededThings`

The number of AWS IoT things that successfully completed the job.

`numberOfFailedThings`

The number of AWS IoT things that failed to complete the job.

`numberOfRejectedThings`

The number of AWS IoT things that rejected the job.

`numberOfQueuedThings`

The number of AWS IoT things that are awaiting execution of the job.

`numberOfInProgressThings`

The number of AWS IoT things that are currently executing the job.

`numberOfRemovedThings`

The number of AWS IoT things that are no longer scheduled to execute the job because they have been deleted or removed from the group that was a target of the job.

`numberOfTimedOutThings`

The number of things whose job execution status is `TIMED_OUT`.

`presignedUrlConfig`

Configuration information for presigned Amazon S3 URLs.

`expiresInSec`

How long (in seconds) presigned URLs are valid. Valid values are 60 - 3600. The default value is 3600 seconds. Presigned URLs are generated when the AWS IoT Jobs service receives an MQTT request for the job document.

`roleArn`

The ARN of an IAM role that grants permission to download files from an Amazon S3 bucket. The role must also grant permission for AWS IoT to download the files. For more information about how to create and configure the role, see Create jobs (p. 541).

`jobExecutionRolloutConfig`

Optional. Allows you to create a staged rollout of a job.

`maximumPerMinute`

The maximum number of things (devices) to which the job is sent for execution, per minute.

`exponentialRate`

Allows you to create an exponential rate of rollout for a job.

`baseRatePerMinute`

The minimum number of things that are notified of a pending job, per minute, at the start of job rollout. This parameter allows you to define the initial rate of rollout.

`incrementFactor`

The exponential factor to increase the rate of rollout for a job.

`rateIncreaseCriteria`

The criteria to initiate the increase in rate of rollout for a job. You can specify either `numberOfNotifiedThings` or `numberOfSucceededThing`, but not both.

`numberOfNotifiedThings`

The threshold for number of notified things that initiate the increase in rate of rollout.

`numberOfSucceededThings`

The threshold for number of succeeded things that initiate the increase in rate of rollout.

`abortConfig`

Optional. Details of abort criteria to abort the job.

`criteriaList`

The list of abort criteria to define rules to abort the job.

`action`

The type of abort action to initiate a job abort.

`failureType`

The type of job execution failure to define a rule to initiate a job abort.

`minNumberOfExecutedThings`

Minimum number of executed things before evaluating an abort rule.

`thresholdPercentage`

The threshold as a percentage of the total number of executed things that initiate a job abort.

`timeoutConfig`

Optional. Specifies the amount of time each device has to finish its execution of the job. The timer is started when the job execution status is set to `IN_PROGRESS`. If the job execution status is not set to another terminal state before the time expires, it is set to `TIMED_OUT`.

`inProgressTimeoutInMinutes`

Specifies the amount of time, in minutes, this device has to finish execution of this job. A timer is started, or restarted, whenever this job's execution status is specified as `IN_PROGRESS` with this field populated. If the job execution status is not set to a terminal state before the timer expires, or before another job execution status update is sent with this field populated, the status is set to `TIMED_OUT`.

JobSummary

JobSummary data type

The `JobSummary` object contains a job summary.

Syntax (2)

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
}
```

Description (2)

`jobArn`

An ARN that identifies the job.

`jobId`

The unique identifier you assigned to this job when it was created.

`status`

The job status. Can be one of `IN_PROGRESS`, `CANCELED`, or `SUCCEEDED`.

`targetSelection`

Specifies whether the job continues to run (CONTINUOUS) or is complete after all those things specified as targets have completed the job (SNAPSHOT). If CONTINUOUS, the job might also be run on a thing when a change is detected in a target. For example, a job runs on a thing when the thing is added to a target group, even after the job was completed by all things originally in the group.

`thingGroupId`

The ID of the thing group.

`createdAt`

The UNIX timestamp for when the job was created.

`lastUpdatedAt`

The UNIX timestamp for when the job was last updated.

`completedAt`

The UNIX timestamp for when the job was completed.

JobExecution

JobExection data type

The `JobExecution` object represents the execution of a job on a device.

Syntax (3)

```
{
    "approximateSecondsBeforeTimedOut": 50,
    "executionNumber": 1234567890,
    "forceCanceled": true|false,
    "jobId": "string",
    "lastUpdatedAt": timestamp,
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "forceCanceled": boolean,
    "statusDetails": {
        "detailsMap": {
            "string": "string" ...
        },
        "status": "string"
    },
    "thingArn": "string",
    "versionNumber": 123
}
```

Description (3)

approximateSecondsBeforeTimedOut

The estimated number of seconds that remain before the job execution status is changed to `TIMED_OUT`. The timeout interval can be anywhere between 1 minute and 7 days (1 to 10080 minutes). The actual job execution timeout can occur up to 60 seconds later than the estimated duration.

`jobId`

The unique identifier you assigned to this job when it was created.

`executionNumber`

A number that identifies this job execution on this device. It can be used later in commands that return or update job execution information.

`thingArn`

The AWS IoT thing ARN.

`queuedAt`

The time, in seconds since the epoch, when the job execution was queued.

`lastUpdatedAt`

The time, in seconds since the epoch, when the job execution was last updated.

`startedAt`

The time, in seconds since the epoch, when the job execution was started.

`status`

The status of the job execution. Can be one of `QUEUED`, `IN_PROGRESS`, `FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED`, or `REMOVED`.

`statusDetails`

A collection of name-value pairs that describe the status of the job execution.

JobExecutionSummary

JobExecutionSummary data type

The `JobExecutionSummary` object contains job execution summary information:

Syntax (4)

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

Description (4)

`executionNumber`

A number that identifies a job execution on a device. It can be used later in commands that return or update job execution information.

`queuedAt`

The time, in seconds since the epoch, when the job execution was queued.

`lastUpdatedAt`

The time, in seconds since the epoch, when the job execution was last updated.

`startAt`

The time, in seconds since the epoch, when the job execution was started.

`status`

The status of the job execution: `QUEUED`, `IN_PROGRESS`, `FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED`, or `REMOVED`.

JobExecutionSummaryForJob

JobExecutionSummaryForJob data type

The `JobExecutionSummaryForJob` object contains a summary of information about job executions for a specific job.

Syntax (5)

```
{
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
        ...
    ]
}
```

Description (5)

`thingArn`

The AWS IoT thing ARN.

`jobExecutionSummary`

An JobExecutionSummary (p. 614) object.

JobExecutionSummaryForThing

JobExecutionSummaryForThing data type

The `JobExecutionSummaryForThing` object contains a summary of information about a job execution on a specific thing.

Syntax (6)

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

Description (6)

`jobId`

The unique identifier you assigned to this job when it was created.

`jobExecutionSummary`

A JobExecutionSummary (p. 614) object.

Jobs device MQTT and HTTPS APIs

The following commands are available over the MQTT and HTTPS protocols.

- GetPendingJobExecutions (p. 616)
- StartNextPendingJobExecution (p. 620)
- DescribeJobExecution (p. 625)
- UpdateJobExecution (p. 629)
- JobExecutionsChanged (p. 635)
- NextJobExecutionChanged (p. 636)

Note
Jobs device commands can be issued by publishing MQTT messages to the Reserved topics used for Jobs commands (p. 92). Your client is automatically subscribed to the response message topics of these commands, which means that the message broker will publish response message topics to the client that published the command message whether your client has subscribed to the response message topics or not.
Because the message broker publishes response messages, even without an explicit subscription to them, your client must be configured to receive and identify the messages it receives. Your client must also confirm that the *`thingName`* in the incoming message topic applies to the client's thing name before the client acts on the message.
Messages that AWS IoT sends in response to MQTT Jobs API command messages are charged to your account, whether you subscribed to them explicitly or not.

GetPendingJobExecutions

Gets the list of all jobs for a thing that are not in a terminal state.

Note
AWS IoT publishes the response messages directly to the client that made the request, whether the client has subscribed to the response topics or not. Clients should expect to receive the response messages even if they have not subscribed to them. These response messages do not pass through the message broker and they cannot be subscribed to by other clients or rules.
Job progress messages that are processed through the message broker and can be used by AWS IoT rules are published as Jobs events (p. 973).

MQTT (12)

To invoke this API, publish a message on `$aws/things/`*`thingName`*`/jobs/get`.

Request payload:

```
{ "clientToken": "string" }
```

`clientToken`

Optional. A client token used to correlate requests and responses. Enter an arbitrary value here and it is reflected in the response.

The message broker will publish `$aws/things/`*`thingName`*`/jobs/get/accepted` and `$aws/things/`*`thingName`*`/jobs/get/rejected` even without a specific subscription to them. However, for your client to receive the messages, it must be listening for them. For more information, see the note about Jobs API messages (p. 616).

Response payload:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
}
```

`inProgressJobs`

A list of JobExecutionSummary (p. 638) objects with status `IN_PROGRESS`.

`queuedJobs`

A list of JobExecutionSummary (p. 638) objects with status `QUEUED`.

`clientToken`

A client token used to correlate requests and responses.

`timestamp`

The time, in seconds since the epoch, when the message was sent.

HTTPS (12)

Request:

```
GET /things/thingName/jobs
```

`thingName`

The name of the thing associated with the device.

Response:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

`inProgressJobs`

A list of JobExecutionSummary (p. 638) objects.

`queuedJobs`

A list of JobExecutionSummary (p. 638) objects.

CLI (12)

Synopsis:

```
aws iot-jobs-data  get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"thingName": "string"
}
```

`cli-input-json` Fields:

Name	Type	Description
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing that is executing the job.

Output:

```
{
"inProgressJobs": [
{
  "jobId": "string",
  "queuedAt": long,
  "startedAt": long,
  "lastUpdatedAt": long,
  "versionNumber": long,
  "executionNumber": long
}
],
"queuedJobs": [
{
  "jobId": "string",
  "queuedAt": long,
  "startedAt": long,
  "lastUpdatedAt": long,
  "versionNumber": long,
  "executionNumber": long
}
]
}
```

CLI output fields:

Name	Type	Description
inProgressJobs	list member: JobExecutionSummary java class: java.util.List	A list of JobExecutionSummary objects with status IN_PROGRESS.
JobExecutionSummary	JobExecutionSummary	
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.

Name	Type	Description
queuedAt	long	The time, in seconds since the epoch, when the job execution was enqueued.
startedAt	long java class: java.lang.Long	The time, in seconds since the epoch, when the job execution started.
lastUpdatedAt	long	The time, in seconds since the epoch, when the job execution was last updated.
versionNumber	long	The version of the job execution. Job execution versions are incremented each time the AWS IoT Jobs service receives an update from a device.
executionNumber	long java class: java.lang.Long	A number that identifies a job execution on a device.
queuedJobs	list member: JobExecutionSummary java class: java.util.List	A list of JobExecutionSummary objects with status QUEUED.
JobExecutionSummary	JobExecutionSummary	
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
queuedAt	long	The time, in seconds since the epoch, when the job execution was enqueued.
startedAt	long java class: java.lang.Long	The time, in seconds since the epoch, when the job execution started.
lastUpdatedAt	long	The time, in seconds since the epoch, when the job execution was last updated.
versionNumber	long	The version of the job execution. Job execution versions are incremented each time the AWS IoT Jobs service receives an update from a device.

Name	Type	Description
executionNumber	long java class: java.lang.Long	A number that identifies a job execution on a device.

StartNextPendingJobExecution

Gets and starts the next pending job execution for a thing (status IN_PROGRESS or QUEUED).

- Any job executions with status IN_PROGRESS are returned first.
- Job executions are returned in the order in which they were created.
- If the next pending job execution is QUEUED, its state is changed to IN_PROGRESS and the job execution's status details are set as specified.
- If the next pending job execution is already IN_PROGRESS, its status details are not changed.
- If no job executions are pending, the response does not include the `execution` field.
- You can optionally create a step timer by setting a value for the `stepTimeoutInMinutes` property. If you don't update the value of this property by running `UpdateJobExecution`, the job execution times out when the step timer expires.

Note
AWS IoT publishes the response messages directly to the client that made the request, whether the client has subscribed to the response topics or not. Clients should expect to receive the response messages even if they have not subscribed to them. These response messages do not pass through the message broker and they cannot be subscribed to by other clients or rules. Job progress messages that are processed through the message broker and can be used by AWS IoT rules are published as Jobs events (p. 973).

MQTT (13)

To invoke this API, publish a message on `$aws/things/`*`thingName`*`/jobs/start-next`.

Request payload:

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`statusDetails`

A collection of name-value pairs that describe the status of the job execution. If not specified, the `statusDetails` are unchanged.

`stepTimeOutInMinutes`

Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in field `stepTimeoutInMinutes`) the job execution status is set to `TIMED_OUT`. Setting this timeout has no effect on that job execution timeout that might have been specified when the job was created (`CreateJob` using the `timeoutConfig` field).

`clientToken`

A client token used to correlate requests and responses. Enter an arbitrary value here and it is reflected in the response.

The message broker will publish `$aws/things/thingName/jobs/start-next/accepted` and `$aws/things/thingName/jobs/start-next/rejected` even without a specific subscription to them. However, for your client to receive the messages, it must be listening for them. For more information, see the note about Jobs API messages (p. 616).

Response payload:

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
"clientToken" : "string"
}
```

`execution`

A JobExecution (p. 636) object. For example:

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

`timestamp`

The time, in milliseconds since the epoch, when the message was sent to the device.

`clientToken`

A client token used to correlate requests and responses.

HTTPS (13)

Request:

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

`thingName`

The name of the thing associated with the device.

`statusDetails`

A collection of name-value pairs that describe the status of the job execution. If not specified, the `statusDetails` are unchanged.

`stepTimeOutInMinutes`

Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in field `stepTimeoutInMinutes`) the job execution status is set to `TIMED_OUT`. Setting this timeout has no effect on that job execution timeout that might have been specified when the job was created (`CreateJob` using the `timeoutConfig` field).

Response:

```
{
"execution" :  JobExecution
}
```

`execution`

A JobExecution (p. 636) object. For example:

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

CLI (13)

Synopsis:

```
aws iot-jobs-data  start-next-pending-job-execution \
--thing-name <value> \
{--step-timeout-in-minutes <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

`cli-input-json` Fields:

Name	Type	Description
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing associated with the device.
statusDetails	map key: DetailsKey value: DetailsValue	A collection of name-value pairs that describe the status of the job execution. If not specified, the statusDetails are unchanged.
stepTimeOutInMinutes	long	Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in field `stepTimeoutInMinutes`) the job execution status is set to `TIMED_OUT`. Setting this timeout has no effect on that job execution timeout that might have been specified when the job was created (`CreateJob` using the `timeoutConfig` field).
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	

Output:

```
{
"execution": {
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
  "string": "string"
},
"queuedAt": long,
"startedAt": long,
"lastUpdatedAt": long,
"versionNumber": long,
"executionNumber": long,
"jobDocument": "string"
}
}
```

CLI output fields:

Name	Type	Description
execution	JobExecution	A JobExecution object.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing that is executing the job.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The status of the job execution. Can be one of: QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, or REMOVED.
statusDetails	map key: DetailsKey value: DetailsValue	A collection of name-value pairs that describe the status of the job execution.
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	
queuedAt	long	The time, in seconds since the epoch, when the job execution was enqueued.
startedAt	long java class: java.lang.Long	The time, in seconds since the epoch, when the job execution was started.

Name	Type	Description
lastUpdatedAt	long	The time, in seconds since the epoch, when the job execution was last updated.
versionNumber	long	The version of the job execution. Job execution versions are incremented each time they are updated by a device.
executionNumber	long java class: java.lang.Long	A number that identifies a job execution on a device. It can be used later in commands that return or update job execution information.
jobDocument	string length max:32768	The content of the job document.

DescribeJobExecution

Gets detailed information about a job execution.

You can set the `jobId` to `$next` to return the next pending job execution for a thing (status IN_PROGRESS or QUEUED).

MQTT (14)

To invoke this API, publish a message on `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get`.

Request payload:

```
{
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

`thingName`

The name of the thing associated with the device.

`jobId`

The unique identifier assigned to this job when it was created.

Or use `$next` to return the next pending job execution for a thing (status IN_PROGRESS or QUEUED). In this case, any job executions with status IN_PROGRESS are returned first. Job executions are returned in the order in which they were created.

`executionNumber`

Optional. A number that identifies a job execution on a device. If not specified, the latest job execution is returned.

`includeJobDocument`

Optional. Unless set to `false`, the response contains the job document. The default is `true`.

`clientToken`

A client token used to correlate requests and responses. Enter an arbitrary value here and it is reflected in the response.

The message broker will publish `$aws/things/thingName/jobs/jobId/get/accepted` and `$aws/things/thingName/jobs/jobId/get/rejected` even without a specific subscription to them. However, for your client to receive the messages, it must be listening for them. For more information, see the note about Jobs API messages (p. 616).

Response payload:

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

`execution`

A JobExecution (p. 636) object.

`timestamp`

The time, in seconds since the epoch, when the message was sent.

`clientToken`

A client token used to correlate requests and responses.

HTTPS (14)

The job's execution status must be `QUEUED` or `IN_PROGRESS`.

Request:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

`thingName`

The name of the thing associated with the device.

`jobId`

The unique identifier assigned to this job when it was created.

Or use `$next` to return the next pending job execution for a thing (status IN_PROGRESS or QUEUED). In this case, any job executions with status IN_PROGRESS are returned first. Job executions are returned in the order in which they were created.

`includeJobDocument`

Optional. Unless set to `false`, the response contains the job document. The default is `true`.

`executionNumber`

Optional. A number that identifies a job execution on a device. If not specified, the latest job execution is returned.

Response:

```
{
"execution" : JobExecution,
}
```

`execution`

A JobExecution (p. 636) object.

CLI (14)

The job's execution status must be `QUEUED` or `IN_PROGRESS`.

Synopsis:

```
aws iot-jobs-data  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string pattern: [a-zA-Z0-9_-]+|^$next	The unique identifier assigned to this job when it was created, or `$next` to return the next pending job execution for a thing (status IN_PROGRESS or QUEUED). In this case, any job executions with status IN_PROGRESS are returned first. Job executions are returned in the order in which they were created.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The thing name associated with the device the job execution is running on.
includeJobDocument	boolean java class: java.lang.Boolean	Optional. Unless set to false, the response contains the job document. The default is true.

Name	Type	Description
executionNumber	long java class: java.lang.Long	Optional. A number that identifies a job execution on a device. If not specified, the latest job execution is returned.

Output:

```
{
"execution": {
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
  "string": "string"
},
"queuedAt": long,
"startedAt": long,
"lastUpdatedAt": long,
"versionNumber": long,
"executionNumber": long,
"jobDocument": "string"
}
}
```

CLI output fields:

Name	Type	Description
execution	JobExecution	Contains data about a job execution.
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier you assigned to this job when it was created.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing that is executing the job.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The status of the job execution. Can be one of: QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, or REMOVED.
statusDetails	map key: DetailsKey value: DetailsValue	A collection of name-value pairs that describe the status of the job execution.
DetailsKey	string	

Name	Type	Description
	length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	
queuedAt	long	The time, in seconds since the epoch, when the job execution was enqueued.
startedAt	long java class: java.lang.Long	The time, in seconds since the epoch, when the job execution was started.
lastUpdatedAt	long	The time, in seconds since the epoch, when the job execution was last updated.
versionNumber	long	The version of the job execution. Job execution versions are incremented each time they are updated by a device.
executionNumber	long java class: java.lang.Long	A number that identifies a job execution on a device. It can be used later in commands that return or update job execution information.
jobDocument	string length max:32768	The content of the job document.

UpdateJobExecution

Updates the status of a job execution. You can optionally create a step timer by setting a value for the `stepTimeoutInMinutes` property. If you don't update the value of this property by running `UpdateJobExecution` again, the job execution times out when the step timer expires.

Note
AWS IoT publishes the response messages directly to the client that made the request, whether the client has subscribed to the response topics or not. Clients should expect to receive the response messages even if they have not subscribed to them. These response messages do not pass through the message broker and they cannot be subscribed to by other clients or rules. Job progress messages that are processed through the message broker and can be used by AWS IoT rules are published as Jobs events (p. 973).

MQTT (15)

To invoke this API, publish a message on `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/update`.

Request payload:

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`status`

The new status for the job execution (IN_PROGRESS, FAILED, SUCCEEDED, or REJECTED). This must be specified on every update.

`statusDetails`

A collection of name-value pairs that describe the status of the job execution. If not specified, the `statusDetails` are unchanged.

`expectedVersion`

The expected current version of the job execution. Each time you update the job execution, its version is incremented. If the version of the job execution stored in the AWS IoT Jobs service does not match, the update is rejected with a `VersionMismatch` error, and an ErrorResponse (p. 639) that contains the current job execution status data is returned. (This makes it unnecessary to perform a separate `DescribeJobExecution` request to obtain the job execution status data.)

`executionNumber`

Optional. A number that identifies a job execution on a device. If not specified, the latest job execution is used.

`includeJobExecutionState`

Optional. When included and set to `true`, the response contains the `JobExecutionState` field. The default is `false`.

`includeJobDocument`

Optional. When included and set to `true`, the response contains the `JobDocument`. The default is `false`.

`stepTimeoutInMinutes`

Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by again calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in this field) the job execution status is set to `TIMED_OUT`. Setting or resetting this timeout has no effect on the job execution timeout that might have been specified when the job was created (by using `CreateJob` with the `timeoutConfig`).

`clientToken`

A client token used to correlate requests and responses. Enter an arbitrary value here and it is reflected in the response.

The message broker will publish `$aws/things/thingName/jobs/jobId/update/accepted` and `$aws/things/thingName/jobs/jobId/update/rejected` even without a specific subscription to them. However, for your client to receive the messages, it must be listening for them. For more information, see the note about Jobs API messages (p. 616).

Response payload:

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

`executionState`

A JobExecutionState (p. 637) object.

`jobDocument`

A job document (p. 537) object.

`timestamp`

The time, in seconds since the epoch, when the message was sent.

`clientToken`

A client token used to correlate requests and responses.

HTTPS (15)

Request:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

`thingName`

The name of the thing associated with the device.

`jobId`

The unique identifier assigned to this job when it was created.

`status`

The new status for the job execution (IN_PROGRESS, FAILED, SUCCEEDED, or REJECTED). This must be specified on every update.

`statusDetails`

Optional. A collection of name-value pairs that describe the status of the job execution. If not specified, the `statusDetails` are unchanged.

`expectedVersion`

Optional. The expected current version of the job execution. Each time you update the job execution, its version is incremented. If the version of the job execution stored in the AWS IoT Jobs service does not match, the update is rejected with a `VersionMismatch` error, and an ErrorResponse (p. 639) that contains the current job execution status data is returned. (This makes it unnecessary to perform a separate `DescribeJobExecution` request to obtain the job execution status data.)

`includeJobExecutionState`

Optional. When included and set to `true`, the response contains the JobExecutionState data. The default is `false`.

`includeJobDocument`

Optional. When set to `true`, the response contains the job document. The default is `false`.

`stepTimeoutInMinutes`

Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by again calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in this field) the job execution status is set to `TIMED_OUT`. Setting or resetting this timeout has no effect on the job execution timeout that might have been specified when the job was created (by using `CreateJob` with the `timeoutConfig`).

`executionNumber`

Optional. A number that identifies a job execution on a device.

Response:

```
{
"executionState": JobExecutionState,
"jobDocument": "string"
}
```

`executionState`

A JobExecutionState (p. 637) object.

`jobDocument`

The contents of the job document (p. 537).

CLI (15)

Synopsis:

```
aws iot-jobs-data  update-job-execution \
--job-id <value> \
--thing-name <value> \
--status <value> \
[--status-details <value>] \
[--expected-version <value>] \
[--include-job-execution-state | --no-include-job-execution-state] \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--step-timeout-in-minutes <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

`cli-input-json` Fields:

Name	Type	Description
jobId	string length max:64 min:1 pattern: [a-zA-Z0-9_-]+	The unique identifier assigned to this job when it was created.
thingName	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	The name of the thing associated with the device.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The new status for the job execution (IN_PROGRESS, FAILED, SUCCEEDED, or REJECTED). This must be specified on every update.
statusDetails	map key: DetailsKey value: DetailsValue	Optional. A collection of name-value pairs that describe the status of the job execution. If not specified, the statusDetails are unchanged.
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	
stepTimeoutInMinutes		

Name	Type	Description
long Specifies the amount of time this device has to finish execution of this job. If the job execution status is not set to a terminal state before this timer expires, or before the timer is reset (by again calling `UpdateJobExecution`, setting the status to `IN_PROGRESS` and specifying a new timeout value in this field) the job execution status is set to `TIMED_OUT`. Setting or resetting this timeout has no effect on the job execution timeout that might have been specified when the job was created (by using `CreateJob` with the `timeoutConfig`).		
expectedVersion	long java class: java.lang.Long	Optional. The expected current version of the job execution. Each time you update the job execution, its version is incremented. If the version of the job execution stored in the AWS IoT Jobs service does not match, the update is rejected with a VersionMismatch error, and an ErrorResponse that contains the current job execution status data is returned. (This makes it unnecessary to perform a separate DescribeJobExecution request to obtain the job execution status data.)
includeJobExecutionState	boolean java class: java.lang.Boolean	Optional. When included and set to true, the response contains the JobExecutionState data. The default is false.
includeJobDocument	boolean java class: java.lang.Boolean	Optional. When set to true, the response contains the job document. The default is false.
executionNumber	long java class: java.lang.Long	Optional. A number that identifies a job execution on a device.

Output:

```
{
"executionState": {
"status": "string",
"statusDetails": {
  "string": "string"
},
"versionNumber": long
},
"jobDocument": "string"
}
```

CLI output fields:

Name	Type	Description
executionState	JobExecutionState	A JobExecutionState object.
status	string enum: QUEUED | IN_PROGRESS | SUCCEEDED | FAILED | TIMED_OUT | REJECTED | REMOVED | CANCELED	The status of the job execution. Can be one of: QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, or REMOVED.
statusDetails	map key: DetailsKey value: DetailsValue	A collection of name-value pairs that describe the status of the job execution.
DetailsKey	string length max:128 min:1 pattern: [a-zA-Z0-9:_-]+	
DetailsValue	string length max:1024 min:1 pattern: [^\\p{C}]*+	
versionNumber	long	The version of the job execution. Job execution versions are incremented each time they are updated by a device.
jobDocument	string length max:32768	The contents of the job documents.

JobExecutionsChanged

Sent whenever a job execution is added to or removed from the list of pending job executions for a thing.

MQTT (16)

Topic: `$aws/things/thingName/jobs/notify`

Message payload:

```
{
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary (p. 614) ... ]
},
"timestamp": timestamp,
}
```

HTTPS (16)

Not available.

CLI (16)

Not available.

NextJobExecutionChanged

Sent whenever there is a change to which job execution is next on the list of pending job executions for a thing, as defined for DescribeJobExecution (p. 625) with jobId `$next`. This message is not sent when the next job's execution details change, only when the next job that would be returned by `DescribeJobExecution` with jobId `$next` has changed. Consider job executions J1 and J2 with state QUEUED. J1 is next on the list of pending job executions. If the state of J2 is changed to IN_PROGRESS while the state of J1 remains unchanged, then this notification is sent and contains details of J2.

MQTT (17)

Topic: `$aws/things/thingName/jobs/notify-next`

Message payload:

```
{
"execution" : JobExecution (p. 613),
"timestamp": timestamp,
}
```

HTTPS (17)

Not available.

CLI (17)

Not available.

Jobs device MQTT and HTTPS data types

The following data types are used to communicate with the AWS IoT Jobs service over the MQTT and HTTPS protocols.

JobExecution

JobExecution data type

Contains data about a job execution.

Syntax (7)

```
{
    "jobId" : "string",
    "thingName" : "string",
    "jobDocument" : "string",
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
    },
    "queuedAt" : "timestamp",
    "startedAt" : "timestamp",
    "lastUpdatedAt" : "timestamp",
    "versionNumber" : "number",
    "executionNumber": long
}
```

Description (7)

`jobId`

The unique identifier you assigned to this job when it was created.

`thingName`

The name of the thing that is executing the job.

`jobDocument`

The content of the job document.

`status`

The status of the job execution. Can be one of: QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, or REMOVED.

`statusDetails`

A collection of name-value pairs that describe the status of the job execution.

`queuedAt`

The time, in seconds since the epoch, when the job execution was enqueued.

`startedAt`

The time, in seconds since the epoch, when the job execution was started.

`lastUpdatedAt`

The time, in seconds since the epoch, when the job execution was last updated.

`versionNumber`

The version of the job execution. Job execution versions are incremented each time they are updated by a device.

`executionNumber`

A number that identifies a job execution on a device. It can be used later in commands that return or update job execution information.

JobExecutionState

JobExecutionState data type

Contains data about the state of a job execution.

Syntax (8)

```
{
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

Description (8)

`status`

The status of the job execution. Can be one of: QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, or REMOVED.

`statusDetails`

A collection of name-value pairs that describe the status of the job execution.

`versionNumber`

The version of the job execution. Job execution versions are incremented each time they are updated by a device.

JobExecutionSummary

JobExecutionSummary data type

Contains a subset of information about a job execution.

Syntax (9)

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

Description (9)

`jobId`

The unique identifier you assigned to this job when it was created.

`queuedAt`

The time, in seconds since the epoch, when the job execution was enqueued.

`startedAt`

The time, in seconds since the epoch, when the job execution started.

`lastUpdatedAt`

The time, in seconds since the epoch, when the job execution was last updated.

`versionNumber`

The version of the job execution. Job execution versions are incremented each time the AWS IoT Jobs service receives an update from a device.

`executionNumber`

A number that identifies a job execution on a device.

ErrorResponse

ErrorResponse data type

Contains information about an error that occurred during an AWS IoT Jobs service operation.

Syntax (10)

```
{
    "code": "ErrorCode",
    "message": "string",
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
}
```

Description (10)

`code`

ErrorCode can be set to:

InvalidTopic

The request was sent to a topic in the AWS IoT Jobs namespace that does not map to any API.

InvalidJson

The contents of the request could not be interpreted as valid UTF-8-encoded JSON.

InvalidRequest

The contents of the request were invalid. For example, this code is returned when an `UpdateJobExecution` request contains invalid status details. The message contains details about the error.

InvalidStateTransition

An update attempted to change the job execution to a state that is invalid because of the job execution's current state (for example, an attempt to change a request in state SUCCEEDED to state IN_PROGRESS). In this case, the body of the error message also contains the `executionState` field.

ResourceNotFound

The `JobExecution` specified by the request topic does not exist.

VersionMismatch

The expected version specified in the request does not match the version of the job execution in the AWS IoT Jobs service. In this case, the body of the error message also contains the `executionState` field.

InternalError

There was an internal error during the processing of the request.

RequestThrottled

The request was throttled.

TerminalStateReached

Occurs when a command to describe a job is performed on a job that is in a terminal state.

`message`

An error message string.

`clientToken`

An arbitrary string used to correlate a request with its reply.

`timestamp`

The time, in seconds since the epoch.

`executionState`

A JobExecutionState (p. 637) object. This field is included only when the `code` field has the value `InvalidStateTransition` or `VersionMismatch`. This makes it unnecessary in these cases to perform a separate `DescribeJobExecution` request to obtain the current job execution status data.

Job rollout and abort configuration

AWS IoT jobs can be deployed using variable rollout rates as various criteria and thresholds are met. Job rollouts can also be aborted if the number of failed jobs matches a set of criteria. These rollout configurations give you more granular control over a job's blast radius. Job rollout rate criteria are set at job creation through the `JobExecutionsRolloutConfig` object. Job abort criteria are set at job creation through the `AbortConfig` object.

Using job rollout rates

You set a job rollout rate by configuring the `ExponentialRolloutRate` property of the `JobExecutionsRolloutConfig` object when you run the `CreateJob` API. The following example sets a variable rollout rate by using the `exponentialRate` parameter.

```
{
...
  "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": 50,
         "incrementFactor": 2,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": 1000, // Set one or the other
            "numberOfSucceededThings": 1000 // of these two values.
         },
         "maximumPerMinute": 1000
...
}
```

The `baseRatePerMinute` parameter specifies the rate at which the jobs are executed until the `numberOfNotifiedThings` or `numberOfSucceededThings` threshold has been met.

The `incrementFactor` parameter specifies the exponential factor by which the rollout rate increases after the `numberOfNotifiedThings` or `numberOfSucceededThings` threshold has been met.

The `rateIncreaseCriteria` parameter is an object that specifies either the `numberOfNotifiedThings` or `numberOfSucceededThings` threshold.

The `maximumPerMinute` parameter specifies the upper limit of the rate at which job executions can occur. Valid values range from 1 to 1000. This parameter is required when you pass an `ExponentialRate` object. In a variable rate rollout, this value establishes the upper limit of a job rollout rate.

A job rollout with the configuration above would start at a rate of 50 job executions per minute. It would continue at that rate until either 1000 things have received job execution notifications (if a value for `numberOfNotifiedThings` has been specified) or 1000 successful job executions have occurred (if a value for `numberOfSucceededThings` has been specified).

The following table illustrates how the rollout would proceed over the first four increments.

Rollout rate per minute	50	100	200	400
Number of notified devices or successful executions	1000	2000	3000	4000

The following configuration sets a static rollout rate.

```
{
...
  "jobExecutionsRolloutConfig": {
         "maximumPerMinute": 1000
...
}
```

The `maximumPerMinute` parameter specifies the upper limit of the rate at which job executions can occur. Valid values range from 1 to 1000. This parameter is optional. If you don't specify a value, the default value of 1000 is used.

Using job rollout abort configurations

You set up a job abort condition by configuring the optional `AbortConfig` object when you run the `CreateJob` API. This section describes the effect that the following sample configuration would have on a job rollout that was experiencing multiple failed executions.

```
   "abortConfig": {
      "criteriaList": [
         {
            "action": "CANCEL",
            "failureType": "FAILED",
            "minNumberOfExecutedThings": 100,
            "thresholdPercentage": 20
         },
         {
                "action": "CANCEL",
                "failureType": "TIMED_OUT",
                "minNumberOfExecutedThings": 200,
                "thresholdPercentage": 50
            }
        ]
    }
```

The `action` parameter specifies the action to take when the abort criteria have been met. This parameter is required, and `CANCEL` is the only valid value.

The `failureType` parameter specifies which failure types should trigger a job abort. Valid values are `FAILED`, `REJECTED`, `TIMED_OUT`, and `ALL`.

The `minNumberOfExecutedThings` parameter specifies the number of completed job executions that must occur before the service checks to see if the job abort criteria have been met. In this example, AWS IoT doesn't check to see if a job abort should occur until at least 100 devices have completed job executions.

The `thresholdPercentage` parameter specifies the total number of executed things that initiate a job abort. In this example, AWS IoT initiates a job abort and cancels the job rollout if at least 20% of all completed executions have failed in any way after 100 executions have completed.

Note
Deletion of job executions affects the computation value of the total completed execution. When a job aborts, the service creates an automated `comment` and `reasonCode` to differentiate a user-driven cancellation from a job abort cancellation.

Job limits

For job limit information, see AWS AWS IoT Device Management endpoints and quotas in the AWS General Reference.

AWS IoT secure tunneling

When devices are deployed behind restricted firewalls at remote sites, you need a way to gain access to those device for troubleshooting, configuration updates, and other operational tasks. Secure tunneling helps customers establish bidirectional communication to remote devices over a secure connection that is managed by AWS IoT. Secure tunneling does not require updates to your existing inbound firewall rule, so you can keep the same security level provided by firewall rules at a remote site.

For example, a sensor device located at a factory that is a couple hundred miles away is having trouble measuring the factory temperature. You can use secure tunneling to open and quickly start a session to that sensor device. After you have identified the problem (for example, a bad configuration file), you can reset the file and restart the sensor device through the same session. Compared to a more traditional troubleshooting (for example, sending a technician to the factory to investigate the sensor device), secure tunneling decreases incident response and recovery time and operational costs.

Secure tunneling concepts

Client access token (CAT)

A pair of tokens generated by secure tunneling when a new tunnel is created. The CAT is used by the source and destination devices to connect to the Secure Tunneling service.

Destination application

The application that runs on the destination device. For example, the destination application can be an SSH daemon for establishing an SSH session using secure tunneling.

Destination device

The remote device you want to access.

Device agent

An IoT application that connects to the AWS IoT device gateway and listens for new tunnel notifications over MQTT.

Local proxy

A software proxy that runs on the source and destination devices and relays a data stream between the Secure Tunneling service and the device application. The local proxy can be run in source mode or destination mode. For more information, see Local proxy (p. 652).

Source device

The device an operator uses to initiate a session to the destination device, usually a laptop or desktop computer.

Tunnel

A logical pathway through AWS IoT that enables bidirectional communication between a source device and destination device.

AWS IoT Secure Tunneling tutorial

In this tutorial, you open a tunnel and use it to start an SSH session to a remote device. The remote device is behind firewalls that block all inbound traffic, making direct SSH into the device impossible.

Before you begin, make sure that you understand how to register a device in the AWS IoT registry and connect a device to the AWS IoT device gateway.

Prerequisites

- The firewalls the remote device is behind must allow outbound traffic on port 443.
- You have created an IoT thing named `RemoteDeviceA` in the AWS IoT registry.
- You have an IoT device agent running on the remote device that connects to the AWS IoT device gateway and is configured with an MQTT topic subscription. This tutorial includes a snippet that shows you how to implement an agent. For more information, see IoT agent snippet (p. 656).
- You must have an SSH daemon running on the remote device.
- You have downloaded the local proxy source code from GitHub and built it for the platform of your choice. We'll refer to the built local proxy executable file as `localproxy` in this tutorial.

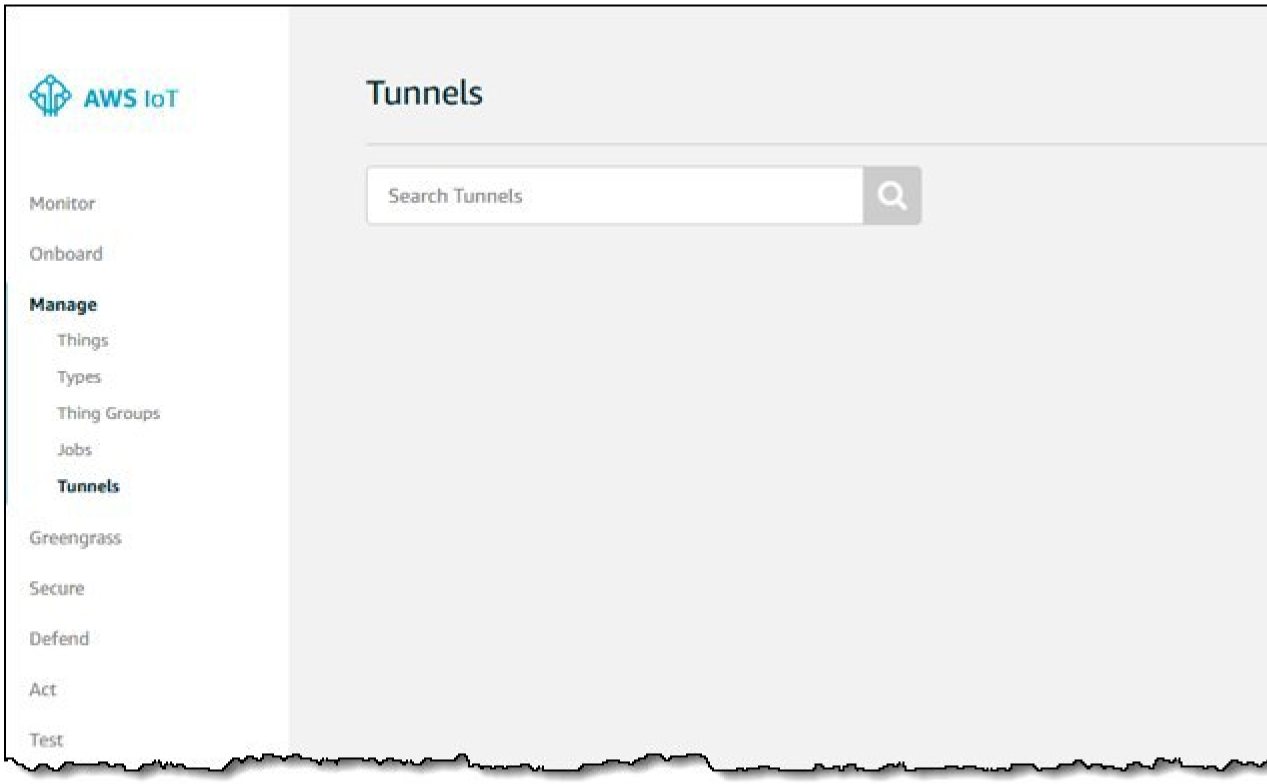
Open a tunnel

- The firewalls the remote device is behind must allow outbound traffic on port 443.
- You have created an IoT thing named `RemoteDeviceA` in the AWS IoT registry.
- You have an IoT device agent running on the remote device that connects to the AWS IoT device gateway and is configured with an MQTT topic subscription. This tutorial includes a snippet that shows you how to implement an agent. For more information, see IoT agent snippet (p. 656).
- You must have an SSH daemon running on the remote device.
- You have downloaded the local proxy source code from GitHub and built it for the platform of your choice. We'll refer to the built local proxy executable file as `localproxy` in this tutorial.

If you configure the destination when calling `OpenTunnel`, the Secure Tunneling service delivers the destination client access token to the remote device over MQTT and the reserved MQTT topic (`$aws/things/RemoteDeviceA/tunnels/notify`). For more information, see Reserved topics (p. 87). Upon receipt of the MQTT message, the IoT agent on the remote device starts the local proxy in destination mode. You can omit the destination configuration if you want to deliver the destination client access token to the remote device through another method. For more information, see Configuring a remote device (p. 657).

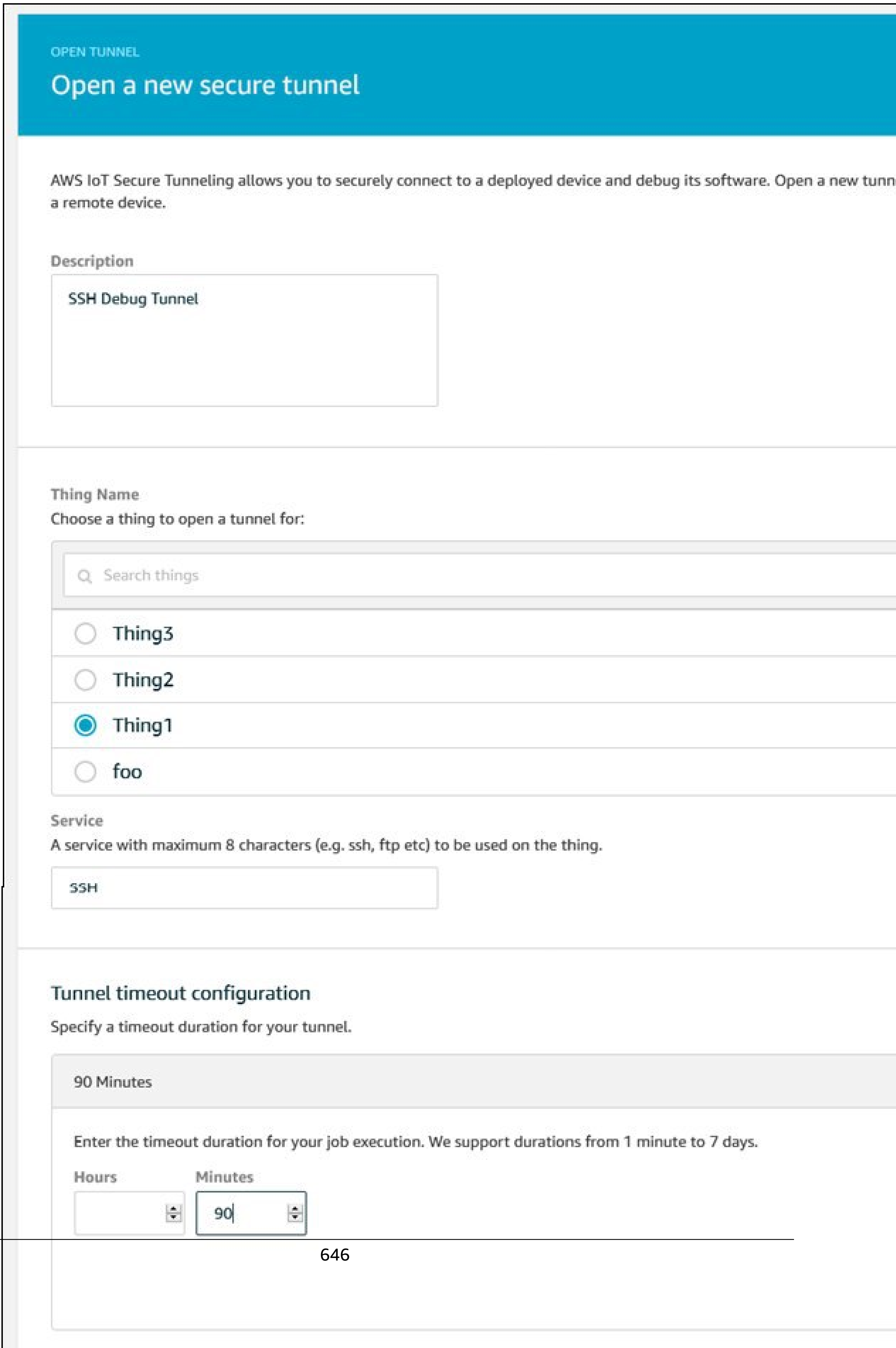
To open a tunnel in the console

1. In the AWS IoT console, navigate to **Manage** and **Tunnels**.



2. Select **Open New**.
3. On the **Open a new secure tunnel** screen, enter the following:

   a. Description: a description for your tunnel.
   b. Thing Name: the thing you want to open a tunnel for.
   c. Service: a service to be used on the thing (e.g. ssh, ftp etc).
   d. Tunnel timeout configuration: specify a timeout duration for your tunnel.
   e. Resource Tags: apply tags to your resources to help organize and identify them. Consists of a case-sensitive key-value pair.



OPEN TUNNEL

Open a new secure tunnel

AWS IoT Secure Tunneling allows you to securely connect to a deployed device and debug its software. Open a new tunn
a remote device.

Description

SSH Debug Tunnel

Thing Name

Choose a thing to open a tunnel for:

Search things

( ) Thing3

( ) Thing2

(●) Thing1

( ) foo

Service

A service with maximum 8 characters (e.g. ssh, ftp etc) to be used on the thing.

SSH

Tunnel timeout configuration

Specify a timeout duration for your tunnel.

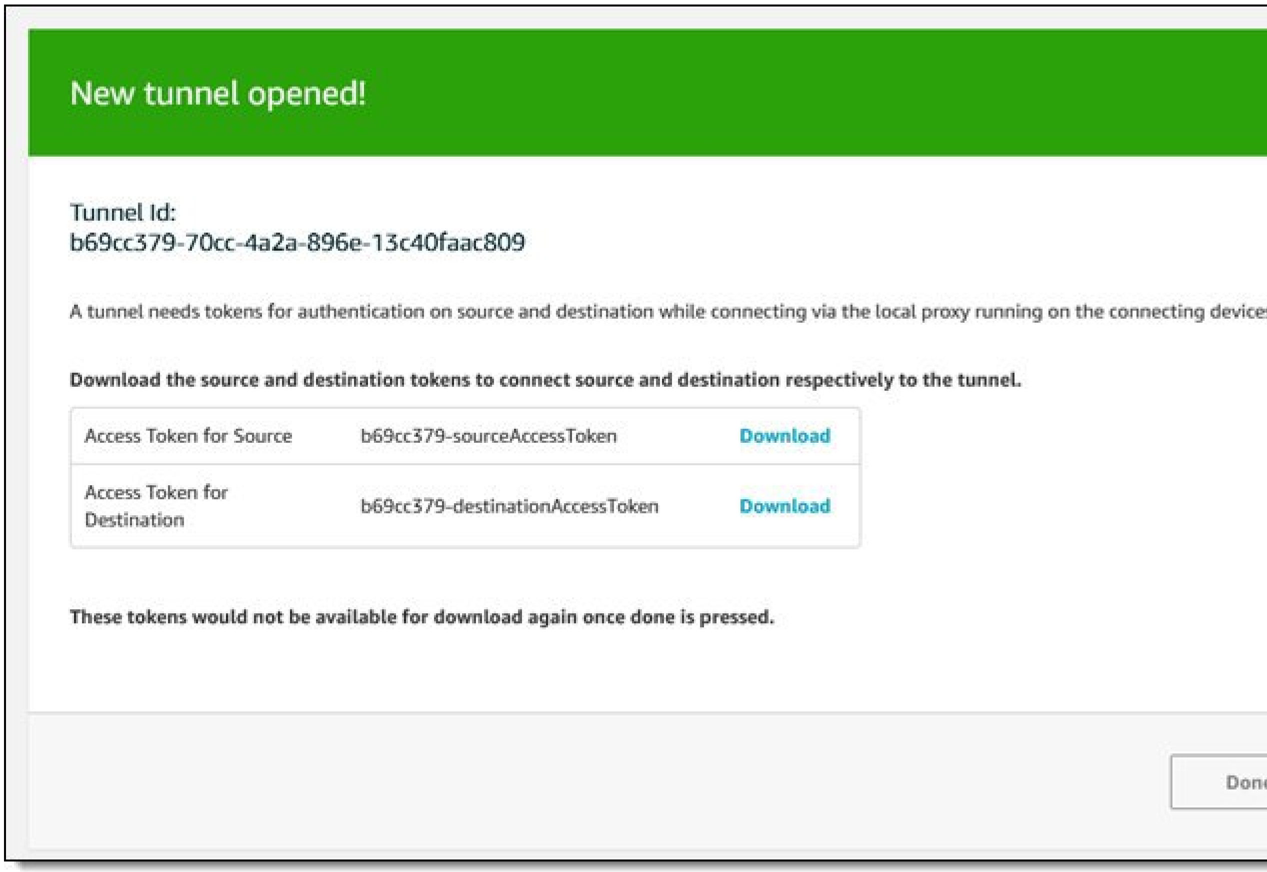
90 Minutes

Enter the timeout duration for your job execution. We support durations from 1 minute to 7 days.

Hours:

Minutes: 90

4. Download the client access tokens for the source and destination.



New tunnel opened!

Tunnel Id:
b69cc379-70cc-4a2a-896e-13c40faac809

A tunnel needs tokens for authentication on source and destination while connecting via the local proxy running on the connecting device

Download the source and destination tokens to connect source and destination respectively to the tunnel.

Access Token for Source	b69cc379-sourceAccessToken	Download
Access Token for Destination	b69cc379-destinationAccessToken	Download

These tokens would not be available for download again once done is pressed.

5. Select **Done**.

Start the local proxy

Open a terminal on your laptop, copy the source client access token, and use it to start the local proxy in source mode. In the following command, the local proxy is configured to listen for new connections on port 5555.

./localproxy -r us-east-1 -s 5555 -t ***source-client-access-token***

> **Note**
> The AWS Region in this command must be the same AWS Region where the tunnel was created.

-r

Specifies the AWS Region where your tunnel is created.

-s

Specifies the port to which the proxy should connect.

-t

Specifies the client token text.

> **Note**
> If you receive the following error, set up the CA path. For information, see GitHub.

```
Could not perform SSL handshake with proxy server: certificate verify
failed
```

Start an SSH session

Open another terminal and use the following command to start a new SSH session by connecting to the local proxy on port 5555.

ssh *username*@localhost -p 5555

You might be prompted for a password for the SSH session. When you are done with the SSH session, type **`exit`** to close the session.

Closing the tunnel

1. Open the AWS IoT console.
2. Choose the tunnel and from **Actions**, choose **Close**. Closing the tunnel causes both local proxy instances to close.

Secure tunnel lifecycle

AWS IoT Secure Tunneling demo

To quickly demo AWS IoT Secure Tunneling, use our AWS IoT Secure Tunneling demo on GitHub.

Tunnels can have one of the following statuses:

- `OPEN`
- `CLOSED`

Connections can have one of the following statuses:

- `CONNECTED`
- `DISCONNECTED`

When you open a tunnel, it has a status of `OPEN`. The tunnel's source and destination connection status is set to `DISCONNECTED`. When a device (source or destination) connects to the tunnel, the corresponding connection status changes to `CONNECTED`, while the tunnel status remains `OPEN`. When a device disconnects from the tunnel, the corresponding connection status changes back to `DISCONNECTED`. A device can connect to and disconnect from a tunnel repeatedly as long as the tunnel remains `OPEN`.

A tunnel's status becomes `CLOSED` when you call `CloseTunnel` or the tunnel remains `OPEN` for longer than the `MaxLifetimeTimeout` value. You can configure `MaxLifetimeTimeout` when calling `OpenTunnel`. `MaxLifetimeTimeout` defaults to 12 hours if you do not specify a value.

After a tunnel is `CLOSED`, it might be visible in the AWS IoT console for at least three hours before it is deleted. While the tunnel is visible, you can call `DescribeTunnel` and `ListTunnels` to view tunnel metadata. A tunnel cannot be reopened when it is `CLOSED`. The Secure Tunneling service rejects attempts to connect to a `CLOSED` tunnel.

Controlling access to tunnels

The Secure Tunneling service provides the following service-specific actions, resources, and condition context keys for use in IAM permission policies.

Tunnel access prerequisites

- Learn how to secure AWS resources by using IAM policies.
- Learn how to create and evaluate IAM conditions.
- Learn how to secure AWS resources using resource tags.

iot:OpenTunnel

The `iot:OpenTunnel` policy action grants a principal permission to call OpenTunnel. You must specify the wildcard tunnel ARN `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*` in the `Resource` element of the IAM policy statement. You can specify a thing ARN (`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:thing/ `*`<thing-name>`*) in the `Resource` element of the IAM policy statement to manage `OpenTunnel` permission for specific IoT things.

For example, the following policy statement allows you to open a tunnel to the IoT thing named `TestDevice`.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

The `iot:OpenTunnel` policy action supports the following condition keys:

- `iot:ThingGroupArn`
- `iot:TunnelDestinationService`
- `aws:RequestTag/`*`tag-key`*
- `aws:TagKeys`

The following policy statement allows you to open a tunnel to the thing if the thing belongs to a thing group with a name that starts with `TestGroup` and the configured destination service on the tunnel is SSH.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ]
        }
    }
}
```

You can also use resource tags to control permission to open tunnels. For example, the following policy statement allows a tunnel to be opened if the tag key `Owner` is present with a value of `Admin` and no other tags are specified.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/Owner": "Admin"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "Owner"
        }
    }
}
```

iot:DescribeTunnel

The `iot:DescribeTunnel` policy action grants a principal permission to call DescribeTunnel. You can specify a fully qualified tunnel ARN (for example, `arn:aws:iot:`*`aws-region`*`:` *`aws-account-id`*`:tunnel/`*`tunnel-id`*) or use the wildcard tunnel ARN (`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`) in the `Resource` element of the IAM policy statement.

The `iot:DescribeTunnel` policy action supports the following condition key:

- `aws:ResourceTag/`*`tag-key`*

The following policy statement allows you to call `DescribeTunnel` if the requested tunnel is tagged with the key `Owner` with a value of `Admin`.

```
{
    "Effect": "Allow",
    "Action": "iot:DescribeTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/Owner": "Admin"
        }
    }
}
```

iot:ListTunnels

The `iot:ListTunnels` policy action grants a principal permission to call ListTunnels. You must specify the wildcard tunnel ARN (`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`) in the `Resource` element of the IAM policy statement. To manage `ListTunnels` permission on selected IoT things, you can also specify a thing ARN (`arn:aws:iot:aws-region:aws-account-id:thing/thing-name`) in the `Resource` element of the IAM policy statement.

The following policy statement allows you to list tunnels for the thing named `TestDevice`.

```
{
    "Effect": "Allow",
    "Action": "iot:ListTunnels",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:ListTagsForResource

The `iot:ListTagsForResource` policy action grants a principal permission to call `ListTagsForResource`. You can specify a fully qualified tunnel ARN (`arn:aws:iot:aws-region:aws-account-id:tunnel/tunnel-id`) or use the wildcard tunnel ARN (`arn:aws:iot:aws-region:aws-account-id:tunnel/*`) in the `Resource` element of the IAM policy statement.

iot:CloseTunnel

The `iot:CloseTunnel` policy action grants a principal permission to call CloseTunnel. You can specify a fully qualified tunnel ARN (`arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`) or use the wildcard tunnel ARN (`arn:aws:iot:aws-region:aws-account-id:tunnel/*`) in the `Resource` element of the IAM policy statement.

The `iot:CloseTunnel` policy action supports the following condition keys:

- `iot:Delete`
- `aws:ResourceTag/tag-key`

The following policy statement allows you to call `CloseTunnel` if the request's `Delete` parameter is `false` and the requested tunnel is tagged with the key `Owner` with a value of `QATeam`.

```
{
    "Effect": "Allow",
    "Action": "iot:CloseTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "Bool": {
            "iot:Delete": "false"
        },
        "StringEquals": {
            "aws:ResourceTag/Owner": "QATeam"
        }
    }
}
```

iot:TagResource

The `iot:TagResource` policy action grants a principal permission to call `TagResource`. You can specify a fully qualified tunnel ARN (`arn:aws:iot:aws-region: aws-account-`

id:tunnel/*tunnel-id*) or use the wildcard tunnel ARN (arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*) in the Resource element of the IAM policy statement.

iot:UntagResource

The iot:UntagResource policy action grants a principal permission to call UntagResource. You can specify a fully qualified tunnel ARN (arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*tunnel-id*) or use the wildcard tunnel ARN (arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*) in the Resource element of the IAM policy statement.

For more information about AWS IoT security see Identity and access management for AWS IoT (p. 277).

Local proxy

The local proxy is a process that acts as the recipient or sender of incoming TCP connections. It transmits data sent by the device application through the Secure Tunneling service over a WebSocket secure connection. You can download the local proxy source from GitHub. The local proxy can run in two modes: source or destination. In source mode, the local proxy runs on the same device or network as the client application that initiates the TCP connection. In destination mode, the local proxy runs on the remote device, along with the destination application. Currently, a single tunnel can support only one TCP connection at a time.

The local proxy first establishes a connection to the Secure Tunneling service. When you start the local proxy, use the -r argument to specify the AWS Region in which the tunnel is opened. Use the -t argument to pass either the source or destination client access token returned from the OpenTunnel. Two local proxies using the same client access token value cannot be connected at the same time.

After the WebSocket connection is established, the local proxy performs either source mode or destination mode behaviors, depending on its configuration.

By default, the local proxy attempts to reconnect to the Secure Tunneling service if any I/O errors occur or if the WebSocket connection is closed unexpectedly. This causes the TCP connection to close. If any TCP socket errors occur, the local proxy sends a message through the tunnel to notify the other side to close its TCP connection. By default, the local proxy always uses SSL communication.

After you use the tunnel, it is safe to stop the local proxy process. We recommend that you explicitly close the tunnel by calling CloseTunnel. Active tunnel clients might not be closed immediately after calling CloseTunnel.

Local proxy security best practices

When running the local proxy, follow these security best practices:

- Avoid the use of the -t local proxy argument to pass in an access token. We recommend that you use the AWSIOT_TUNNEL_ACCESS_TOKEN environment variable to set the access token for the local proxy.
- Run the local proxy executable with least privileges in the operating system or environment.
  - Avoid running the local proxy as an administrator on Windows.
  - Avoid running the local proxy as root on Linux and macOS.
- Consider running the local proxy on separate hosts, containers, sandboxes, chroot jail, or a virtualized environment.
- Build the local proxy with relevant security flags, depending on your toolchain.
- On devices with multiple network interfaces, use the -b argument to bind the TCP socket to the network interface used to communicate with the destination application.

Multiplex data streams in a secure tunnel

You can use multiple data streams per tunnel by using the Secure Tunneling multiplexing feature. With multiplexing, you can troubleshoot devices using multiple connections or ports (for example, a web browser that requires sending multiple HTTP and SSH data streams). You can also reduce your operational load by eliminating the need to build, deploy, and start multiple local proxies or open multiple tunnels to the same device.

Example use case

You can use the multiplexing feature in the event that a device in the field requires more than one connection to the device in order to properly troubleshoot it. For example, you might need to connect to an on-device web application to change some networking parameters, while simultaneously issuing shell commands through the terminal to verify that the device is working properly with the new networking parameters. In this scenario, you may need to connect to the device through both HTTP and SSH and transfer two parallel data streams in order to concurrently access the web application and terminal. With the multiplexing feature, these two independent streams can be transferred over the same tunnel at the same time.

How to set up a multiplexed tunnel

The following procedure will walk you through how to set up a multiplexed tunnel for troubleshooting devices using applications that require connections to multiple ports. You will set up one tunnel with two multiplexed streams: one HTTP stream and one SSH stream.

1. First, configure the destination device with configuration files. Configuration files can be provided on the device if the port mappings are unlikely to change. On the destination device, create a configuration directory called `config` in the same folder where the local proxy is running. Then, create a file called `SSHSource.ini` in this directory. The content of this file is:

```
HTTP1 = 5555
SSH1 = 3333
```

   Note
   You can skip this step if you prefer to specify the port mapping through the CLI or don't need to start local proxy on designated listening ports.

2. Next, configure the source device with configuration files. In the same folder as where the local proxy is running, create a configuration directory called `config` and give local proxy the read permission to this directory. Then, create a file called `SSHDestination.ini` in this directory. The content of this file is:

```
HTTP1 = 80
SSH1 = 22
```

   Note
   You can skip this step if you prefer to specify the port mapping through the CLI. If so, you will need to update the tunnel agent (p. 656) to use the new parameters.

3. Open a tunnel with the service identifier `HTTP1` and `SSH1`. `thingName` is optional if your device isn't registered with AWS IoT.

```
aws iotsecuretunneling open-tunnel \
--destination-config thingName=foo,services=HTTP1,SSH1
```

A destination and source client access token will be given after this call. Note the `destination_client_access_token` and `source_client_access_token` for next steps. The output should look similar to the following.

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

4. Next, start the destination local proxy. You will be connected to the secure tunneling service upon token delivery. A local proxy running on destination devices starts in destination mode. You have two options to achieve this:

   1. Start destination local proxy with configuration files from Step 1.

   ```
   ./localproxy -r us-east-1 -m dst -t destination_client_access_token
   ```

   2. Start destination local proxy with the mapping specified through the CLI.

   ```
   ./localproxy -r us-east-1 -d HTTP1=80,SSH1=22 -t destination_client_access_token
   ```

5. Now, start the source local proxy. A local proxy running on source devices starts in source mode. You have three options to achieve this:

   1. Start source local proxy with configuration files from Step 2.

   ```
   ./localproxy -r us-east-1 -m src -t source_client_access_token
   ```

   2. Start source local proxy with the mapping specified through the CLI.

   ```
   ./localproxy -r us-east-1 -s HTTP1=5555,SSH1=3333 -t source_client_access_token
   ```

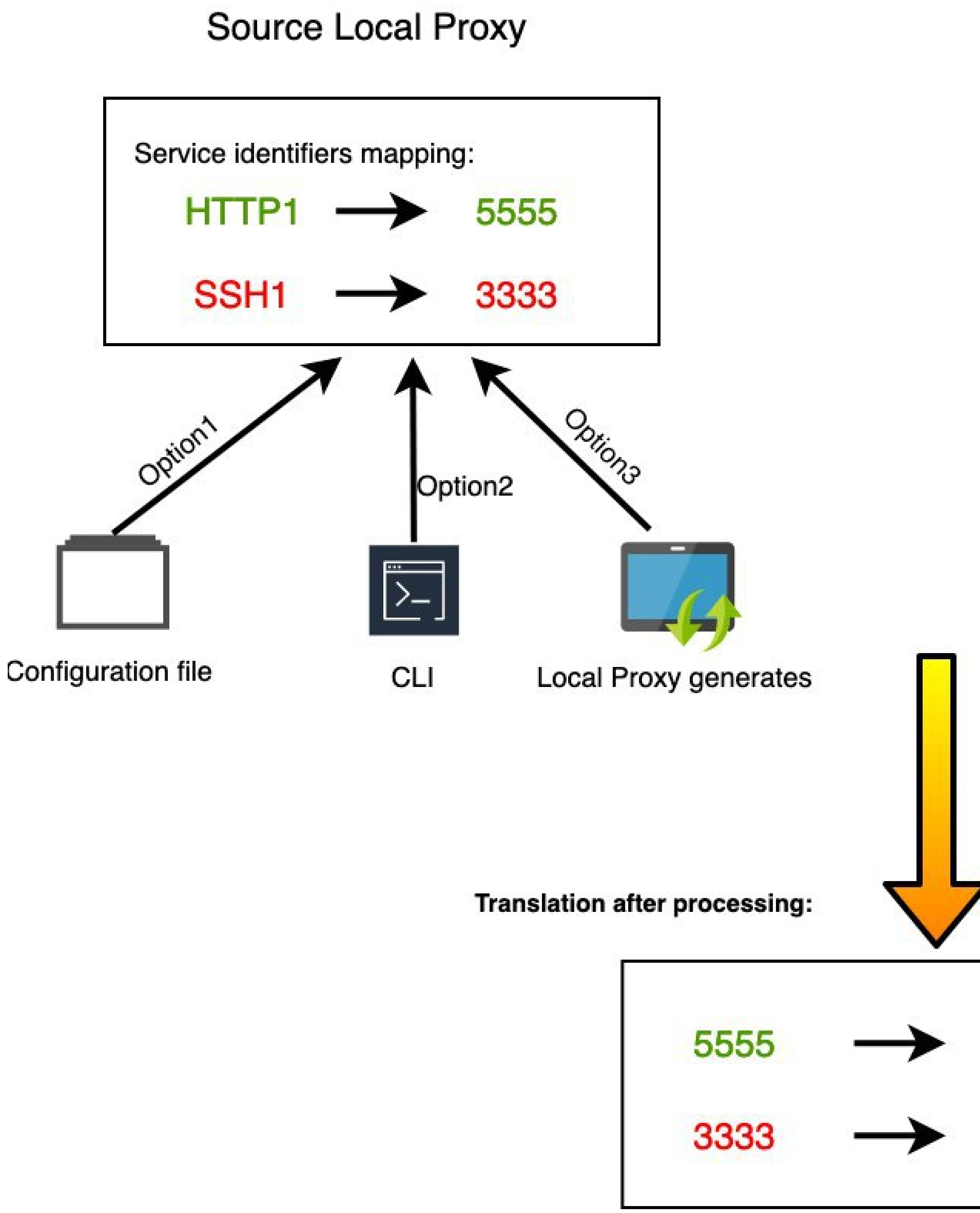
   3. Start source local proxy with no configuration files and no mapping specified from the CLI. Local proxy will pick up available ports to use and manage the mappings for you.

   ```
   ./localproxy -r us-east-1 -m src -t source_client_access_token
   ```

6. Application data from SSH and HTTP connection can now be transferred concurrently over the multiplexed tunnel. As can be seen from the map below, the service identifier acts as a readable format to translate the port mapping between the source and destination device. With this configuration, the secure tunneling service will:

   1. Forward any incoming HTTP traffic from port 5555 on the source device to port 80 on the destination device.

   2. Forward any incoming SSH traffic from port 3333 on the source device to port 22 on the destination device.


Source Local Proxy
Service identifiers mapping:
HTTP1 → 5555
SSH1 → 3333
Option1
Option2
Option3
Configuration file
CLI
Local Proxy generates
Translation after processing:
5555 →
3333 →

IoT agent snippet

The IoT agent is used to receive the MQTT message that includes the client access token and start a local proxy on the remote device. You must install and run the IoT agent on the remote device if you want the Secure Tunneling service to deliver the client access token. The IoT agent must subscribe to the following reserved IoT MQTT topic:

`$aws/things/thing-name/tunnels/notify`

Where `thing-name` is the name of IoT thing associated with the remote device.

The following is an example MQTT message payload:

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

After it receives an MQTT message, the IoT agent must start a local proxy on the remote device with the appropriate parameters.

The following Java code demonstrates how to use the AWS IoT Device SDK and ProcessBuilder from the Java library to build a simple IoT agent to work with the Secure Tunneling service.

```
// Find the IoT device endpoint for your AWS account
final String endpoint = iotClient.describeEndpoint(new
 DescribeEndpointRequest().withEndpointType("iot:Data-ATS")).getEndpointAddress();

// Instantiate the IoT Agent with your AWS credentials
final String thingName = "RemoteDeviceA";
final String tunnelNotificationTopic = String.format("$aws/things/%s/tunnels/notify",
 thingName);
final AWSIotMqttClient mqttClient = new AWSIotMqttClient(endpoint, thingName,
                 "your_aws_access_key", "your_aws_secret_key");

try {
    mqttClient.connect();
    final TunnelNotificationListener listener = new
 TunnelNotificationListener(tunnelNotificationTopic);
    mqttClient.subscribe(listener, true);
}
finally {
    mqttClient.disconnect();
}

private static class TunnelNotificationListener extends AWSIotTopic {
    public TunnelNotificationListener(String topic) {
        super(topic);
    }

    @Override
    public void onMessage(AWSIotMessage message) {
        try {
            // Deserialize the MQTT message
            final JSONObject json = new JSONObject(message.getStringPayload());

            final String accessToken = json.getString("clientAccessToken");
            final String region = json.getString("region");
            final String clientMode = json.getString("clientMode");
            if (!clientMode.equals("destination")) {
                throw new RuntimeException("Client mode " + clientMode + " in the MQTT
 message is not expected");
            }

            final JSONArray servicesArray = json.getJSONArray("services");
            if (servicesArray.length() > 1) {
                throw new RuntimeException("Services in the MQTT message has more than 1
 service");
            }
            final String service = servicesArray.get(0).toString();
            if (!service.equals("SSH")) {
                throw new RuntimeException("Service " + service + " is not supported");
            }

            // Start the destination local proxy in a separate process to connect to the
 SSH Daemon listening port 22
            final ProcessBuilder pb = new ProcessBuilder("localproxy",
                        "-t", accessToken,
                        "-r", region,
                        "-d", "localhost:22");
            pb.start();
        }
        catch (Exception e) {
            log.error("Failed to start the local proxy", e);
        }
    }
}
```

Configuring a remote device

If you want to deliver the destination client access token to the remote device through methods other than subscribing to the reserved IoT MQTT topic, you might need two components on the remote device:

- A destination client access token listener.
- A local proxy.

The destination client access token listener should work with the client access token delivery mechanism of your choice. It must be able to start a local proxy in destination mode.

Device provisioning

AWS provides several different ways to provision a device and install unique client certificates on it. This section describes each way and how to select the best one for your IoT solution.

Select the option that fits your situation best

- **You can install certificates on IoT devices before they are delivered**

  If you can securely install unique client certificates on your IoT devices before they are delivered for use by the end user, you want to use *just-in-time* provisioning (JITP) (p. 666) or *just-in-time* registration (JITR) (p. 214).

  Using JITP and JITR, the certificate authority (CA) used to sign the device certificate is registered with AWS IoT and is recognized by AWS IoT when the device first connects. The device is provisioned in AWS IoT on its first connection using the details of its provisioning template.

  For more information on single thing, JITP, JITR, and bulk provisioning of devices that have unique certificates, see the section called "Provisioning devices that have device certificates" (p. 665).
- **End users or installers can use an app to install certificates on their IoT devices**

  If you cannot securely install unique client certificates on your IoT device before they are delivered to the end user, but the end user or an installer can use an app to register the devices and install the unique device certificates, you want to use the provisioning by trusted user (p. 661) process.

  Using a trusted user, such as an end user or an installer with a known account, can simplify the device manufacturing process. Instead of a unique client certificate, devices have a temporary certificate that enables the device to connect to AWS IoT for only 5 minutes. During that 5-minute window, the trusted user obtains a unique client certificate with a longer life and installs it on the device. The limited life of the claim certificate minimizes the risk of a compromised certificate.

  For more information, see the section called "Provisioning by trusted user" (p. 661).
- **End users CANNOT use an app to install certificates on their IoT devices**

  If neither of the previous options will work in your IoT solution, the provisioning by claim (p. 660) process is an option. With this process, your IoT devices have a claim certificate that is shared by other devices in the fleet. The first time a device connects with a claim certificate, AWS IoT registers the device using its provisioning template and issues the device its unique client certificate for subsequent access to AWS IoT

  This option enables automatic provisioning of a device when it connects to AWS IoT, but could present a larger risk in the event of a compromised claim certificate. If a claim certificate becomes compromised, you can deactivate the certificate. Deactivating the claim certificate prevents all devices with that claim certificate from being registered in the future. However; deactivating the claim certificate does not block devices that have already been provisioned.

  For more information, see the section called "Provisioning by claim" (p. 660).

Provisioning devices in AWS IoT

When you provision a device with AWS IoT, you must create resources so your devices and AWS IoT can communicate securely. Other resources can be created to help you manage your device fleet. The following resources can be created during the provisioning process:

- An IoT thing.

  IoT things are entries in the AWS IoT device registry. Each thing has a unique name and set of attributes, and is associated with a physical device. Things can be defined using a thing type or grouped into thing groups. For more information, see Managing devices with AWS IoT (p. 172).

  Although not required, creating a thing makes it possible to manage your device fleet more effectively by searching for devices by thing type, thing group, and thing attributes. For more information, see Fleet indexing service (p. 684).
- An X.509 certificate.

  Devices use X.509 certificates to perform mutual authentication with AWS IoT. You can register an existing certificate or have AWS IoT generate and register a new certificate for you. You associate a certificate with a device by attaching it to the thing that represents the device. You must also copy the certificate and associated private key onto the device. Devices present the certificate when connecting to AWS IoT. For more information, see Authentication (p. 200).
- An IoT policy.

  IoT policies define the operations a device can perform in AWS IoT. IoT policies are attached to device certificates. When a device presents the certificate to AWS IoT, it is granted the permissions specified in the policy. For more information, see Authorization (p. 233). Each device needs a certificate to communicate with AWS IoT.

AWS IoT supports automated fleet provisioning using provisioning templates. Provisioning templates describe the resources AWS IoT requires to provision your device. Templates contain variables that enable you to use one template to provision multiple devices. When you provision a device, you specify values for the variables specific to the device using a dictionary or *map*. To provision another device, specify new values in the dictionary.

You can use automated provisioning whether or not your devices have unique certificates (and their associated private key).

Fleet provisioning APIs

There are several categories of APIs used in fleet provisioning:

- These control plane functions create and manage the fleet provisioning templates and configure trusted user policies.
  - CreateProvisioningTemplate
  - CreateProvisioningTemplateVersion
  - DeleteProvisioningTemplate
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioningTemplate
  - DescribeProvisioningTemplateVersion
  - ListProvisioningTemplates
  - ListProvisioningTemplateVersions
  - UpdateProvisioningTemplate
- Trusted users can use this control plane function to generate a temporary onboarding claim. This temporary claim is passed to the device during Wi-Fi config or similar method.
  - CreateProvisioningClaim.
- The MQTT API used during the provisioning process by devices with a provisioning claim certificate embedded in a device, or passed to it by a trusted user.

- the section called “CreateCertificateFromCsr” (p. 679)
- the section called “CreateKeysAndCertificate” (p. 680)
- the section called “RegisterThing” (p. 682)

Provisioning devices that don't have device certificates using fleet provisioning

By using AWS IoT fleet provisioning, AWS IoT can generate and securely deliver device certificates and private keys to your devices when they connect to AWS IoT for the first time. AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA).

There are two ways to use fleet provisioning:

- By claim
- By trusted user

Provisioning by claim

Devices can be manufactured with a provisioning claim certificate and private key (which are special purpose credentials) embedded in them. If these certificates are registered with AWS IoT, the service can exchange them for unique device certificates that the device can use for regular operations. This process includes the following steps:

Before you deliver the device

1. Call `CreateProvisioningTemplate` to create a provisioning template. This API returns a template ARN. For more information, see Device provisioning MQTT API (p. 678).

   You can also create a fleet provisioning template in the AWS IoT console.

   a. From the navigation pane, choose **Onboard**, then choose **Fleet provisioning templates**.
   b. Choose **Create** and follow the prompts.
2. Create certificates and associated private keys to be used as provisioning claim certificates.
3. Register these certificates with AWS IoT and associate an IoT policy that restricts the use of the certificates. The following example IoT policy restricts the use of the certificate associated with this policy to provisioning devices.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Connect"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["iot:Publish","iot:Receive"],
            "Resource": [
                "arn:aws:iot:aws-region:aws-account-id:topic/$aws/certificates/create/
*",
                "arn:aws:iot:aws-region:aws-account-id:topic/$aws/provisioning-
templates/templateName/provision/*"
```

```
                ]
            },
            {
                "Effect": "Allow",
                "Action": "iot:Subscribe",
                "Resource": [
                    "arn:aws:iot:aws-region:aws-account-id:topicfilter/$aws/certificates/
create/*",
                    "arn:aws:iot:aws-region:aws-account-id:topicfilter/$aws/provisioning-
templates/templateName/provision/*"
                ]
            }
        ]
}
```

4. Give the AWS IoT service permission to create or update IoT resources such as things and certificates in your account when provisioning devices. Do this by attaching the `AWSIoTThingsRegistration` managed policy to an IAM role (called the provisioning role) that trusts the AWS IoT service principal.
5. Manufacture the device with the provisioning claim certificate securely embedded in it.

The device is now ready to be delivered to where it will be installed for use.

Important
Provisioning claim private keys should be secured at all times, including on the device. We recommend that you use AWS IoT CloudWatch metrics and logs to monitor for indications of misuse. If you detect misuse, disable the provisioning claim certificate so it cannot be used for device provisioning.

To initialize the device for use

1. The device uses the AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client (p. 990) to connect to and authenticate with AWS IoT using the provisioning claim certificate that is installed on the device.
2. The device obtains a permanent certificate and private key by using one of these options. The device will use the certificate and key for all future authentication with AWS IoT.

   a. Call `CreateKeysAndCertificate` (p. 680) to create a new certificate and private key using the AWS certificate authority.

      Or

   b. Call `CreateCertificateFromCsr` (p. 679) to generate a certificate from a certificate signing request that keeps its private key secure.
3. From the device, call `RegisterThing` (p. 682) to register the device with AWS IoT and create cloud resources.

   The Fleet Provisioning service creates cloud resources such as things, thing groups, and attributes, as defined in the provisioning template.
4. After saving the permanent certificate on the device, the device must disconnect from the session that it initiated with the provisioning claim certificate and reconnect using the permanent certificate.

The device is now ready to communicate normally with AWS IoT.

Provisioning by trusted user

In many cases, a device connects to AWS IoT for the first time when a trusted user, such as an end user or installation technician, uses a mobile app to configure the device in its deployed location.

Important
You must manage the trusted user's access and permission to perform this procedure. One way to do this is to provide and maintain an account for the trusted user that authenticates them and grants them access to the AWS IoT features and APIs required to perform this procedure.

Before you deliver the device

1. Call `CreateProvisioningTemplate` to create a provisioning template and return its *`templateArn`* and *`templateName`*.
2. Create an IAM role that is used by a trusted user to initiate the provisioning process. The provisioning template allows only that user to provision a device. For example:

```
{
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim",
    ],
    "Resource": [
        "arn:aws:aws-region:aws-account-id:provisioningtemplate/templateName"
    ]
}
```

3. Give the AWS IoT service permission to create or update IoT resources, such as things and certificates in your account when provisioning devices. You do this by attaching the `AWSIoTThingsRegistration` managed policy to an IAM role (called the *provisioning role*) that trusts the AWS IoT service principal.
4. Provide the means to identify your trusted users, such as by providing them with an account that can authenticate them and authorize their interactions with the AWS APIs necessary to register their devices.

To initialize the device for use

1. A trusted user signs in to your provisioning mobile app or web service.
2. The mobile app or web application uses the IAM role and calls `CreateProvisioningClaim` to obtain a temporary provisioning claim certificate from AWS IoT.

   Note
   For security, the temporary provisioning claim certificate returned by `CreateProvisioningClaim` is valid for only five minutes. The following steps must successfully return a valid certificate before the temporary provisioning claim certificate expires. Temporary provisioning claim certificates do not appear in your account's list of certificates.
3. The mobile app or web application supplies the temporary provisioning claim certificate to the device along with any required configuration information, such as Wi-Fi credentials.
4. The device uses the temporary provisioning claim certificate to connect to AWS IoT using the AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client (p. 990).

   Note
   The device must perform the following steps such that it calls `RegisterThing` (p. 682) within five minutes of connecting to AWS IoT with the temporary provisioning claim certificate.
5. The device obtains a permanent certificate and private key by using one of these options. The device will use the certificate and key for all future authentication with AWS IoT.

   a. Call `CreateKeysAndCertificate` (p. 680) to create a new certificate and private key using the AWS certificate authority.

   Or

   b. Call `CreateCertificateFromCsr` (p. 679) to generate a certificate from a certificate signing request that keeps its private key secure.

6. The device calls `RegisterThing` (p. 682) to register the device with AWS IoT and create cloud resources. Remember that this must occur within five minutes of connecting to AWS IoT with the temporary provisioning claim certificate.

   The Fleet Provisioning service creates cloud resources such as IoT things, thing groups, and attributes, as defined in the provisioning template.
7. After saving the permanent certificate on the device, the device must disconnect from the session that it initiated with the temporary provisioning claim certificate and reconnect using the permanent certificate.

The device is now ready to communicate normally with AWS IoT.

Using pre-provisioning hooks with the AWS CLI

The following procedure creates a provisioning template with pre-provisioning hooks. The Lambda function used here is an example that can be modified.

To create and apply a pre-provisioning hook to a provisioning template

1. Create a Lambda function that has a defined input and output. Lambda functions are highly customizable the `allowProvisioning` and `parameterOverrides` are required for creating pre-provisioning hooks. For more information about creating Lambda functions, see Using AWS Lambda with the AWS Command Line Interface.

   The following is an example of a Lambda function output:

```
{
  "allowProvisioning": True,
  "parameterOverrides": {
    "incomingKey0": "incomingValue0",
    "incomingKey1": "incomingValue1"
  }
}
```

2. AWS IoT uses resource-based policies to call Lambda, so you must give AWS IoT permission to call your Lambda function.

   The following is an example using add-permission give IoT permission to your Lambda.

```
aws lambda add-permission \
    --function-name myLambdaFunction \
    --statement-id iot-permission \
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. Add a pre-provisioning hook to a template using either the create-provisioning-template or update-provisioning-template command.

   The following CLI example uses the create-provisioning-template to create a provisioning template that has pre-provisioning hooks:

```
aws iot create-provisioning template \
    --template-name myTemplate \
    --provisioning-role-arn arn:aws:iam:us-east-1:1234564789012:role/myRole \
    --template-body file://template.json \
    --pre-provisioning-hook file://hooks.json
```

The output of this command looks like the following:

```
{
    "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
myTemplate",
    "defaultVersionId": 1,
    "templateName": myTemplate
}
```

You can also load a parameter from a file instead of typing it all as a command line parameter value to save time. For more information, see Loading AWS CLI Parameters from a File. The following shows the `template` parameter in expanded JSON format:

```
{
    "Parameters" : {
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingTypeName" : {"Fn::Join":["",["ThingTypePrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["widgets", "WA"],
                "BillingGroup": "BillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:504350838278:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
    }
}
```

The following shows the `pre-provisioning-hook` parameter in expanded JSON format:

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

Provisioning devices that have device certificates

AWS IoT provides three ways to provision devices when they already have a device certificate (and associated private key) on them:

- Single-thing provisioning with a provisioning template. This is a good option if you only need to provision devices one at a time.
- Just-in-time provisioning (JITP) with a template that provisions a device when it first connects to AWS IoT. This is a good option if you need to register large numbers of devices, but you don't have information about them that you can assemble into a bulk provisioning list.
- Bulk registration. This option allows you to specify a list of single-thing provisioning template values that are stored in a file in an S3 bucket. This approach works well if you have a large number of known devices whose desired characteristics you can assemble into a list.

Topics

Single thing provisioning

To provision a thing, use the RegisterThing API or the `register-thing` CLI command. The `register-thing` CLI command takes the following arguments:

--template-body

The provisioning template.

--parameters

A list of name-value pairs for the parameters used in the provisioning template, in JSON format (for example, `{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`).

See Provisioning templates (p. 669).

RegisterThing or `register-thing` returns the ARNs for the resources and the text of the certificate it created:

```
{
    "certificatePem": "certificate-text",
    "resourceArns": {
    "PolicyLogicalName": "arn:aws:iot:us-
west-2:123456789012:policy/2A6577675B7CD1823E271C7AAD8184F44630FFD7",
    "certificate": "arn:aws:iot:us-west-2:123456789012:cert/
cd82bb924d4c6ccbb14986dcb4f40f30d892cc6b3ce7ad5008ed6542eea2b049",
    "thing": "arn:aws:iot:us-west-2:123456789012:thing/MyProvisionedThing"
    }
}
```

If a parameter is omitted from the dictionary, the default value is used. If no default value is specified, the parameter is not replaced with a value.

Just-in-time provisioning

You can have your devices provisioned when they first attempt to connect to AWS IoT. Just-in-time provisioning (JITP) settings are made on CA certificates. To provision the device, you must enable automatic registration and associate a provisioning template with the CA certificate used to sign the device certificate. Provisioning successes and errors are logged as Device provisioning metrics (p. 329) in Amazon CloudWatch.

You can make these settings when you register a CA certificate with the RegisterCACertificate API or the `register-ca-certificate` CLI command:

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --verification-cert
                file://your-verification-cert --set-as-active --allow-auto-registration --
registration-config file://your-template
```

For more information, see Registering a CA Certificate.

You can also use the UpdateCACertificate API or the `update-ca-certificate` CLI command to update the settings for a CA certificate:

```
aws iot update-ca-certificate --cert-id caCertificateId --new-auto-registration-status
 ENABLE --registration-config file://your-template
```

Note
JITP calls other AWS IoT control plane APIs during the provisioning process. These calls might exceed the AWS IoT Throttling Quotas set for your account and result in throttled calls. Contact AWS Customer Support to raise your throttling quotas, if necessary.

When a device attempts to connect to AWS IoT by using a certificate signed by a registered CA certificate, AWS IoT loads the template from the CA certificate and uses it to call RegisterThing (p. 682). The JITP workflow first registers a certificate with a status value of

PENDING_ACTIVATION. When the device provisioning flow is complete, the status of the certificate is changed to ACTIVE.

AWS IoT defines the following parameters that you can declare and reference in provisioning templates:

- `AWS::IoT::Certificate::Country`
- `AWS::IoT::Certificate::Organization`
- `AWS::IoT::Certificate::OrganizationalUnit`
- `AWS::IoT::Certificate::DistinguishedNameQualifier`
- `AWS::IoT::Certificate::StateName`
- `AWS::IoT::Certificate::CommonName`
- `AWS::IoT::Certificate::SerialNumber`
- `AWS::IoT::Certificate::Id`

The values for these provisioning template parameters are limited to what JITP can extract from the subject field of the certificate of the device being provisioned. The certificate must contain values for all of the parameters in the template body. The `AWS::IoT::Certificate::Id` parameter refers to an internally generated ID, not an ID that is contained in the certificate. You can get the value of this ID using the `principal()` function inside an AWS IoT rule.

The following JSON file is an example of a complete JITP template. The value of the `templateBody` field must be a JSON object specified as an escaped string and can use only the values in the preceding list. You can use a variety of tools to create the required JSON output, such as `json.dumps` (Python) or `JSON.stringify` (Node). The value of the `roleARN` field must be the ARN of a role that has the `AWSIoTThingsRegistration` attached to it. Also, your template can use an existing `PolicyName` instead of the inline `PolicyDocument` in the example. (The first example adds line breaks for readability, but you can copy and paste the template that appears directly below it.)

```
        {
            "templateBody" : "{
                \r\n    \"Parameters\" : {\r\n
                            \"AWS::IoT::Certificate::CommonName\": {\r\n    \"Type\":
 \"String\"\r\n        },\r\n
                            \"AWS::IoT::Certificate::SerialNumber\": {\r\n  \"Type\":
 \"String\"\r\n        },\r\n
                            \"AWS::IoT::Certificate::Country\": {\r\n   \"Type\": \"String
\"\r\n        },\r\n
                            \"AWS::IoT::Certificate::Id\": {\r\n    \"Type\": \"String\"\r
\n        }\r\n    },\r\n
                        \"Resources\": {\r\n
                            \"thing\": {\r\n
                                \"Type\": \"AWS::IoT::Thing\",\r\n
                                \"Properties\": {\r\n
                                    \"ThingName\": {\r\n    \"Ref\":
 \"AWS::IoT::Certificate::CommonName\"\r\n  },\r\n
                                    \"AttributePayload\": {\r\n
                                        \"version\": \"v1\",\r\n
                                        \"serialNumber\": {\r\n
                                            \"Ref\": \"AWS::IoT::Certificate::SerialNumber
\"\r\n    }\r\n   },\r\n
                                    \"ThingTypeName\": \"lightBulb-versionA\",\r\n

                                    \"ThingGroups\": [\r\n
                                        \"v1-lightbulbs\",\r\n  {\r\n

                                            \"Ref\": \"AWS::IoT::Certificate::Country\"\r
\n }\r\n   ]\r\n   },\r\n
```

```
                            \"OverrideSettings\": {\r\n
                                \"AttributePayload\": \"MERGE\",\r\n
                                \"ThingTypeName\": \"REPLACE\",\r\n
                                \"ThingGroups\": \"DO_NOTHING\"\r\n }\r\n   },\r\n
                        \"certificate\": {\r\n
                            \"Type\": \"AWS::IoT::Certificate\",\r\n
                            \"Properties\": {\r\n
                                \"CertificateId\": {\r\n    \"Ref\":
 \"AWS::IoT::Certificate::Id\"\r\n  },\r\n
                                \"Status\": \"ACTIVE\"\r\n  },\r\n
                            \"OverrideSettings\": {\r\n
                                \"Status\": \"DO_NOTHING\"\r\n  }\r\n   },\r\n
                        \"policy\": {\r\n
                            \"Type\": \"AWS::IoT::Policy\",\r\n
                            \"Properties\": {\r\n
                                \"PolicyDocument\": \"{
                                    \\\"Version\\\": \\\"2012-10-17\\\",
                                    \\\"Statement\\\": [{
                                    \\\"Effect\\\": \\\"Allow\\\",
                                    \\\"Action\\\":[\\\"iot:Publish\\\"],
                                    \\\"Resource\\\": [\\\"arn:aws:iot:us-
east-1:123456789012:topic\/sample\/topic\\\"] }] }\"\r\n   }\r\n   }\r\n
                              }\r\n}",
            "roleArn" : "arn:aws:iam::123456789012:role/Provisioning-JITP"
        }
```

Here is a version you can copy and paste:

```
    {
      "templateBody" : "{\r\n    \"Parameters\" : {\r\n
 \"AWS::IoT::Certificate::CommonName\": {\r\n            \"Type\": \"String\"\r\n        },
\r\n        \"AWS::IoT::Certificate::SerialNumber\": {\r\n            \"Type\": \"String
\"\r\n        },\r\n        \"AWS::IoT::Certificate::Country\": {\r\n            \"Type\":
 \"String\"\r\n        },\r\n        \"AWS::IoT::Certificate::Id\": {\r\n            \"Type
\": \"String\"\r\n        }\r\n    },\r\n    \"Resources\": {\r\n        \"thing\": {\r\n
            \"Type\": \"AWS::IoT::Thing\",\r\n            \"Properties\": {\r\n
      \"ThingName\": {\r\n                    \"Ref\": \"AWS::IoT::Certificate::CommonName
\"\r\n                },\r\n                \"AttributePayload\": {\r\n
   \"version\": \"v1\",\r\n                    \"serialNumber\": {\r\n
      \"Ref\": \"AWS::IoT::Certificate::SerialNumber\"\r\n                    }\r\n
           },\r\n                \"ThingTypeName\": \"lightBulb-versionA\",\r\n
      \"ThingGroups\": [\r\n                    \"v1-lightbulbs\",\r\n
 {\r\n                        \"Ref\": \"AWS::IoT::Certificate::Country\"\r\n
        }\r\n                ]\r\n            },\r\n            \"OverrideSettings\": {\r
\n                \"AttributePayload\": \"MERGE\",\r\n                \"ThingTypeName\":
 \"REPLACE\",\r\n                \"ThingGroups\": \"DO_NOTHING\"\r\n            }\r\n
   },\r\n        \"certificate\": {\r\n            \"Type\": \"AWS::IoT::Certificate\",\r\n
            \"Properties\": {\r\n                \"CertificateId\": {\r\n
   \"Ref\": \"AWS::IoT::Certificate::Id\"\r\n                },\r\n                \"Status
\": \"ACTIVE\"\r\n            },\r\n            \"OverrideSettings\": {\r\n
  \"Status\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"policy\": {\r\n
           \"Type\": \"AWS::IoT::Policy\",\r\n            \"Properties\": {\r\n
      \"PolicyDocument\": \"{ \\\"Version\\\": \\\"2012-10-17\\\", \\\"Statement\\\": [{ \
\\"Effect\\\": \\\"Allow\\\", \\\"Action\\\":[\\\"iot:Publish\\\"], \\\"Resource\\\": [\\
\"arn:aws:iot:us-east-1:123456789012:topic\/foo\/bar\\\"] }] }\"\r\n            }\r\n
  }\r\n    }\r\n}",
      "roleArn" : "arn:aws:iam::123456789012:role/JITPRole"
    }
```

This sample template declares values for the `AWS::IoT::Certificate::CommonName`, `AWS::IoT::Certificate::SerialNumber`, `AWS::IoT::Certificate::Country`, and

AWS::IoT::Certificate::Id provisioning parameters that are extracted from the certificate and used in the Resources section. The JITP workflow then uses this template to perform the following actions:

- Register a certificate and set its status to PENDING_ACTIVE.
- Create one thing resource.
- Create one policy resource.
- Attach the policy to the certificate.
- Attach the certificate to the thing.
- Update the certificate status to ACTIVE.

You can also use CloudTrail to troubleshoot issues with your JITP template. For information about the metrics that are logged in Amazon CloudWatch, see Device provisioning metrics (p. 329).

Bulk registration

You can use the `start-thing-registration-task` command to register things in bulk. This command takes a registration template, an S3 bucket name, a key name, and a role ARN that allows access to the file in the S3 bucket. The file in the S3 bucket contains the values used to replace the parameters in the template. The file must be a newline-delimited JSON file. Each line contains all of the parameter values for registering a single device. For example:

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

The following bulk registration-related APIs might be useful:

- ListThingRegistrationTasks: Lists the current bulk thing provisioning tasks.
- DescribeThingRegistrationTask: Provides information about a specific bulk thing registration task.
- StopThingRegistrationTask: Stops a bulk thing registration task.
- ListThingRegistrationTaskReports: Used to check the results and failures for a bulk thing registration task.

Note

- Only one bulk registration operation task can run at a time (per account).
- Bulk registration operations call other AWS IoT control plane APIs. These calls might exceed the AWS IoT Throttling Quotas in your account and cause throttle errors. Contact AWS Customer Support to raise your AWS IoT throttling quotas, if necessary.

Provisioning templates

A provisioning template is a JSON document that uses parameters to describe the resources your device must use to interact with AWS IoT. A template contains two sections: Parameters and Resources. There are two types of provisioning templates in AWS IoT. One is used for just-in-time provisioning (JITP) and bulk registration and the second is used for fleet provisioning.

Parameters section

The Parameters section declares the parameters used in the Resources section. Each parameter declares a name, a type, and an optional default value. The default value is used when the dictionary passed in with the template does not contain a value for the parameter. The `Parameters` section of a template document looks like the following:

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

This template snippet declares four parameters: `ThingName`, `SerialNumber`, `Location`, and `CSR`. All of these parameters are of type `String`. The `Location` parameter declares a default value of `"WA"`.

Resources section

The `Resources` section of the template declares the resources required for your device to communicate with AWS IoT: a thing, a certificate, and one or more IoT policies. Each resource specifies a logical name, a type, and a set of properties.

A logical name allows you to refer to a resource elsewhere in the template.

The type specifies the kind of resource you are declaring. Valid types are:

- `AWS::IoT::Thing`
- `AWS::IoT::Certificate`
- `AWS::IoT::Policy`

The properties you specify depend on the type of resource you are declaring.

Thing resources

Thing resources are declared using the following properties:

- `ThingName`: String.
- `AttributePayload`: Optional. A list of name-value pairs.
- `ThingTypeName`: Optional. String for an associated thing type for the thing.
- `ThingGroups`: Optional. A list of groups to which the thing belongs.

Certificate resources

You can specify certificates in one of the following ways:

- A certificate signing request (CSR).

- A certificate ID of an existing device certificate. (Only certificate IDs can be used with a fleet provisioning template.)
- A device certificate created with a CA certificate registered with AWS IoT. If you have more than one CA certificate registered with the same subject field, you must also pass in the CA certificate used to sign the device certificate.

> **Note**
> When you declare a certificate in a template, use only one of these methods. For example, if you use a CSR, you cannot also specify a certificate ID or a device certificate. For more information, see X.509 client certificates (p. 203).

For more information, see X.509 Certificate overview (p. 200).

Certificate resources are declared using the following properties:

- `CertificateSigningRequest`: String.
- `CertificateID`: String.
- `CertificatePem`: String.
- `CACertificatePem`: String.
- `Status`: Optional. String that can be `ACTIVE` or `INACTIVE`. Defaults to ACTIVE.

Examples:

- Certificate specified with a CSR:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- Certificate specified with an existing certificate ID:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- Certificate specified with an existing certificate .pem and CA certificate .pem:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate"
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

Policy resources

Policy resources are declared using one of the following properties:

- `PolicyName`: Optional. String. Defaults to a hash of the policy document. If you are using an existing AWS IoT policy, for the `PolicyName` property, enter the name of the policy. Do not include the `PolicyDocument` property.
- `PolicyDocument`: Optional. A JSON object specified as an escaped string. If `PolicyDocument` is not provided, the policy must already be created.

> **Note**
> If a `Policy` section is present, `PolicyName` or `PolicyDocument`, but not both, must be specified.

Override settings

If a template specifies a resource that already exists, the `OverrideSettings` section allows you to specify the action to take:

`DO_NOTHING`

    Leave the resource as is.

`REPLACE`

    Replace the resource with the resource specified in the template.

`FAIL`

    Cause the request to fail with a `ResourceConflictsException`.

`MERGE`

    Valid only for the `ThingGroups` and `AttributePayload` properties of a `thing`. Merge the existing attributes or group memberships of the thing with those specified in the template.

When you declare a thing resource, you can specify `OverrideSettings` for the following properties:

- `ATTRIBUTE_PAYLOAD`
- `THING_TYPE_NAME`
- `THING_GROUPS`

When you declare a certificate resource, you can specify `OverrideSettings` for the `Status` property.

`OverrideSettings` are not available for policy resources.

Resource example

The following template snippet declares a thing, a certificate, and a policy:

```
{
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\", \"Statement\":
 [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\": [\"arn:aws:iot:us-
east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

The thing is declared with:

- The logical name `"thing"`.
- The type `AWS::IoT::Thing`.
- A set of thing properties.

  The thing properties include the thing name, a set of attributes, an optional thing type name, and an optional list of thing groups to which the thing belongs.

Parameters are referenced by `{"Ref":"parameter-name"}`. When the template is evaluated, the parameters are replaced with the parameter's value from the dictionary passed in with the template.

The certificate is declared with:

- The logical name `"certificate"`.
- The type `AWS::IoT::Certificate`.
- A set of properties.

  The properties include the CSR for the certificate, and setting the status to `ACTIVE`. The CSR text is passed as a parameter in the dictionary passed with the template.

The policy is declared with:

- The logical name `"policy"`.
- The type `AWS::IoT::Policy`.
- Either the name of an existing policy or a policy document.

Template example for JITP and bulk registration

The following JSON file is an example of a complete provisioning template that specifies the certificate with a CSR:

(The `PolicyDocument` field value must be a JSON object specified as an escaped string.)

```
{
    "Parameters" : {
        "ThingName " : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\", \"Statement\":
 [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\": [\"arn:aws:iot:us-
east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

The following JSON file is an example of a complete provisioning template that specifies an existing certificate with a certificate ID:

```
{
    "Parameters" : {
        "ThingName " : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CertificateId" : {
            "Type" : "String"
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref" : "CertificateId"}
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\", \"Statement\":
 [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\": [\"arn:aws:iot:us-
east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

Fleet provisioning

Fleet provisioning templates are used by AWS IoT to set up cloud and device configuration. These templates use the same parameters and resources as the JITP and bulk registration templates. For more information, see Provisioning templates (p. 669). Fleet provisioning templates can contain a `Mapping` section and a `DeviceConfiguration` section. You can use intrinsic functions inside a fleet provisioning template to generate device specific configuration. Fleet provisioning templates are named resources and are identified by ARNs (for example, `arn:aws:iot:us-west-2:1234568788:provisioningtemplate/templateName`).

Mappings

The optional `Mappings` section matches a key to a corresponding set of named values. For example, if you want to set values based on an AWS Region, you can create a mapping that uses the AWS Region name as a key and contains the values you want to specify for each specific Region. You use the `Fn::FindInMap` intrinsic function to retrieve values in a map.

You cannot include parameters, pseudo parameters, or call intrinsic functions in the `Mappings` section.

Device configuration

The device configuration section contains arbitrary data you want to send to your devices when provisioning. For example:

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
}
```

If you're sending messages to your devices by using the JavaScript Object Notation (JSON) payload format, AWS IoT Core formats this data as JSON. If you're using the Concise Binary Object Representation (CBOR) payload format, AWS IoT Core formats this data as CBOR. The `DeviceConfiguration` section doesn't support nested JSON objects.

Intrinsic functions

Intrinsic functions are used in any section of the provisioning template except the `Mappings` section.

`Fn::Join`

Appends a set of values into a single value, separated by the specified delimiter. If a delimiter is the empty string, the set of values are concatenated with no delimiter.

`Fn::Select`

Returns a single object from a list of objects by index.

Important
`Fn::Select` does not check for `null` values or if the index is out of bounds of the array. Both conditions result in a provisioning error, so you should ensure you chose a valid index value, and that the list contains non-null values.

`Fn::FindInMap`

Returns the value corresponding to keys in a two-level map that is declared in the `Mappings` section.

`Fn::Split`

Splits a string into a list of string values so you can select an element from the list of strings. You specify a delimiter that determine where the string is split (for example, a comma). After you split a string, use `Fn::Select` to select an element.

For example, if a comma-delimited string of subnet IDs is imported to your stack template, you can split the string at each comma. From the list of subnet IDs, use `Fn::Select` to specify a subnet ID for a resource.

`Fn::Sub`

Substitutes variables in an input string with values that you specify. You can use this function to construct commands or outputs that include values that aren't available until you create or update a stack.

Fleet provisioning template example

```
{
    "Parameters" : {
        "ThingName " : {
            "Type" : "String"
        },
        "SerialNumber": {
            "Type": "String"
        },
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Ref" : "ThingName"},
                "ThingTypeName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["v1-lightbulbs", "WA"],
                "BillingGroup": "LightBulbBillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/foo/bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"}, "LocationUrl"]}
        }
}
```

Note
An existing provisioning template can be updated to add a pre-provisioning hook (p. 677).

Pre-provisioning hooks

When using AWS IoT fleet provisioning, you can set up a Lambda function to validate parameters passed from the device before allowing the device to be provisioned. This Lambda function must exist in your account before you provision a device because it's called every time a device sends a request through the section called “RegisterThing” (p. 682). For devices to be provisioned, your Lambda function must accept the input object and return the output object described in this section. The provisioning proceeds only if the Lambda function returns an object with `"allowProvisioning": True`.

Important
AWS recommends using pre-provisioning hooks when creating provisioning templates to allow more control of which and how many devices your account onboards.

Pre-provision hook input

AWS IoT sends this object to the Lambda function when a device registers with AWS IoT.

```
{
    "claimCertificateId" : "string",
    "certificateId" : "string",
    "certificatePem" : "string",
    "templateArn" : "arn:aws:iot:us-east-1:1234567890:provisioningtemplate/MyTemplate",
    "clientId" : "221a6d10-9c7f-42f1-9153-e52e6fc869c1",
    "parameters" : {
        "key" : "value",
        ...
    }
}
```

The `parameters` object passed to the Lambda function contains the properties in the `parameters` argument passed in the the section called "RegisterThing" (p. 682) request payload.

Pre-provision hook return value

The Lambda function must return a response that indicates whether it has authorized the provisioning request and the values of any properties to override.

The following is an example of a successful response from the pre-provisioning function.

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
        "Key": "newCustomValue",
        ...
    }
}
```

`"parameterOverrides"` values will be added to `"parameters"` parameter of the the section called "RegisterThing" (p. 682) request payload.

Note

- If the Lambda function fails or doesn't return the `"allowProvisioning"` parameter in the response, the provisioning request will fail and the error will be returned in the response.
- The Lambda function must finish running and return within 5 seconds, otherwise the provisioning request fails.

Device provisioning MQTT API

The Fleet Provisioning service supports these MQTT APIs:

- the section called "CreateCertificateFromCsr" (p. 679)
- the section called "CreateKeysAndCertificate" (p. 680)
- the section called "RegisterThing" (p. 682)

This API supports response buffers in Concise Binary Object Representation (CBOR) format and JavaScript Object Notation (JSON), depending on the *payload-format* of the topic. For the sake of clarity, however, the response and request examples in this section are shown in JSON format.

payload-format	**Response format data type**
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Object Notation (JSON)

Important
Before publishing a request message topic, subscribe to the response topics to receive the response. The messages used by this API use MQTT's publish/subscribe protocol to provide a request and response interaction.
If you do not subscribe to the response topics *before* you publish a request, you might not receive the results of that request.

CreateCertificateFromCsr

Creates a certificate from a certificate signing request (CSR). AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA). The new certificate has a `PENDING_ACTIVATION` status. When you call `RegisterThing` to provision a thing with this certificate, the certificate status changes to `ACTIVE` or `INACTIVE` as described in the template.

CreateCertificateFromCsr request

Publish a message with the `$aws/certificates/create-from-csr/`*`payload-format`* topic.

`payload-format`

The message payload format as `cbor` or `json`.

CreateCertificateFromCsr request payload

```
{
    "certificateSigningRequest": "string"
}
```

`certificateSigningRequest`

The CSR, in PEM format.

CreateCertificateFromCsr response

Subscribe to `$aws/certificates/create-from-csr/`*`payload-format`*`/accepted`.

`payload-format`

The message payload format as `cbor` or `json`.

CreateCertificateFromCsr response payload

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

`certificateOwnershipToken`

The token to prove ownership of the certificate during provisioning.

`certificateId`

The ID of the certificate. Certificate management operations only take a certificateId.

`certificatePem`

The certificate data, in PEM format.

CreateCertificateFromCsr error

To receive error responses, subscribe to `$aws/certificates/create-from-csr/`*`payload-format`*`/rejected`.

`payload-format`

The message payload format as `cbor` or `json`.

CreateCertificateFromCsr error payload

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

The status code.

`errorCode`

The error code.

`errorMessage`

The error message.

CreateKeysAndCertificate

Creates new keys and a certificate. AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA). The new certificate has a `PENDING_ACTIVATION` status. When you call `RegisterThing` to provision a thing with this certificate, the certificate status changes to `ACTIVE` or `INACTIVE` as described in the template.

CreateKeysAndCertificate request

Publish a message on `$aws/certificates/create/`*`payload-format`* with an empty message payload.

`payload-format`

The message payload format as `cbor` or `json`.

CreateKeysAndCertificate response

Subscribe to `$aws/certificates/create/`*`payload-format`*`/accepted`.

`payload-format`

The message payload format as `cbor` or `json`.

CreateKeysAndCertificate response

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

`certificateId`

The certificate ID.

`certificatePem`

The certificate data, in PEM format.

`privateKey`

The private key.

`certificateOwnershipToken`

The token to prove ownership of the certificate during provisioning.

CreateKeysAndCertificate error

To receive error responses, subscribe to `$aws/certificates/create/`*`payload-format`*`/rejected`.

`payload-format`

The message payload format as `cbor` or `json`.

CreateKeysAndCertificate error payload

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

The status code.

`errorCode`

The error code.

`errorMessage`

The error message.

RegisterThing

Provisions a thing using a pre-defined template.

RegisterThing request

Publish a message on `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*.

`payload-format`

The message payload format as `cbor` or `json`.

`templateName`

The provisioning template name.

RegisterThing request payload

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
        "string": "string",
        ...
    }
}
```

`certificateOwnershipToken`

The token to prove ownership of the certificate. The token is generated by AWS IoT when you create a certificate over MQTT.

`parameters`

Optional. Key-value pairs from the device that are used by the pre-provisioning hooks (p. 677) to evaluate the registration request.

RegisterThing response

Subscribe to `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/accepted`.

`payload-format`

The message payload format as `cbor` or `json`.

`templateName`

The provisioning template name.

RegisterThing response payload

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

`deviceConfiguration`

The device configuration defined in the template.

`thingName`

The name of the IoT thing created during provisioning.

RegisterThing error response

To receive error responses, subscribe to `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/rejected`.

`payload-format`

The message payload format as `cbor` or `json`.

`templateName`

The provisioning template name.

RegisterThing error response payload

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

The status code.

`errorCode`

The error code.

`errorMessage`

The error message.

Fleet indexing service

Fleet Indexing is a managed service that you can use to index, search, and aggregate your registry data, shadow data, and device connectivity data (device lifecycle events) in the cloud. After you set up your fleet index, the service manages the indexing of updates for your thing groups, thing registries, and device shadows. For more information about aggregation queries, see Querying for aggregate data (p. 695). You can use a simple query language to search across this data. You can also create a dynamic thing group (p. 188) with a search query.

> **Note**
> It might take about 30 seconds for the Fleet Indexing service to update the fleet index after a thing is created, updated, or deleted.

When you enable indexing, AWS IoT creates an index for your things or thing groups. After it's active, you can run queries on your index, such as finding all devices that are handheld and have more than 70 percent battery life. AWS IoT keeps the index continuously updated with your latest data.

`AWS_Things` is the index created for all of your things. `AWS_ThingGroups` is the index that contains all of your thing groups.

You can use the AWS IoT console to manage your indexing configuration and run your search queries. Choose the indexes you would like to use in the console settings page. If you prefer programmatic access, you can use the AWS SDKs or the AWS Command Line Interface (AWS CLI).

For information about pricing this and other services, see the AWS IoT Device Management Pricing page.

Topics

Managing thing indexing

`AWS_Things` is the index created for all of your things. You can control what to index: registry data, shadow data, and device connectivity status data (driven by device lifecycle events).

Enabling thing indexing

You use the **update-indexing-configuration** CLI command or the UpdateIndexingConfiguration API to create the `AWS_Things` index and control its configuration. The `--thing-indexing-configuration` (`thingIndexingConfiguration`) parameter allows you to control what kind of data (for example, registry, shadow, and device connectivity data) is indexed.

The `--thing-indexing-configuration` parameter takes a string with the following structure:

```
{
  "thingIndexingMode": "OFF"|"REGISTRY"|"REGISTRY_AND_SHADOW",
  "thingConnectivityIndexingMode": "OFF"|"STATUS",
  "customFields": [
    { name: field-name, type: String | Number | Boolean },
    ...
  ]
}
```

The `thingIndexingMode` attribute controls what kind of data is indexed. Valid values are:

OFF

No indexing.

REGISTRY

Index registry data.

REGISTRY_AND_SHADOW

Index registry and thing shadow data.

The `thingConnectivityIndexingMode` attribute specifies if thing connectivity data is indexed. Valid values are:

OFF

Thing connectivity data is not indexed.

STATUS

Thing connectivity data is indexed.

The `customFields` attribute is a list of field and data type pairs. Aggregation queries can be performed over these fields based on the data type. The indexing mode you choose (REGISTRY or REGISTRY_AND_SHADOW) effects what fields can be specified in `customFields`. For example, if you specify the `REGISTRY` indexing mode, you cannot specify a field from a thing shadow. Custom fields must be specified in `customFields` to be indexed.

If there is a type inconsistency between a custom field in your configuration and the value being indexed, the Fleet Indexing service ignores the inconsistent value for aggregation queries. CloudWatch logs are helpful when troubleshooting aggregation query problems. For more information, see Troubleshooting aggregation queries for the fleet indexing service (p. 999).

Managed fields contain data associated with IoT things, thing groups, and device shadows. The data type of managed fields are defined by AWS IoT. You specify the values of each managed field when you create an IoT thing. For example thing names, thing groups, and thing descriptions are all managed fields. The Fleet Indexing service indexes managed fields based on the indexing mode you specify:

- Managed fields for the registry

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingType, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- Managed fields for thing shadows

```
"managedFields" : [
  {name:shadow.version, type:Number},
  {name:shadow.delta, type:Boolean}
]
```

- Managed fields for thing connectivity

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean}
]
```

- Managed fields for thing groups

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

Managed fields cannot be changed or appear in `customFields`.

The following is an example of how to use **update-indexing-configuration** to configure indexing:

aws iot update-indexing-configuration --thing-indexing-configuration 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number},{name=attributes.color, type=String},{name=shadow.desired.power, type=Boolean}}]

This command enables indexing for registry and shadow data. Aggregation queries work with the managed fields and the provided `customFields` based on the data type.

You can use the **get-indexing-configuration** CLI command or the GetIndexingConfiguration API to retrieve the current indexing configuration.

The following command shows how to use the **get-indexing-configuration** CLI command to retrieve the current thing indexing configuration where five custom fields (three registry custom fields and two shadow custom fields) are defined.

aws iot get-indexing-configuration

```
{
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    },
    "thingIndexingConfiguration": {
        "thingConnectivityIndexingMode": "STATUS",
        "customFields": [
            {
                "name": "attributes.customField_NUM",
                "type": "Number"
            },
            {
                "name": "shadow.desired.customField_STR",
                "type": "String"
            },
            {
                "name": "shadow.desired.customField_NUM",
                "type": "Number"
            },
            {
                "name": "attributes.customField_STR",
                "type": "String"
            },
            {
                "name": "attributes.customField_BOOL",
                "type": "Boolean"
            }
        ],
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "managedFields": [
            {
                "name": "shadow.delta",
                "type": "Boolean"
            },
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "connectivity.version",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "connectivity.connected",
                "type": "Boolean"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "thingId",
                "type": "String"
            },
            {
                "name": "connectivity.timestamp",
                "type": "Number"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "shadow.version",
                "type": "Number"
            }
        ]
    }
}
```

The following table provides the allowed combinations of `thingIndexingMode` and `thingConnectivityIndexingMode`, and their associated effects. The required `thingIndexingMode` parameter specifies if the `AWS_Things` index contains just registry data or registry and shadow data. The optional `thingConnectivityIndexingMode` parameter specifies whether the index also contains connectivity status data (when devices have last connected and disconnected to AWS IoT).

thingIndexingMode	**thingConnectivityIndexingM**	**Result**
OFF	*Not specified.*	No indexing or delete an index.
OFF	OFF	Equivalent to the previous entry.

thingIndexingMode	thingConnectivityIndexingM	Result
REGISTRY	*Not specified.*	Create or configure the `AWS_Things` index to index registry data only.
REGISTRY	OFF	Equivalent to the previous entry. (Only registry data is indexed.)
REGISTRY_AND_SHADOW	*Not specified.*	Create or configure the `AWS_Things` index to index registry data and shadow data.
REGISTRY_AND_SHADOW	OFF	Equivalent to the previous entry. (Registry data and shadow data are indexed.)
REGISTRY	STATUS	Create or configure the `AWS_Things` index to index registry data and thing connectivity status data (REGISTRY_AND_CONNECTIVITY_STATUS).
REGISTRY_AND_SHADOW	STATUS	Create or configure the `AWS_Things` index to index registry data, shadow data, and thing connectivity status data (REGISTRY_AND_SHADOW_AND_CONNECTIVIT

You can use the AWS IoT **update-indexing-configuration** CLI command to update your indexing configuration. The following examples show how to use the **update-indexing-configuration** CLI command.

Short syntax:

aws iot update-indexing-configuration --thing-indexing-configuration "thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS,customFields=[{name=a {name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean}]"

JSON syntax:

aws iot update-indexing-configuration --cli-input-json \ '{ "thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY_AND_SHADOW", "thingConnectivityIndexingMode": "STATUS", "customFields": [{ "name": "shadow.desired.power", "type": "Boolean" }, { "name": "attributes.color", "type": "String" }, { "name": "attributes.version", "type": "Number" }] } }'

The output of these commands is:

```
{
    "thingIndexingConfiguration": {
        "thingConnectivityIndexingMode": "STATUS",
        "customFields": [
            {
                "type": "String",
                "name": "attributes.color"
            },
            {
                "type": "Number",
                "name": "attributes.version"
            },
            {
                "type": "Boolean",
                "name": "shadow.desired.power"
            }
        ],
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "managedFields": [
            {
                "type": "Boolean",
                "name": "connectivity.connected"
            },
            {
                "type": "String",
                "name": "registry.thingTypeName"
            },
            {
                "type": "String",
                "name": "thingName"
            },
            {
                "type": "Number",
                "name": "shadow.version"
            },
            {
                "type": "String",
                "name": "thingId"
            },
            {
                "type": "Boolean",
                "name": "shadow.delta"
            },
            {
                "type": "Number",
                "name": "connectivity.timestamp"
            },
            {
                "type": "String",
                "name": "registry.thingGroupNames"
            },
            {
                "type": "Number",
                "name": "connectivity.version"
            },
            {
                "type": "Number",
                "name": "registry.version"
            }
        ]
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

In the following example, a new custom field is added to the configuration:

aws iot update-indexing-configuration --thing-indexing-configuration 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number}, {name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean}, {name=shadow.desired.intensity,type=Number}]'

This command added `shadow.desired.intensity` to the indexing configuration.

Note
Updating the custom fields indexing configuration overwrites all existing custom fields. Make sure to specify all custom fields when calling **update-indexing-configuration**.

After the index is rebuilt you can, use aggregation query on the newly added fields, search registry data, shadow data, and thing connectivity status data.

When changing the indexing mode, make sure all of your custom fields are valid using the new indexing mode. For example, if you start off with `REGISTRY_AND_SHADOW` mode with a custom field called `shadow.desired.temperature` you must delete the `shadow.desired.temperature` custom field before changing the indexing mode to `REGISTRY`. If your indexing configuration contains custom fields that are not indexed by the indexing mode, the update fails.

Describing a thing index

The following command shows you how to use the **describe-index** CLI command to retrieve the current status of the thing index.

```
aws iot describe-index --index-name "AWS_Things"
{
    "indexName": "AWS_Things",
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

The first time you enable indexing, AWS IoT builds your index. You can't query the index if `indexStatus` is in the `BUILDING` state. The `schema` for the things index indicates which type of data (`REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS`) is indexed.

Changing the configuration of your index causes the index to be rebuilt. During this process, the `indexStatus` is `REBUILDING`. You can execute queries on data in the things index while it is being rebuilt. For example, if you change the index configuration from `REGISTRY` to `REGISTRY_AND_SHADOW` while the index is being rebuilt, you can query registry data, including the latest updates. However, you can't query the shadow data until the rebuild is complete. The amount of time it takes to build or rebuild the index depends on the amount of data.

Querying a thing index

Use the **search-index** CLI command to query data in the index.

aws iot search-index --index-name "AWS_Things" --query-string "thingName:mything*"

```

{
    "things":[{
         "thingName":"mything1",
         "thingGroupNames":[
            "mygroup1"
         ],
         "thingId":"a4b9f759-b0f2-4857-8a4b-967745ed9f4e",
         "attributes":{
            "attribute1":"abc"
         },
         "connectivity": {
            "connected":false,
            "timestamp":1556649874716
         }
    },
    {
        "thingName":"mything2",
        "thingTypeName":"MyThingType",
        "thingGroupNames":[
            "mygroup1",
            "mygroup2"
        ],
        "thingId":"01014ef9-e97e-44c6-985a-d0b06924f2af",
        "attributes":{
            "model":"1.2",
            "country":"usa"
        },
        "shadow":{
            "desired":{
                "location":"new york",
                "myvalues":[3, 4, 5]
            },
            "reported":{
                "location":"new york",
                "myvalues":[1, 2, 3],
                "stats":{
                    "battery":78
                }
            },
            "metadata":{
                 "desired":{
                       "location":{
                            "timestamp":123456789
                        },
                       "myvalues":{
                             "timestamp":123456789
                       }
                  },
                  "reported":{
                        "location":{
                             "timestamp":34535454
                         },
                        "myvalues":{
                             "timestamp":34535454
                        },
                        "stats":{
                             "battery":{
                                   "timestamp":34535454
                             }
                        }
                 }
            },
            "version":10,
            "timestamp":34535454
        },
        "connectivity": {
            "connected":true,
            "timestamp":1556649855046
        }
    }],
    "nextToken":"AQFCuvk7zZ3D9pOYMbFCeHbdZ+h=G"
}
```

In the JSON response, `"connectivity"` (as enabled by the `thingConnectivityIndexingMode=STATUS` setting) provides a Boolean value and a timestamp that indicates if the device is connected to AWS IoT Core. The device `"mything1"` disconnected (`false`) at POSIX time `1556649874716`:

```
"connectivity": {
    "connected":false,
    "timestamp":1556649874716
}
```

The device "mything2" connected (true) at POSIX time 1556649855046:

```
"connectivity": {
    "connected":true,
    "timestamp":1556649855046
}
```

Timestamps are given in milliseconds since epoch, so 1556649855046 represents 6:44:15.046 PM on Tuesday, April 30, 2019 (GMT).

> **Important**
> If a device has been disconnected for approximately an hour, the "timestamp" value of the connectivity status might be missing.

Restrictions and limitations

These are the restrictions and limitations for AWS_Things.

Shadow fields with complex types

A shadow field is indexed only if the value of the field is a simple type, a JSON object that does not contain an array, or an array that consists entirely of simple types. Simple type means a string, number, or one of the literals true or false. For example, given the following shadow state, the value of field "palette" is not indexed because it's an array that contains items of complex types. The value of field "colors" is indexed because each value in the array is a string.

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
    }
}
```

Nested shadow field names

The names of nested shadow fields are stored as a period (.) delimited string. For example, given a shadow document:

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

the name of field three is stored as desired.one.two.three. If you also have a shadow document like this:

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

both match a query for shadow.desired.one.two.three:v2. As a best practice, do not use periods in shadow field names.

Shadow metadata

A field in a shadow's metadata section is indexed, but only if the corresponding field in the shadow's "state" section is indexed. (In the previous example, the "palette" field in the shadow's metadata section is not indexed either.)

Unregistered shadows

If you use UpdateThingShadow to create a shadow using a thing name that hasn't been registered in your AWS IoT account, fields in this shadow are not indexed.

Numeric values

If any registry or shadow data is recognized by the service as a numeric value, it's indexed as such. You can form queries involving ranges and comparison operators on numeric values (for example, "attribute.foo<5" or "shadow.reported.foo:[75 TO 80]"). To be recognized as numeric, the value of the data must be a valid JSON number type literal (an integer in the range -2^53...2^53-1 or a double-precision floating point with optional exponential notation) or part of an array that contains only such values.

Null values

Null values are not indexed.

Maximum number of custom fields for aggregation queries

5

Maximum number of requested percentiles for aggregation queries.

100

Authorization

You can specify the things index as a resource ARN in an AWS IoT policy action, as follows.

Action	Resource
iot:SearchIndex	An index ARN (for example, arn:aws:iot:*your-aws-regionyour-aws-account*:index/AWS_Things).

Action	Resource
`iot:DescribeIndex`	An index ARN (for example, `arn:aws:iot:your-aws-region:index/AWS_Things`).

Note
If you have permissions to query the fleet index, you can access the data of things across the entire fleet.

Managing thing group indexing

`AWS_ThingGroups` is the index that contains all of your thing groups. You can use this index to search for groups based on group name, description, attributes, and all parent group names.

Enabling thing group indexing

You can use the `thing-group-indexing-configuration` setting in the UpdateIndexingConfiguration API to create the `AWS_ThingGroups` index and control its configuration. You can use the GetIndexingConfiguration API to retrieve the current indexing configuration.

Use the **get-indexing-configuration** CLI command to retrieve the current thing and thing group indexing configurations.

aws iot get-indexing-configuration

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

Use the **update-indexing-configuration** CLI command to update the thing group indexing configurations.

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

Note
You can also update configurations for both thing and thing group indexing in a single command, as follows.

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

The following are valid values for `thingGroupIndexingMode`.

OFF

No indexing/delete index.

ON

Create or configure the `AWS_ThingGroups` index.

Describing group indexes

Use the **describe-index** CLI command to retrieve the current status of the `AWS_ThingGroups` index.

```
aws iot describe-index --index-name "AWS_ThingGroups"
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT builds your index the first time you enable indexing. You can't query the index if the `indexStatus` is `BUILDING`.

Querying a thing group index

Use the **search-index** CLI command to query data in the index:

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupNames:mythinggroup*"
```

Authorization

You can specify the thing groups index as a resource ARN in an AWS IoT policy action, as follows.

Action	Resource
`iot:SearchIndex`	An index ARN (for example, `arn:aws:iot:`*`your-aws-region`*`:index/AWS_ThingGroups`).
`iot:DescribeIndex`	An index ARN (for example, `arn:aws:iot:`*`your-aws-region`*`:index/AWS_ThingGroups`).

Querying for aggregate data

AWS IoT provides three APIs (`GetStatistics`, `GetCardinality`, and `GetPercentiles`) that allow you to search your device fleet for aggregate data.

GetStatistics

The GetStatistics API and the **get-statistics** CLI command return the count, average, sum, minimum, maximum, sum of squares, variance, and standard deviation for the specified aggregated field.

The **get-statistics** CLI command takes the following parameters:

`index-name`

The name of the index to search. The default value is `AWS_Things`.

`query-string`

The query used to search the index. You can specify `"*"` to get the count of all indexed things in your AWS account.

`aggregationField`

Optional. The field to aggregate. This field must be a managed or custom field defined when you call **update-indexing-configuration**. If you don't specify an aggregation field, `registry.version` is used as aggregation field.

`query-version`

The version of the query to use. The default value is `2017-09-30`.

The type of aggregation field can affect the statistics returned.

GetStatistics with string values

If you aggregate on a string field, calling `GetStatistics` returns a count of devices that have attributes that match the query. For example:

aws iot get-statistics --aggregation-field 'attributes.stringAttribute' --query-string '*'

This command returns the number of devices that contain an attribute named `stringAttribute`:

```
{
  "statistics": {
    "count": 3
  }
}
```

GetStatistics with Boolean values

When you call `GetStatistics` with a boolean aggregation field:

- AVERAGE is the percentage of devices that match the query.
- MINIMUM is 0 or 1 according to the following rules:
  - If all the values for the aggregation field are `false`, MINIMUM is 0.
  - If all the values for the aggregation field are `true`, MINIMUM is 1.
  - If the values for the aggregation field are a mixture of `false` and `true`, MINIMUM is 0.
- MAXIMUM is 0 or 1 according to the following rules:
  - If all the values for the aggregation field are `false`, MAXIMUM is 0.
  - If all the values for the aggregation field are `true`, MAXIMUM is 1.
  - If the values for the aggregation field are a mixture of `false` and `true`, MAXIMUM is 1.
- SUM is the sum of the integer equivalent of the boolean values.
- COUNT is the number of things that match the query.

GetStatistics with numerical values

When you call `GetStatistics` and specify an aggregation field of type `Number`, `GetStatistics` returns the following values:

count

The number of devices that have a field that matches the query.

average

The average of the numerical values that match the query.

sum

The sum of the numerical values that match the query.

minimum

The smallest of the numerical values that match the query.

maximum

The largest of the numerical values that match the query.

sumOfSquares

The sum of the squares of the numerical values that match the query.

variance

The variance of the numerical values that match the query. The variance of a set of values is the average of the squares of the differences of each value from the average value of the set.

stdDeviation

The standard deviation of the numerical values that match the query. The standard deviation of a set of values is a measure of how spread out the values are.

The following example shows how to call **get-statistics** with a numerical custom field.

aws iot get-statistics --aggregation-field 'attributes.numericAttribute2' --query-string '*'

```
{
  "statistics": {
    "count": 3,
    "average": 33.333333333333336,
    "sum": 100.0,
    "minimum": -125.0,
    "maximum": 150.0,
    "sumOfSquares": 43750.0,
    "variance": 13472.22222222222,
    "stdDeviation": 116.06990230986766
  }
}
```

For numerical aggregation fields, if the field values exceed the maximum double value, the statistics values are empty

GetCardinality

The GetCardinality API and the **get-cardinality** CLI command return the approximate count of unique values that match the query. For example, you might want to find the number of devices with battery levels at less than 50 percent:

aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel > 50" --aggregation-field "shadow.reported.batterylevel".

This command returns the number of things with battery levels at more than 50 percent:

```
{
    "cardinality": 100
}
```

`cardinality` is always returned by **get-cardinality** even if there are no matching fields. For example:

aws iot get-cardinality --query-string "thingName:Non-existent*" --aggregation-field "attributes.customField_STR"

```
{
    "cardinality": 0
}
```

The **get-cardinality** CLI command takes the following parameters:

`index-name`

The name of the index to search. The default value is `AWS_Things`.

`query-string`

The query used to search the index. You can specify `"*"` to get the count of all indexed things in your AWS account.

`aggregationField`

The field to aggregate.

`query-version`

The version of the query to use. The default value is `2017-09-30`.

GetPercentiles

The GetPercentiles API and the **get-percentiles** CLI command groups the aggregated values that match the query into percentile groupings. The default percentile groupings are: 1,5,25,50,75,95,99, although you can specify your own when you call `GetPercentiles`. This function returns a value for each percentile group specified (or the default percentile groupings). The percentile group "1" contains the aggregated field value that occurs in approximately one percent of the values that match the query. The percentile group "5" contains the aggregated field value that occurs in approximately five percent of the values that match the query, and so on. The result is an approximation, the more values that match the query, the more accurate the percentile values.

The following example shows how to call the **get-percentiles** CLI command.

aws iot get-percentiles --query-string "thingName:*" --aggregation-field "attributes.customField_NUM" --percents 10 20 30 40 50 60 70 80 90 99

```
{
    "percentiles": [
        {
            "value": 3.0,
            "percent": 80.0
        },
        {
            "value": 2.5999999999999996,
            "percent": 70.0
        },
        {
            "value": 3.0,
            "percent": 90.0
        },
        {
            "value": 2.0,
            "percent": 50.0
        },
        {
            "value": 2.0,
            "percent": 60.0
        },
        {
            "value": 1.0,
            "percent": 10.0
        },
        {
            "value": 2.0,
            "percent": 40.0
        },
        {
            "value": 1.0,
            "percent": 20.0
        },
        {
            "value": 1.4,
            "percent": 30.0
        },
        {
            "value": 3.0,
            "percent": 99.0
        }
    ]
}
```

The following command shows the output returned from **get-percentiles** when there are no matching documents.

aws iot get-percentiles --query-string "thingName:Non-existent*" --aggregation-field "attributes.customField_NUM"

```
{
    "percentiles": []
}
```

The **get-percentile** CLI command takes the following parameters:

`index-name`

The name of the index to search. The default value is `AWS_Things`.

`query-string`

The query used to search the index. You can specify `"*"` to get the count of all indexed things in your AWS account.

`aggregationField`

The field to aggregate, which must be of `Number` type.

`query-version`

The version of the query to use. The default value is `2017-09-30`.

`percents`

Optional. You can use this parameter to specify custom percentile groupings.

Authorization

You can specify the thing groups index as a resource ARN in an AWS IoT policy action, as follows.

Action	Resource
`iot:GetStatistics`	An index ARN (for example, `arn:aws:iot:your-aws-region:index/AWS_Things` or `arn:aws:iot:your-aws-region:index/AWS_ThingGroups`).

Query syntax

Queries are specified using a query syntax.

The query syntax supports the following features.

- Terms and phrases
- Searching fields
- Prefix search
- Range search
- Boolean operators `AND`, `OR`, `NOT` and `–`. The hyphen is used to exclude something from search results (for example, `thingName:(tv* AND -plasma)`).
- Grouping
- Field grouping
- Escaping special characters (as with \)

The query syntax does not support the following features:

- Leading wildcard search (such as "*xyz"), but searching for "*" matches all things
- Regular expressions
- Boosting
- Ranking
- Fuzzy searches
- Proximity search
- Sorting
- Aggregation

A few things to note about the query language:

- The default operator is AND. A query for `"thingName:abc thingType:xyz"` is equivalent to `"thingName:abc AND thingType:xyz"`.
- If a field isn't specified, AWS IoT searches for the term in all fields.
- All field names are case sensitive.
- Search is case insensitive. Words are separated by white space characters as defined by Java's `Character.isWhitespace(int)`.
- Indexing of device shadow data includes reported, desired, delta, and metadata sections.
- Device shadow and registry versions are not searchable, but are present in the response.
- The maximum number of terms in a query is 5.

Example thing queries

Queries are specified in a query string using a query syntax and passed to the `SearchIndex` API. The following table lists some example query strings.

Query string	Result
abc	Queries for "abc" in any registry or shadow field.
thingName:myThingName	Queries for a thing with name "myThingName".
thingName:my*	Queries for things with names that begin with "my".
thingName:ab?	Queries for things with names that have "ab" plus one additional character (for example: "aba", "abb", "abc", and so on.)
thingTypeName:aa	Queries for things that are associated with type aa.
attributes.myAttribute:75	Queries for things with an attribute named "myAttribute" that has the value 75.
attributes.myAttribute:[75 TO 80]	Queries for things with an attribute named "myAttribute" whose value falls within a numeric range (75–80, inclusive).
attributes.myAttribute:{75 TO 80]	Queries for things with an attribute named "myAttribute" whose value falls within the numeric range (>75 and <=80).
attributes.serialNumber:["abcd" TO "abcf"]	Queries for things with an attribute named "serialNumber" whose value is within an alphanumeric string range. This query returns things with a "serialNumber" attribute with values "abcd", "abce", or "abcf".
attributes.myAttribute:i*t	Queries for things with an attribute named "myAttribute" whose value is 'i', followed by any number of characters, followed by 't'.
attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10	Queries for things that combine terms using Boolean expressions. This query returns things that have an attribute named "attr1" with a value "abc", an attribute named "attr2" that is less than 5, and an attribute named "attr3" that is not greater than 10.
shadow.hasDelta:true	Queries for things whose shadow has a delta element.
NOT attributes.model:legacy	Queries for things where the attribute named "model" is not "legacy".
shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy	Queries for things with the following: • The thing's shadow `stats.battery` attribute has a value between 70 and 100.

Query string	Result
	• The text "v2" or "v3" occurs in a thing's name, type name, or attribute values. • The thing's `model` attribute is not set to "legacy".
shadow.reported.myvalues:2	Queries for things where the `myvalues` array in the shadow's reported section contains a value of 2.
shadow.reported.location:* NOT shadow.desired.stats.battery:*	Queries for things with the following: • The `location` attribute exists in the shadow's `reported` section. • The `stats.battery` attribute does not exist in the shadow's `desired` section.
connectivity.connected:true	Queries for all connected devices.
connectivity.connected:false	Queries for all disconnected devices.
connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]	Queries for all connected devices with a connect timestamp >= 1557651600000 and <= 1557867600000. Timestamps are given in milliseconds since epoch.
connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]	Queries for all disconnected devices with a disconnect timestamp >= 1557651600000 and <= 1557867600000. Timestamps are given in milliseconds since epoch.
connectivity.connected:true AND connectivity.timestamp > 1557651600000	Queries for all connected devices with a connect timestamp > 1557651600000. Timestamps are given in milliseconds since epoch.
connectivity.connected:*	Queries for all devices with connectivity information present.

Example thing group queries

Queries are specified in a query string using a query syntax and passed to the `SearchIndex` API. The following table lists some example query strings.

Query string	Result
abc	Queries for "abc" in any field.
thingGroupNames:myGroupThingName	Queries for a thing group with name "myGroupThingName".
thingGroupNames:my*	Queries for thing groups with names that begin with "my".
thingGroupNames:ab?	Queries for thing groups with names that have "ab" plus one additional character (for example: "aba", "abb", "abc", and so on.)

Query string	Result
attributes.myAttribute:75	Queries for thing groups with an attribute named "myAttribute" that has the value 75.
attributes.myAttribute:[75 TO 80]	Queries for thing groups with an attribute named "myAttribute" whose value falls within a numeric range (75–80, inclusive).
attributes.myAttribute:[75 TO 80]	Queries for thing groups with an attribute named "myAttribute" whose value falls within the numeric range (>75 and <=80).
attributes.myAttribute:["abcd" TO "abcf"]	Queries for thing groups with an attribute named "myAttribute" whose value is within an alphanumeric string range. This query returns thing groups with a "serialNumber" attribute with values "abcd", "abce", or "abcf".
attributes.myAttribute:i*t	Queries for thing groups with an attribute named "myAttribute" whose value is 'i', followed by any number of characters, followed by 't'.
attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10	Queries for thing groups that combine terms using Boolean expressions. This query returns thing groups that have an attribute named "attr1" with a value "abc", an attribute named "attr2" that is less than 5, and an attribute named "attr3" that is not greater than 10.
NOT attributes.myAttribute:cde	Queries for thing groups where the attribute named "myAttribute" is not "cde".
parentGroupNames:(myParentThingGroupName)	Queries for thing groups whose parent group name matches "myParentThingGroupName".
parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)	Queries for thing groups whose parent group name matches "myParentThingGroupName" or "myRootThingGroupName".
parentGroupNames:(myParentThingGroupNa*)	Queries for thing groups whose parent group name begins with "myParentThingGroupNa".

AWS IoT Streaming service

You can use the AWS IoT Streaming service to manage files and transfer them to AWS IoT devices in your fleet. In the AWS Cloud, you can create a *stream* that contains multiple files, you can update stream data (the file list and descriptions), get the stream data, and more. The AWS IoT Streaming service can transfer data in small blocks to your IoT devices, using the MQTT protocol with support for request and response messages in JSON or CBOR.

Topics

What is a stream?

In AWS IoT, a *stream* is a publicly addressable resource that is an abstraction for a list of files that can be transferred to an IoT device. A typical stream contains the following information:

- **An Amazon Resource Name (ARN)** that uniquely identifies a stream at a given time. This ARN has the pattern `arn:`*`partition`*`:iot:`*`region`*`:`*`account-ID`*`:stream/`*`stream ID`*.
- **A stream ID** that identifies your stream and is used (and usually required) in AWS Command Line Interface (AWS CLI) or SDK commands.
- **A stream description** that provides a description of the stream resource.
- **A stream version** that identifies a particular version of the stream. Because stream data can be modified immediately before devices start the data transfer, the stream version can be used by the devices to enforce a consistency check.
- **A list of files** that can be transferred to devices. For each file in the list, the stream records a file ID, the file size, and the address information of the file which consists of, for example, the Amazon S3 bucket name, object key, and object version.
- **An AWS Identity and Access Management (IAM) role** that grants the AWS IoT Streaming service the permission to read stream files stored in data storage.

The AWS IoT Streaming service provides the following functionality so that devices can transfer data from the AWS Cloud:

- Data transfer using the MQTT protocol.
- Support for JSON or CBOR formats.
- The ability to describe a stream (`DescribeStream` API) to get a stream file list, stream version, and related information.
- The ability to send data in small blocks (`GetStream` API) so that devices with hardware constraints can receive the blocks.
- Support for a dynamic block size per request, to support devices that have different memory capacities.
- Optimization for concurrent streaming requests when multiple devices request data blocks from the same stream file.

- Amazon S3 as data storage for stream files.
- Support for data transfer log publishing from the AWS IoT Streaming service to CloudWatch.

For streaming service quotas, see AWS IoT Core Service Quotas in the *AWS General Reference*.

Managing a stream in the AWS Cloud

AWS IoT provides AWS SDK and AWS CLI commands that you can use to manage a stream in the AWS Cloud. You can use these commands to do the following:

- Create a stream. CLI / SDK
- Describe a stream to get its information. CLI / SDK
- List streams in your AWS account. CLI / SDK
- Update the file list or stream description in a stream. CLI / SDK
- Delete a stream. CLI / SDK

> **Note**
> At this time, streams are not visible in the AWS Management Console. You must use the AWS CLI or AWS SDK to manage a stream in AWS IoT.

Before you use the AWS IoT Streaming service from your devices, you must follow the steps in the next sections to make sure that your devices are properly authorized and can connect to the AWS IoT Device Gateway.

Grant permissions to your devices

You can follow the steps in Create an AWS IoT policy to create a device policy or use an existing device policy. Attach the policy to the certificates associated with your devices and add the following permissions to the device policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {   "Effect": "Allow",
            "Action": [ "iot:Connect" ],
            "Resource": [
                "arn:partition:iot:region:accountID:client/
${iot:Connection.Thing.ThingName}"
            ]
        }
        {
            "Effect": "Allow",
            "Action": [ "iot:Receive", "iot:Publish" ],
            "Resource": [
                "arn:partition:iot:region:accountID:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:accountID:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        }
    ]
}
```

Connect your devices to AWS IoT

Devices that use the AWS IoT Streaming service are required to connect with AWS IoT. The AWS IoT Streaming service integrates with AWS IoT in the AWS Cloud, so your devices should directly connect to the endpoint of the AWS IoT Data Plane.

> **Note**
> The endpoint of the AWS IoT data plane is specific to the AWS account and Region. You must use the endpoint for the AWS account and the Region in which your devices are registered in AWS IoT.

See Connecting to AWS IoT Core (p. 65) for more information.

Using the AWS IoT Streaming service in devices

To initiate the data transfer process, a device must receive an **initial data set**, which includes a stream ID at minimum. You can use an Jobs (p. 537) to schedule data transfer tasks for your devices by including the initial data set in the job document. When a device receives the initial data set, it should then start the interaction with the AWS IoT Streaming service. To exchange data with the AWS IoT Streaming service, a device should:

- Use the MQTT protocol to subscribe to the Streaming service topics (p. 98).
- Send requests and then wait to receive the responses using MQTT messages.

You can optionally include a stream file ID and a stream version in the initial data set. Sending a stream file ID to a device can simplify the programming of the device's firmware/software, because it eliminates the need to make a `DescribeStream` request from the device to get this ID. The device can specify the stream version in a `GetStream` request to enforce a consistency check in case the stream has been updated unexpectedly.

Use DescribeStream to get stream data

The AWS IoT Streaming service provides the `DescribeStream` API to send stream data to a device. The stream data returned by this API includes the stream ID, stream version, stream description and a list of stream files, each of which has a file ID and the file size in bytes. With this information, a device can select arbitrary files to initiate the data transfer process.

> **Note**
> You don't need to use the `DescribeStream` API if your device receives all required stream file IDs in the initial data set.

Follow these steps to make a `DescribeStream` request.

1. Subscribe to the "accepted" topic filter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/description/json`.
2. Subscribe to the "rejected" topic filter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/rejected/json`.
3. Publish a `DescribeStream` request by sending a message to `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/describe/json`.

4. If the request was accepted, your device receives a `DescribeStream` response on the "accepted" topic filter.
5. If the request was rejected, your device receives the error response on the "rejected" topic filter.

   Note
   If you replace `json` with `cbor` in the topics and topic filters shown, your device receives messages in the CBOR format, which is more compact than JSON.

DescribeStream request

A typical `DescribeStream` request in JSON looks like the following example.

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- (Optional) "`c`" is the client token field.

  The client token can't be longer than 64 bytes. A client token that is longer than 64 bytes causes an error response and an `InvalidRequest` error message.

DescribeStream response

A `DescribeStream` response in JSON looks like the following example.

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039",
    "s": 1,
    "d": "This is the description of stream ABC.",
    "r": [
        {
            "f": 0,
            "z": 131072
        },
        {
            "f": 1,
            "z": 51200
        }
    ]
}
```

- "`c`" is the client token field. This is returned if it was given in the `DescribeStream` request. Use the client token to associate the response with its request.
- "`s`" is the stream version as an integer. You can use this version to perform a consistency check with your `GetStream` requests.
- "`r`" contains a list of the files in the stream.
  - "`f`" is the stream file ID as an integer.
  - "`z`" is the stream file size in number of bytes.
- "`d`" contains the description of the stream.

Get data blocks from a stream file

You can use the `GetStream` API so that a device can receive stream files in small data blocks, so it can be used by those devices that have constraints on processing large block sizes. To receive an entire data file, a device might need to send or receive multiple requests and responses until all data blocks are received and processed.

GetStream request

Follow these steps to make a `GetStream` request.

1. Subscribe to the "accepted" topic filter `$aws/things/ThingName/streams/StreamId/data/json`.
2. Subscribe to the "rejected" topic filter `$aws/things/ThingName/streams/StreamId/rejected/json`.
3. Publish a `GetStream` request to the topic `$aws/things/ThingName/streams/StreamId/get/json`.
4. If the request was accepted, your device will receive one or more `GetStream` responses on the "accepted" topic filter. Each response message contains basic information and a data payload for a single block.
5. Repeat steps 3 and 4 to receive all data blocks. You must repeat these steps if the amount of data requested is larger than 128 KB. You must program your device to use multiple `GetStream` requests to receive all of the data requested.
6. If the request was rejected, your device will receive the error response on the "rejected" topic filter.

Note

- If you replace "json" with "cbor" in the topics and topic filters shown, your device will receive messages in the CBOR format, which is more compact than JSON.
- The AWS IoT Streaming service limits the size of a block to 128 KB. If you make a request for a block that is more than 128 KB, the request will fail.
- You can make a request for multiple blocks whose total size is greater than 128 KB (for example, if you make a request for 5 blocks of 32 KB each for a total of 160 KB of data). In this case, the request doesn't fail, but your device must make multiple requests in order to receive all of the data requested. The service will send additional blocks as your device makes additional requests. We recommend that you continue with a new request only after the previous response has been correctly received and processed.
- Regardless of the total size of data requested, you should program your device to initiate retries when blocks are not received, or not received correctly.

A typical `GetStream` request in JSON looks like the following example.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4098,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [optional] "`c`" is the client token field.

  The client token can be no longer than 64 bytes. A client token that is longer than 64 bytes causes an error response and an `InvalidRequest` error message.
- [optional] "`s`" is the stream version field (an integer).

The Streaming service applies a consistency check based on this requested version and the latest stream version in the cloud. If the stream version sent from a device in a `GetStream` request doesn't match the latest stream version in the cloud, the service sends an error response and a `VersionMismatch` error message. Typically, a device receives the expected (latest) stream version in the initial data set or in the response to `DescribeStream`.

- "`f`" is the stream file ID (an integer in the range 0 to 255).

  The stream file ID is required when you create or update a stream using the AWS CLI or SDK. If a device requests a stream file with an ID that doesn't exist, the service sends an error response and a `ResourceNotFound` error message.
- "`l`" is the data block size in bytes (an integer in the range 256 to 131,072).

  Refer to Build a bitmap for a GetStream request (p. 710) for instructions on how to use the bitmap fields to specify what portion of the stream file will be returned in the `GetStream` response. If a device specifies a block size that is out of range, the service sends an error response and a `BlockSizeOutOfBounds` error message.
- [optional] "`o`" is the offset of the block in the stream file (an integer in the range 0 to 98,304).

  Refer to Build a bitmap for a GetStream request (p. 710) for instructions on how to use the bitmap fields to specify what portion of the stream file will be returned in the `GetStream` response. The maximum value of 98,304 is based on a 24 MB stream file size limit and 256 bytes for the minimum block size. The default is 0 if not specified.
- [optional] "`n`" is the number of blocks requested (an integer in the range 0 to 98,304).

  The "n" field specifies either (1) the number of blocks requested, or (2) when the bitmap field ("b") is used, a limit on the number of blocks that will be returned by the bitmap request. This second use is optional. If not defined, it defaults to 131072/*`DataBlockSize`*.
- [optional] "`b`" is a bitmap that represents the blocks being requested.

  Using a bitmap, your device can request non-consecutive blocks, which makes handling retries following an error more convenient. Refer to Build a bitmap for a GetStream request (p. 710) for instructions on how to use the bitmap fields to specify which portion of the stream file will be returned in the `GetStream` response. For this field, convert the bitmap to a string representing the bitmap's value in hexadecimal notation. The bitmap must be less than 12,288 bytes.

GetStream response

A `GetStream` response in JSON looks like this example for each data block that is requested.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4098,
    "i": 2,
    "p": "..."
}
```

- "`c`" is the client token field. This is returned if it was given in the `GetStream` request. Use the client token to associate the response with its request.
- "`f`" is the ID of the stream file to which the current data block payload belongs.
- "`l`" is the size of the data block payload in bytes.
- "`i`" is the ID of the data block contained in the payload. Data blocks are numbered starting from 0.
- "`p`" contains the data block payload. This field is a string which represents the value of the data block in hexadecimal notation.

Build a bitmap for a GetStream request

You can use the bitmap field (`b`) in a `GetStream` request to get non-consecutive blocks from a stream file. This helps devices with limited RAM capacity deal with network delivery issues. A device can request only those blocks that were not received or not received correctly. The bitmap determines which blocks of the stream file will be returned. For each bit which is set to 1 in the bitmap, a corresponding block of the stream file will be returned.

Here's an example of how to specify a bitmap and its supporting fields in a `GetStream` request. For example, you want to receive a stream file in chunks of 256 bytes (the block size). Think of each block of 256 bytes as having a number that specifies its position in the file, starting from 0. So block 0 is the first block of 256 bytes in the file, block 1 is the second, and so on. You want to request blocks 20, 21, 24 and 43 from the file.

Block offset

Because the first block is number 20, specify the offset (field `o`) as 20 to save space in the bitmap.

Number of blocks

To ensure that your device doesn't receive more blocks than it can handle with limited memory resources, you can specify the maximum number of blocks that should be returned in each message sent by the streaming service. Note that this value is disregarded if the bitmap itself specifies less than this number of blocks, or if it would make the total size of the response messages sent by the streaming service greater than the service limit of 128 KB per `GetStream` request.

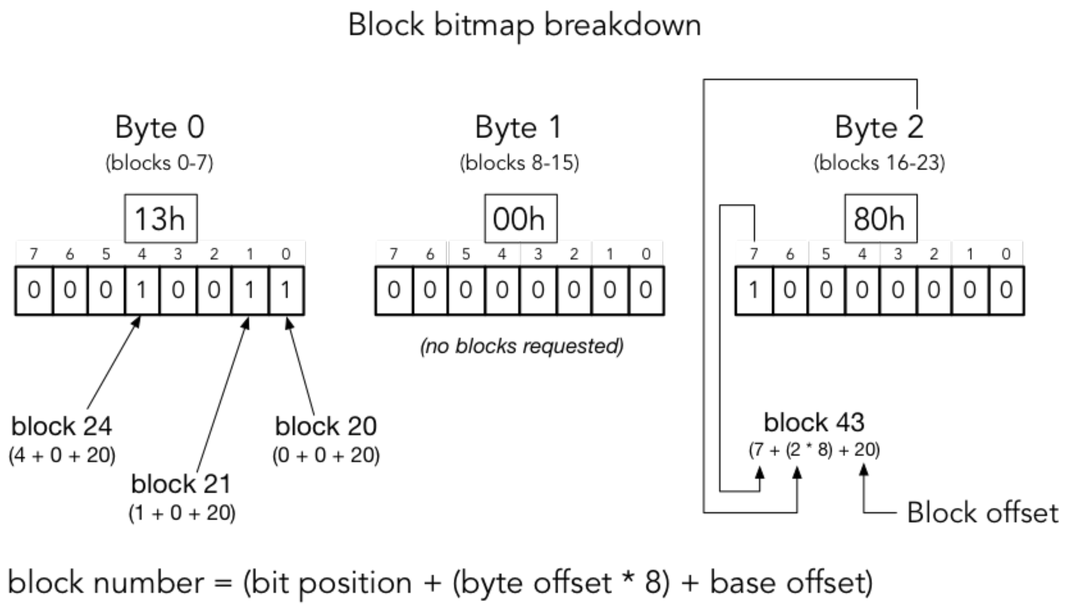
Block bitmap

The bitmap itself is an array of unsigned bytes expressed in hexadecimal notation, and included in the `GetStream` request as a string representation of the number. But to construct this string, let's start by thinking of the bitmap as a long sequence of bits (a binary number). If a bit in this sequence is set to 1, the corresponding block from the stream file will be sent back to the device. For our example, we want to receive blocks 20, 21, 24, and 43, so we must set bits 20, 21, 24, and 43 in our bitmap. We can use the block offset to save space, so after we subtract the offset from each block number, we want to set bits 0, 1, 4, and 23, like the following example.

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Taking one byte (8 bits) at a time, this is conventionally written as: "0b00010011", "0b00000000", and "0b10000000". Bit 0 shows up in our binary representation at the end of the first byte, and bit 23 at the beginning of the last. This can be confusing unless you know the conventions. The first byte contains bits 7-0 (in that order), the second byte contains bits 15-8, the third byte contains bits 23-16, and so on. In hexadecimal notation, this converts to "0x130080".

Tip
You can convert the standard binary to hexadecimal notation. Take four binary digits at a time and convert these to their hexadecimal equivalent. For example, "0001" becomes "1", "0011" becomes "3" and so on.



Putting this all together, the JSON for our `GetStream` request looks like the following.

```
{
    "c" : "1",        // client token
    "s" : 1,          // expected stream version
    "l" : 256,        // block size
    "f" : 1,          // source file index id
    "o" : 20,         // block offset
    "n" : 32,         // number of blocks
    "b" : "0x130080" // A missing blockId bitmap starting from the offset
}
```

Handling errors from the AWS IoT Streaming service

An error response that is sent to a device for both `DescribeStream` and `GetStream` APIs contains a client token, an error code and an error message. A typical error response looks like the following example.

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- "`o`" is the error code that indicates the reason an error occurred. Refer to the error codes later in this section for more details.
- "`m`" is the error message that contains details of the error.
- "`c`" is the client token field. This may be returned if it was given in the `DescribeStream` request. You can use the client token to associate the response with its request.

  The client token field is not always included in an error response. When the client token given in the request isn't valid or is malformed, it's not returned in the error response.

Note
For backward compatibility, fields in the error response may be in non-abbreviated form. For example, the error code might be designated by either "code" or "o" fields and the client token field may be designated by either "clientToken" or "c" fields. We recommend that you use the abbreviation form shown above.

InvalidTopic

The MQTT topic of the stream message is invalid.

InvalidJson

The Stream request is not a valid JSON document.

InvalidCbor

The Stream request is not valid CBOR document.

InvalidRequest

The request is generally identified as malformed. For more information, see the error message.

Unauthorized

The request is not authorized to access the stream data files in the storage medium, such as Amazon S3. For more information, see the error message.

BlockSizeOutOfBounds

The block size is out of bounds. Refer to the "**AWS IoT Streaming**" section in AWS IoT Core Service Quotas.

OffsetOutOfBounds

The offset is out of bounds. Refer to the "**AWS IoT Streaming**" section in AWS IoT Core Service Quotas.

BlockCountLimitExceeded

The number of request block(s) is out of bounds. Refer to the "**AWS IoT Streaming**" section in AWS IoT Core Service Quotas.

BlockBitmapLimitExceeded

The size of the request bitmap is out of bounds. Refer to the "**AWS IoT Streaming**" section in AWS IoT Core Service Quotas.

ResourceNotFound

The requested stream, files, file versions or blocks were not found. Refer to the error message for more details.

VersionMismatch

The stream version in the request doesn't match with the stream version in the Streaming service. This indicates that the stream data had been modified since the stream version was initially received by the device.

ETagMismatch

The S3 ETag in the stream doesn't match with the ETag of the latest S3 object version.

InternalError

An internal error occurred in the Streaming service.

An example use case in FreeRTOS OTA

The FreeRTOS OTA (over-the-air) agent uses the AWS IoT Streaming service to transfer FreeRTOS firmware images to FreeRTOS devices. To send the initial data set to a device, it uses the AWS IoT Job service to schedule an OTA update job to FreeRTOS devices.

For a reference implementation of a Streaming service client, see FreeRTOS OTA agent codes in the FreeRTOS GitHub website.

AWS IoT Device Defender

AWS IoT Device Defender is a security service that allows you to audit the configuration of your devices, monitor connected devices to detect abnormal behavior, and mitigate security risks. It gives you the ability to enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised.

IoT fleets can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make fleet setup complex and error-prone. And because devices are often constrained in computational power, memory, and storage capabilities, this limits the use of encryption and other forms of security on the devices themselves. Also, devices often use software with known vulnerabilities. These factors make IoT fleets an attractive target for hackers and make it difficult to secure your device fleet on an ongoing basis.

AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. AWS IoT Device Defender can audit device fleets to ensure they adhere to security best practices and detect abnormal behavior on devices.

AWS training and certification

Take the following course to get started with AWS IoT Device Defender: AWS IoT Device Defender Primer.

Getting started with AWS IoT Device Defender

You can use the following tutorials to work with AWS IoT Device Defender.

Topics

Setting up

Before you use AWS IoT Device Defender for the first time, complete the following tasks:

- Sign up for AWS (p. 714)
- Create an IAM user (p. 715)

These tasks create an AWS account and an IAM user with administrator privileges for the account.

Sign up for AWS

When you sign up for AWS, your account is automatically signed up for all services in AWS, including AWS IoT Device Defender. If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Note your AWS account number, because you need it for the next task.

Create an IAM user

This procedure describes how to create a IAM user for yourself and add that user to a group that has administrative permissions from an attached managed policy.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   > **Note**
   > We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    > **Note**
    > You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access management and Example policies.

Audit guide

This tutorial provides instructions on how to configure a recurring audit, setting up alarms, reviewing audit results and mitigating audit issues.

Topics

Prerequisites

To complete this tutorial, you need the following:

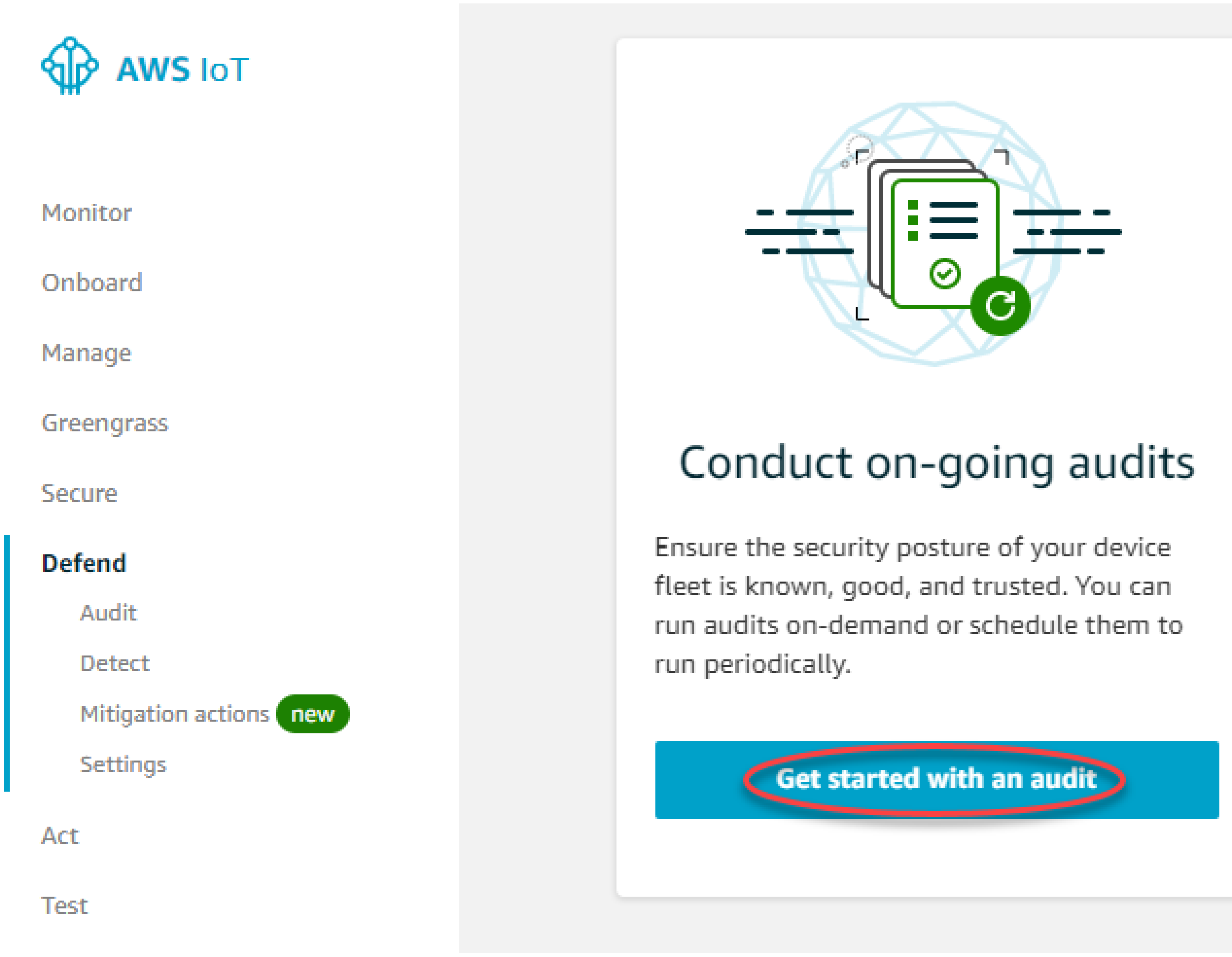
- An AWS account. If you don't have this, see Setting up.

Enable audit checks

In the following procedure, you enable audit checks that look at account and device settings and policies to ensure security measures are in place. In this tutorial we instruct you to enable all audit checks, but you're able to select whichever checks you wish.

Audit pricing is per device count per month (fleet devices connected to AWS IoT). Therefore, adding or removing audit checks would not affect your monthly bill when using this feature.

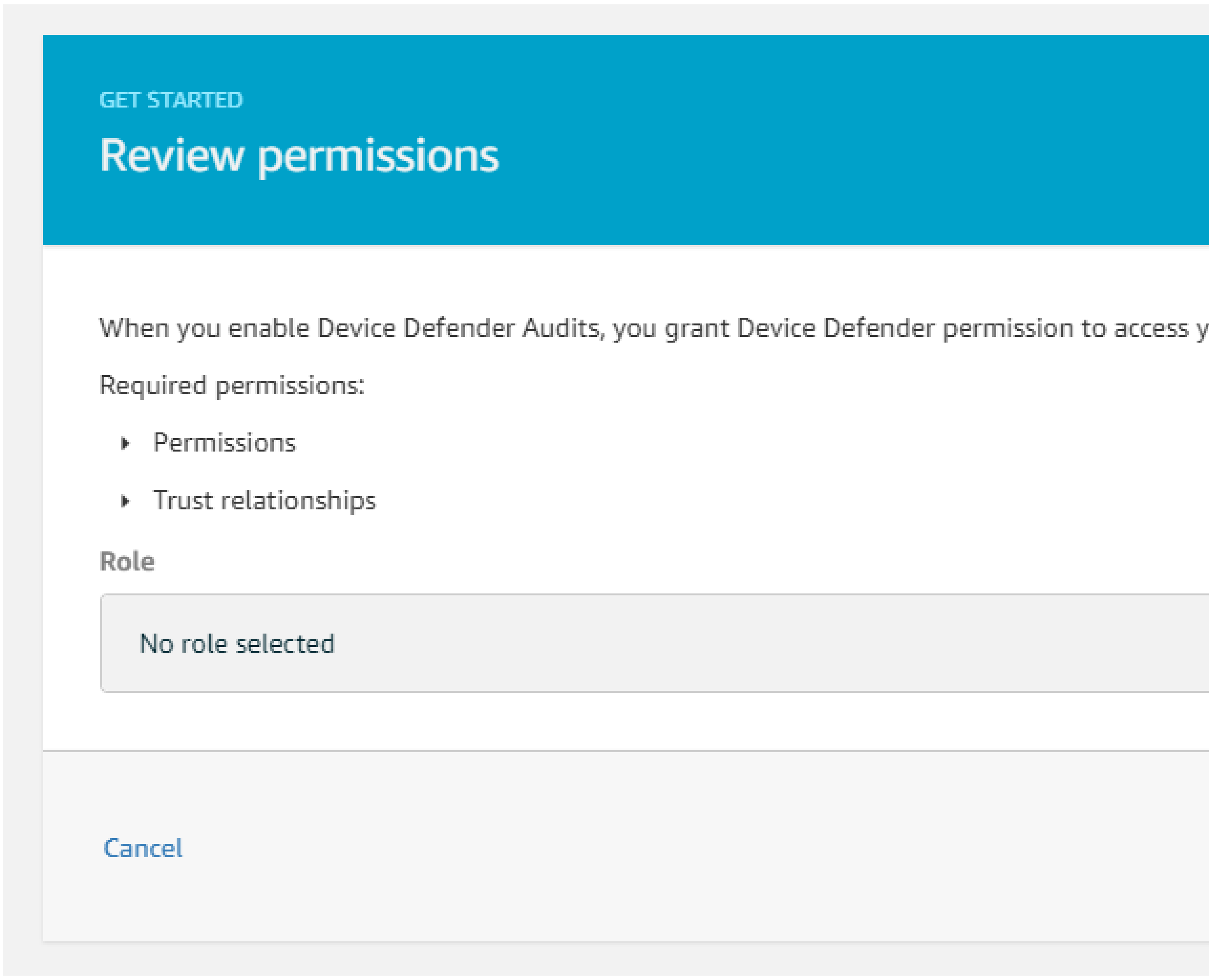
1. In the AWS IoT console, in the navigation pane, expand **Defend** and select **Get started with an audit**.



Conduct on-going audits

Ensure the security posture of your device fleet is known, good, and trusted. You can run audits on-demand or schedule them to run periodically.

2. The **Get started with Device Defender Audit** screen gives an overview of the steps required to enable the audit checks. Once you've reviewed the screen, select **Next**.
3. If you already have a role to use, you can select it. Otherwise select **Create Role** and name it `AWSIoTDeviceDefenderAudit`.


GET STARTED

Review permissions

When you enable Device Defender Audits, you grant Device Defender permission to access y

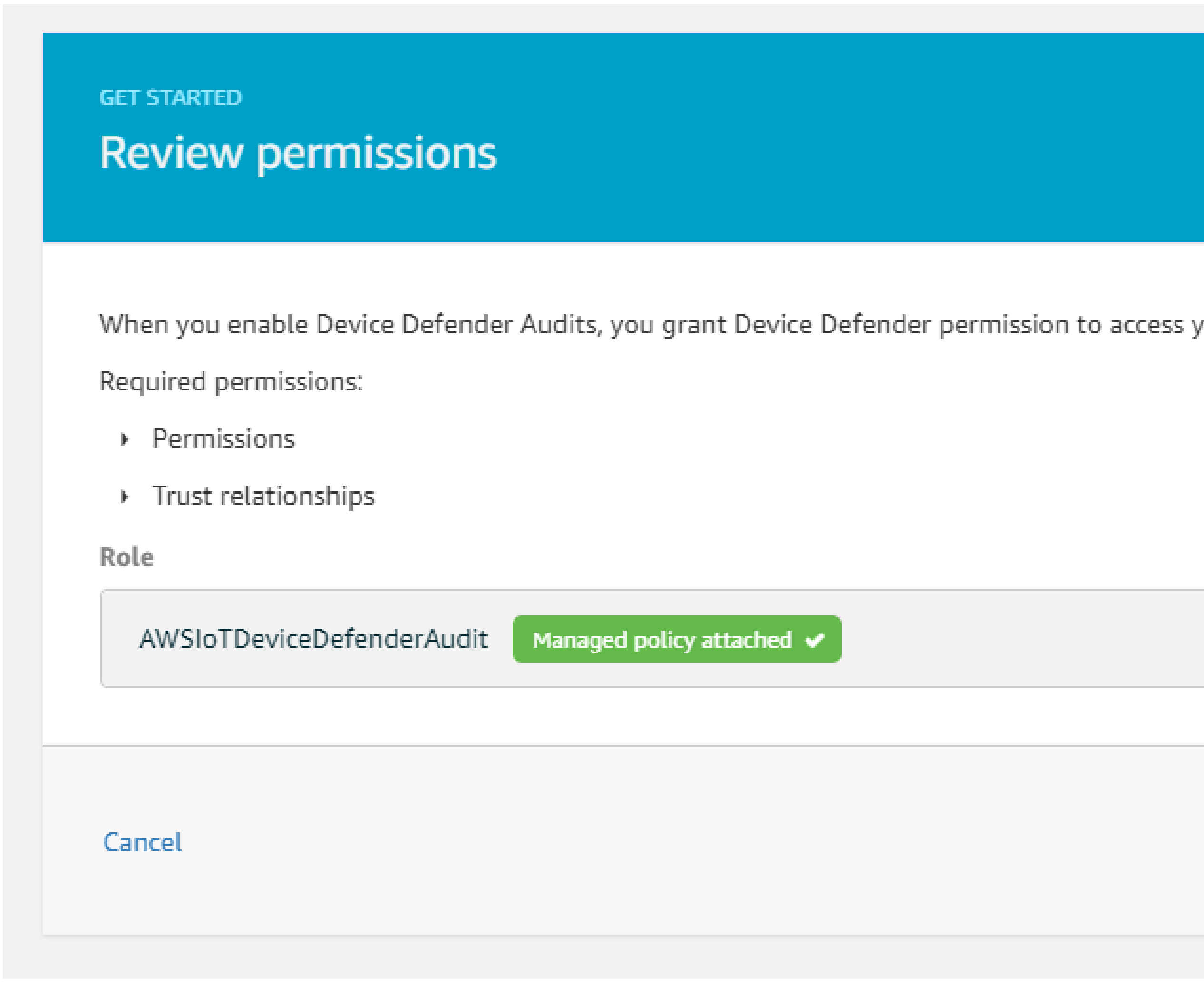
Required permissions:

- Permissions
- Trust relationships

Role

No role selected

You should see the required permissions automatically attached to the role. Select the triangles next to **Permissions** and **Trust relationships** to see what permissions are granted. Select **Next** when you're ready to move on.


GET STARTED

Review permissions

When you enable Device Defender Audits, you grant Device Defender permission to access y

Required permissions:

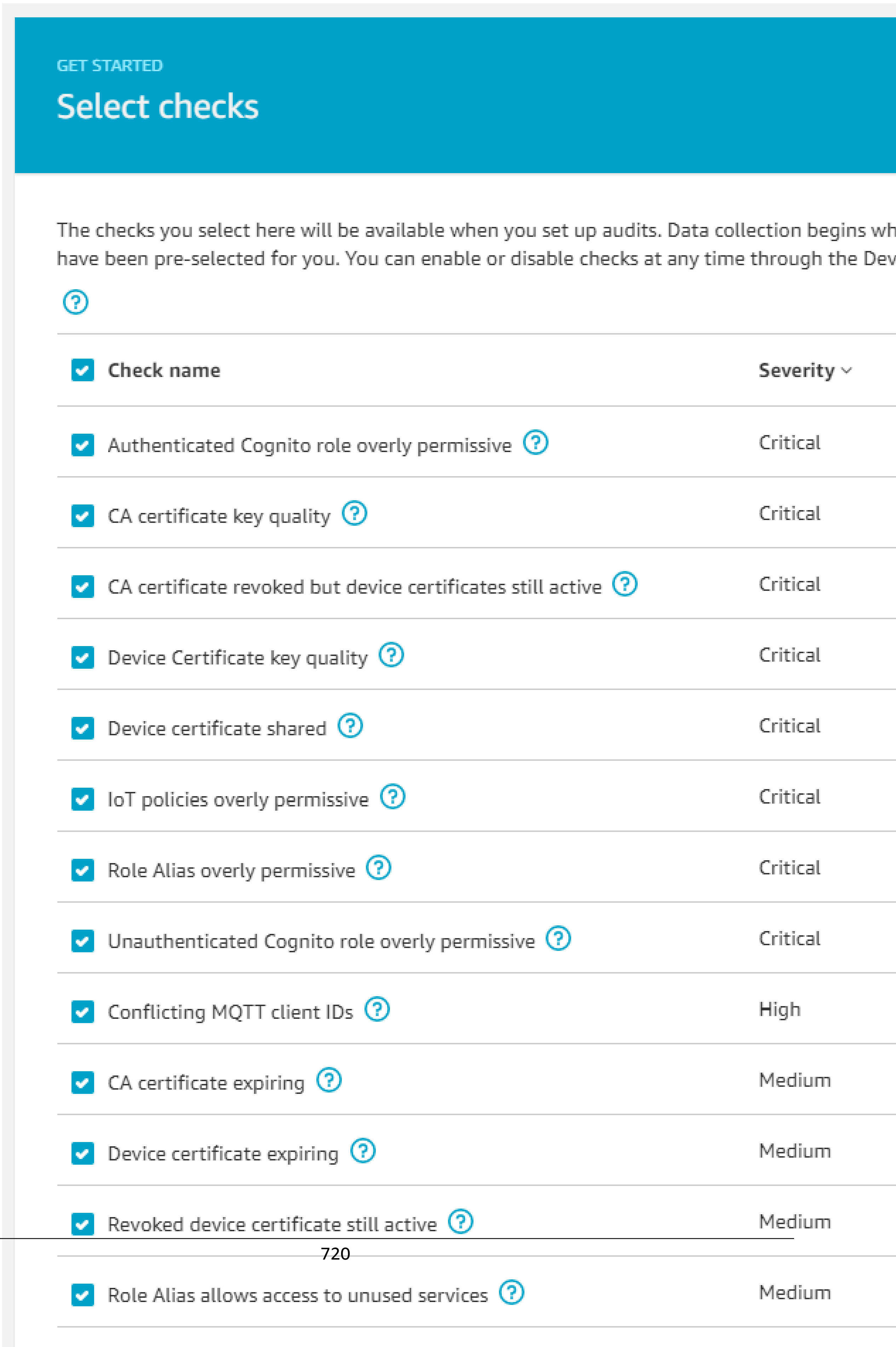
- Permissions
- Trust relationships

Role

AWSIoTDeviceDefenderAudit Managed policy attached ✓

4. On the **Select checks** screen you will see all audit checks you can select. For this tutorial, we instruct you to select all checks but you can select whichever checks you want. Next to each audit check is a help icon that describes what the audit check does. For more information about audit checks, see Audit Checks.

   Select **Next** once you've selected your checks.



GET STARTED

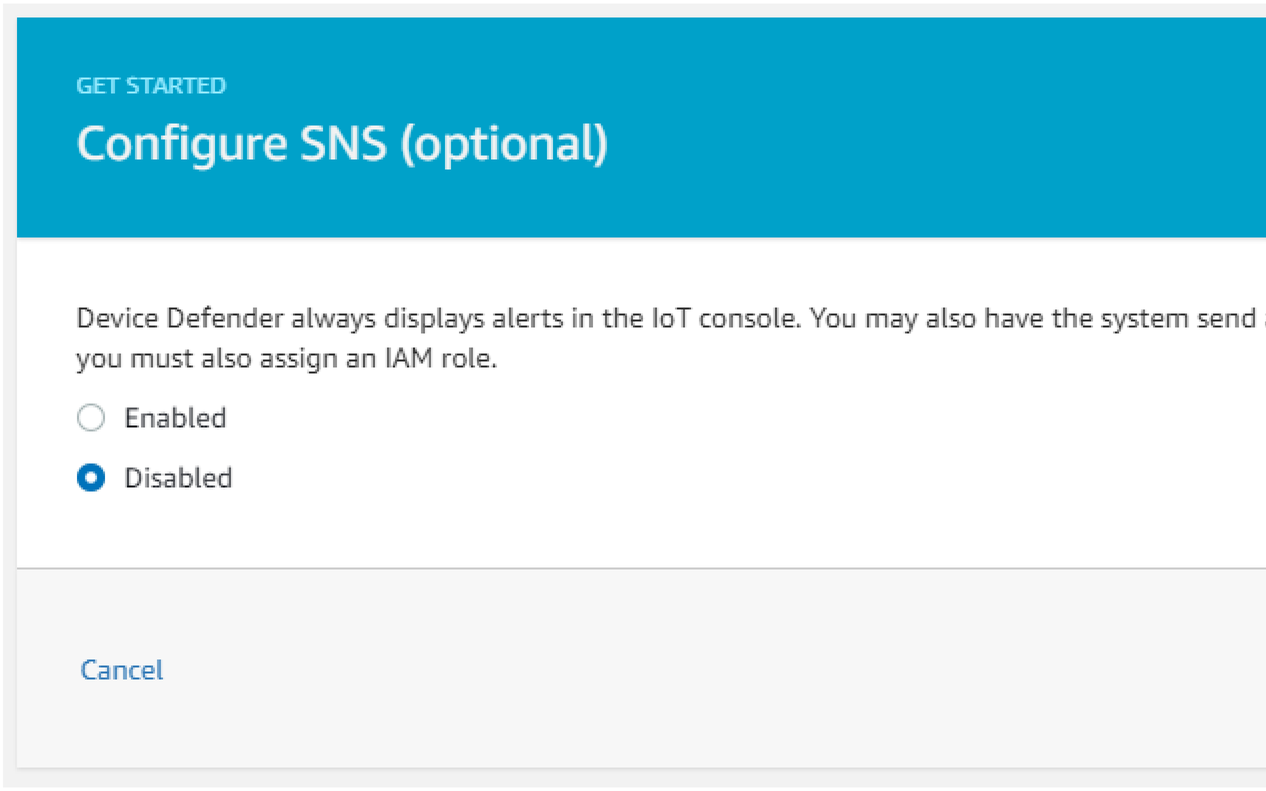
Select checks

The checks you select here will be available when you set up audits. Data collection begins wh
have been pre-selected for you. You can enable or disable checks at any time through the Dev

✓ Check name	Severity
✓ Authenticated Cognito role overly permissive	Critical
✓ CA certificate key quality	Critical
✓ CA certificate revoked but device certificates still active	Critical
✓ Device Certificate key quality	Critical
✓ Device certificate shared	Critical
✓ IoT policies overly permissive	Critical
✓ Role Alias overly permissive	Critical
✓ Unauthenticated Cognito role overly permissive	Critical
✓ Conflicting MQTT client IDs	High
✓ CA certificate expiring	Medium
✓ Device certificate expiring	Medium
✓ Revoked device certificate still active	Medium
✓ Role Alias allows access to unused services	Medium

You can always change your configured Audit checks under **Settings**.

5. On the **Configure SNS (optional)** screen, select **Enable audit**. If you'd like to enable SNS notifications, see Enable SNS notifications (optional) (p. 728).



GET STARTED

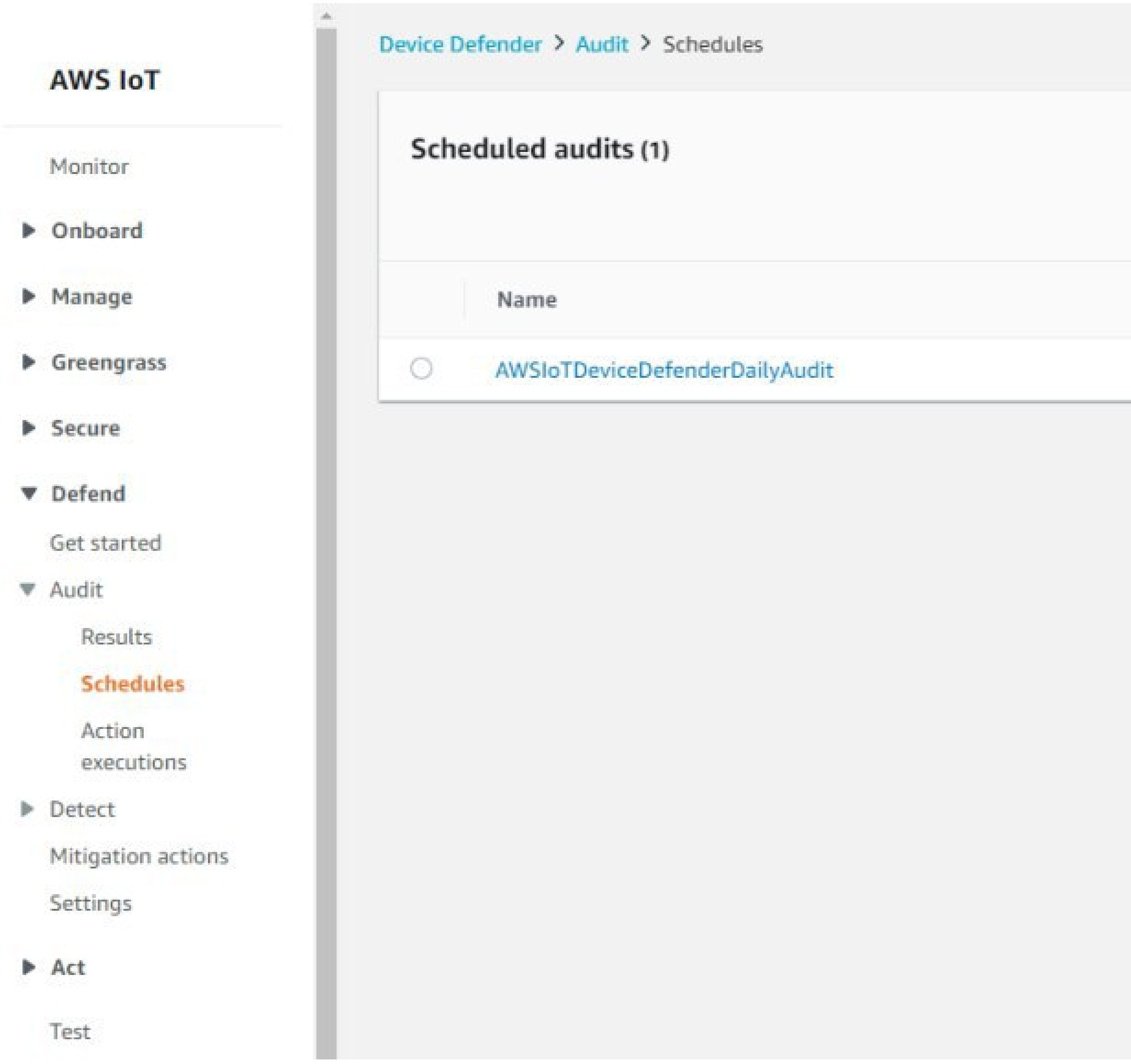
Configure SNS (optional)

Device Defender always displays alerts in the IoT console. You may also have the system send
you must also assign an IAM role.

( ) Enabled

(●) Disabled

6. You'll be redirected to **Schedules** under **Audit**.

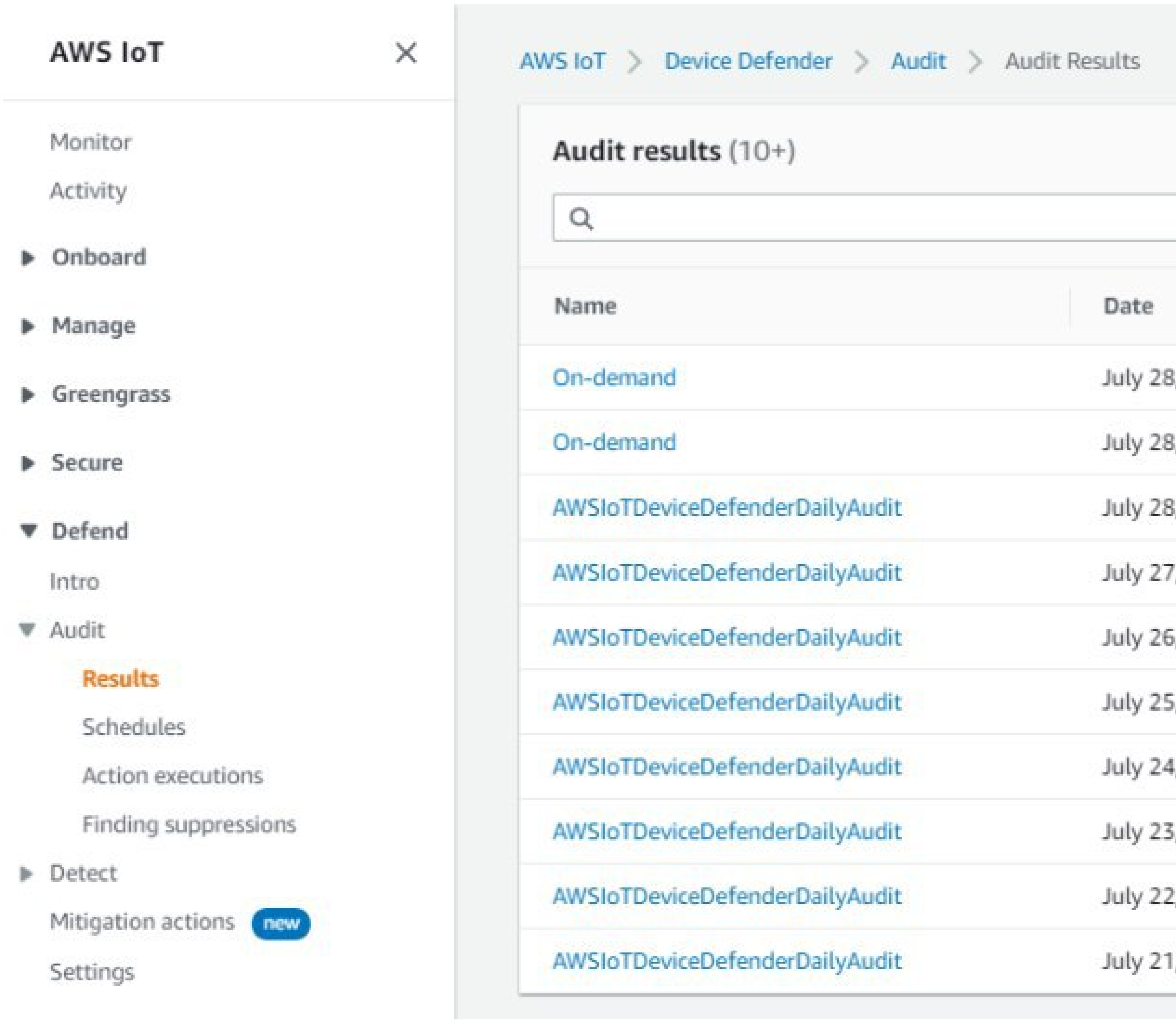


View audit results

The following procedure shows you how to view your audit results. In this tutorial, you see the audit results from the audit checks set up in Enable audit checks (p. 716) tutorial.

To view audit results

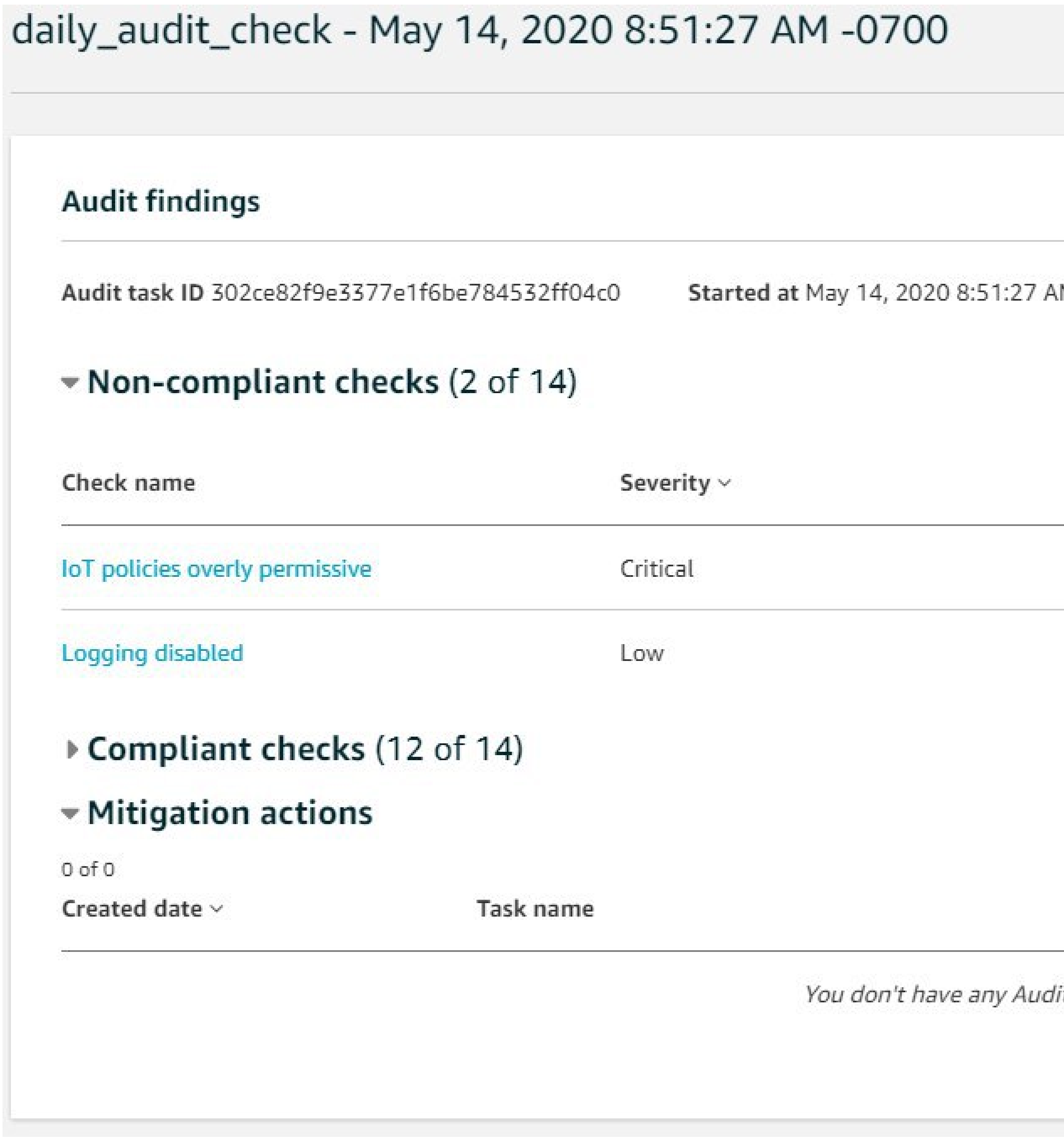
1. In the AWS IoT console, in the navigation pane, expand **Defend**, select **Audit**, and select **Results**.
2. The **Summary** will tell you if you have any non-compliant checks.



Audit results (10+)

Name	Date
On-demand	July 28
On-demand	July 28
AWSIoTDeviceDefenderDailyAudit	July 28
AWSIoTDeviceDefenderDailyAudit	July 27
AWSIoTDeviceDefenderDailyAudit	July 26
AWSIoTDeviceDefenderDailyAudit	July 25
AWSIoTDeviceDefenderDailyAudit	July 24
AWSIoTDeviceDefenderDailyAudit	July 23
AWSIoTDeviceDefenderDailyAudit	July 22
AWSIoTDeviceDefenderDailyAudit	July 21

3. Select the **Name** of the audit check you'd like to investigate.



daily_audit_check - May 14, 2020 8:51:27 AM -0700

Audit findings

Audit task ID 302ce82f9e3377e1f6be784532ff04c0 **Started at** May 14, 2020 8:51:27 AI

Non-compliant checks (2 of 14)

Check name	Severity
IoT policies overly permissive	Critical
Logging disabled	Low

Compliant checks (12 of 14)

Mitigation actions

0 of 0

Created date	Task name

You don't have any Audi

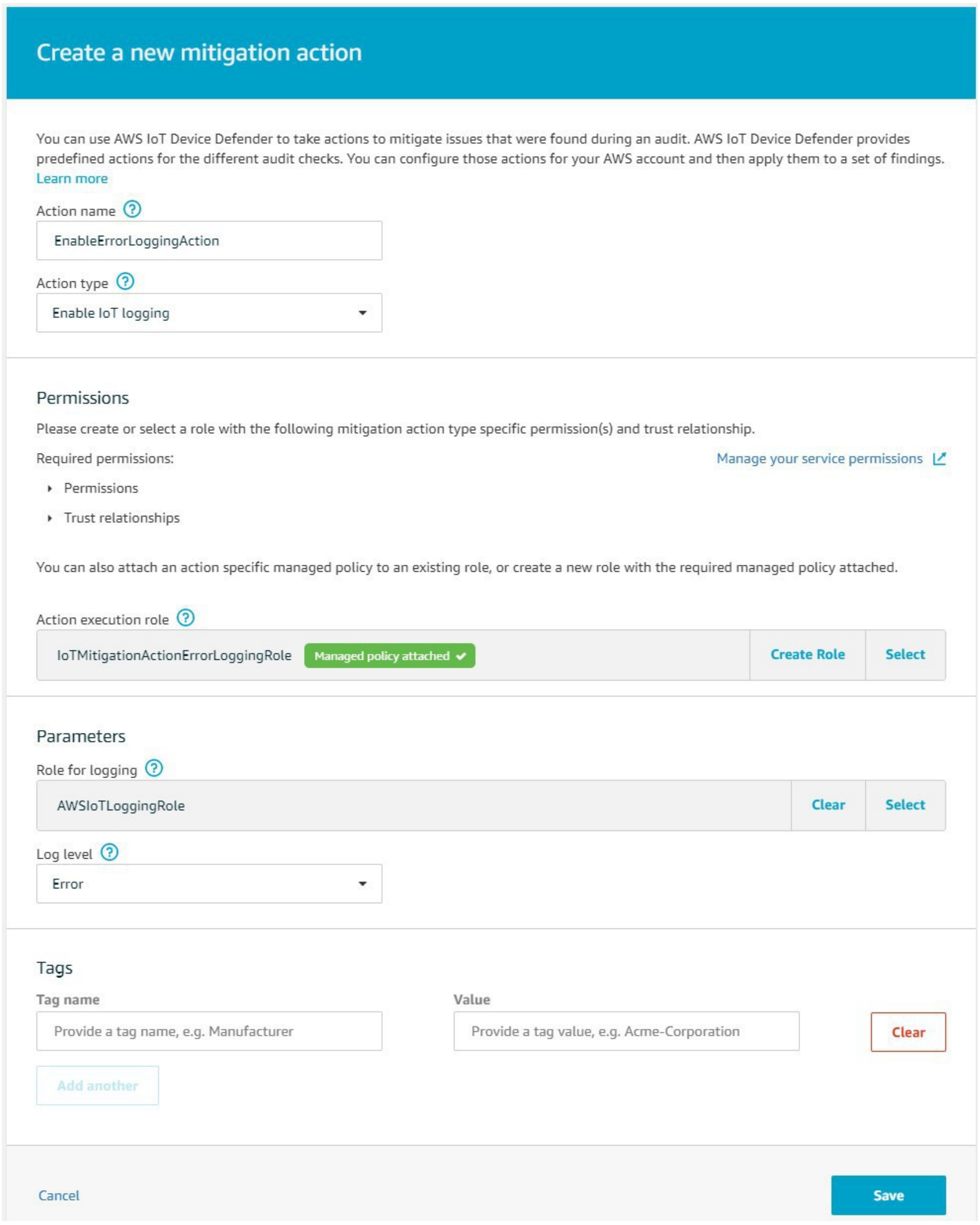
4. Use the question marks for guidance on how to make your non-compliant checks compliant. For example, you can follow Enable logging (optional) (p. 731) to make the "Logging disabled" check compliant.

Creating audit mitigation actions

In the following procedure, you will create a AWS IoT Device Defender Audit Mitigation Action to enable AWS IoT logging. Each audit check has mapped mitigation actions that will affect which **Action type** you choose for the audit check you want to fix. For more information, see Mitigation actions.

To use the AWS IoT console to create mitigation actions

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Mitigation Actions**.
3. On the **Mitigation Actions** page, choose **Create**.
4. On the **Create a Mitigation Action** page, in **Action name**, enter a unique name for your mitigation action such as *`EnableErrorLoggingAction`*.
5. In **Action type**, choose **Enable IoT logging**.
6. In **Action execution role**, select **Create Role**. For **Name**, use *`IoTMitigationActionErrorLoggingRole`*. Then, choose **Create role**.
7. In **Parameters**, under **Role for logging**, select `AWSIoTLoggingRole`. For **Log level**, choose `Error`.



Create a new mitigation action

You can use AWS IoT Device Defender to take actions to mitigate issues that were found during an audit. AWS IoT Device Defender provides predefined actions for the different audit checks. You can configure those actions for your AWS account and then apply them to a set of findings. Learn more

Action name: EnableErrorLoggingAction

Action type: Enable IoT logging

Permissions

Please create or select a role with the following mitigation action type specific permission(s) and trust relationship.

Required permissions:

- Permissions
- Trust relationships

You can also attach an action specific managed policy to an existing role, or create a new role with the required managed policy attached.

Action execution role: IoTMitigationActionErrorLoggingRole (Managed policy attached ✓)

Parameters

Role for logging: AWSIoTLoggingRole

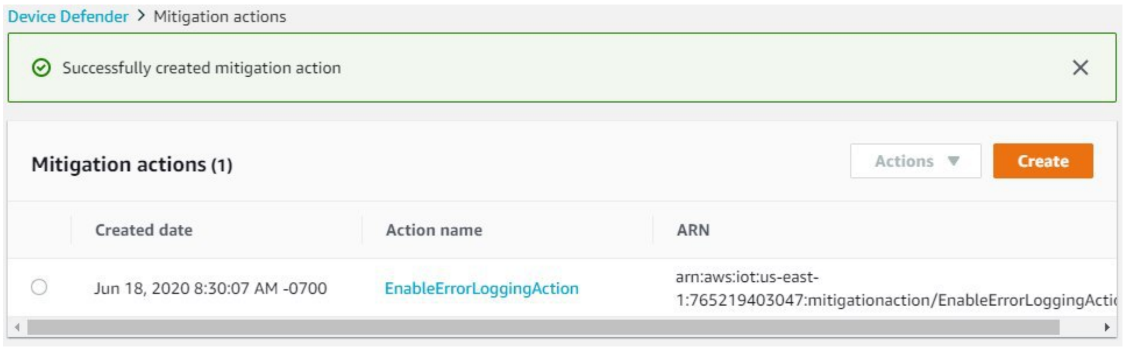
Log level: Error

Tags

Tag name: Provide a tag name, e.g. Manufacturer

Value: Provide a tag value, e.g. Acme-Corporation

8. Choose **Save** to save your mitigation action to your AWS account.
9. Once created, you will see the following screen indication that your mitigation action was created successfully.


Device Defender > Mitigation actions

Successfully created mitigation action

Mitigation actions (1)

Actions ▼ | Create

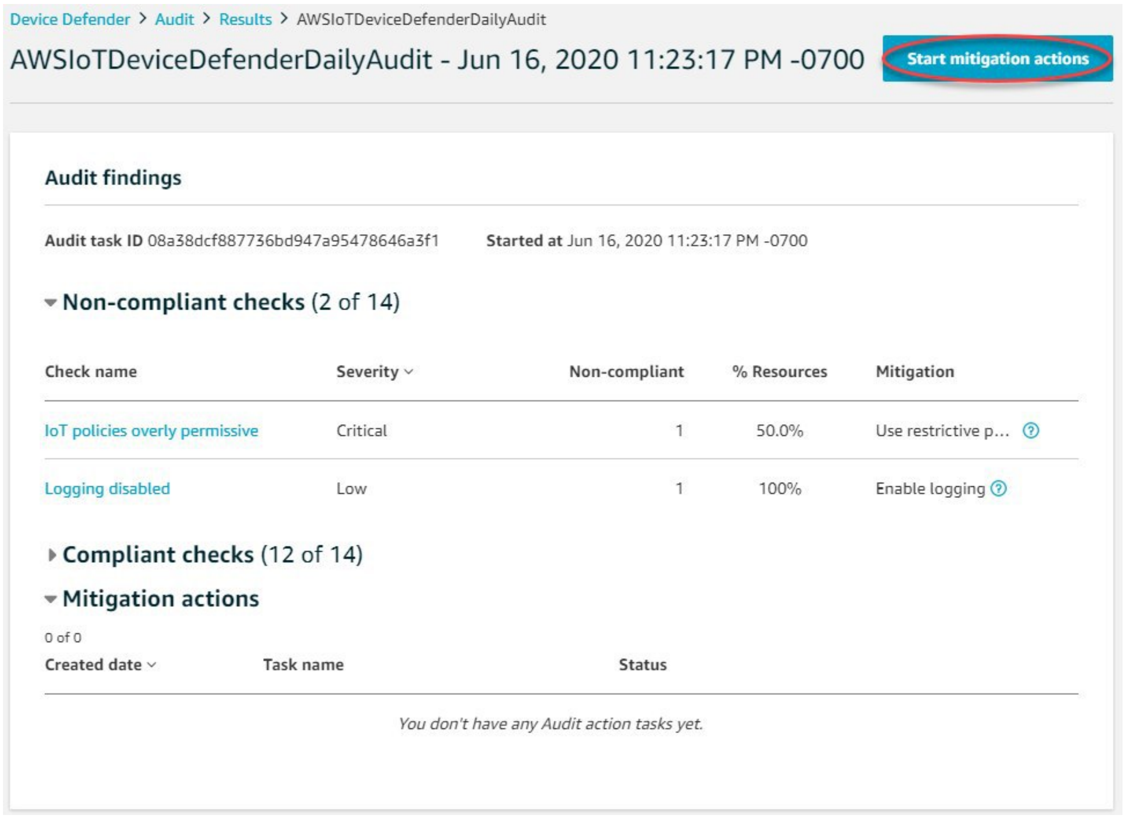
	Created date	Action name	ARN
○	Jun 18, 2020 8:30:07 AM -0700	EnableErrorLoggingAction	arn:aws:iot:us-east-1:765219403047:mitigationaction/EnableErrorLoggingActi

Apply mitigation actions to your audit findings

The following procedure shows you how to apply mitigation actions to your audit results.

To mitigate non-compliant audit findings

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Audit**, and then choose **Results**. Select the name of the audit that you want to respond to.
3. Check your results. Notice that `Logging disabled` is located under **Non-compliant checks**.
4. Select **Start mitigation actions**.


Device Defender > Audit > Results > AWSIoTDeviceDefenderDailyAudit

AWSIoTDeviceDefenderDailyAudit - Jun 16, 2020 11:23:17 PM -0700

Start mitigation actions

Audit findings

Audit task ID 08a38dcf887736bd947a95478646a3f1 **Started at** Jun 16, 2020 11:23:17 PM -0700

▾ Non-compliant checks (2 of 14)

Check name	Severity	Non-compliant	% Resources	Mitigation
IoT policies overly permissive	Critical	1	50.0%	Use restrictive p...
Logging disabled	Low	1	100%	Enable logging

▸ Compliant checks (12 of 14)

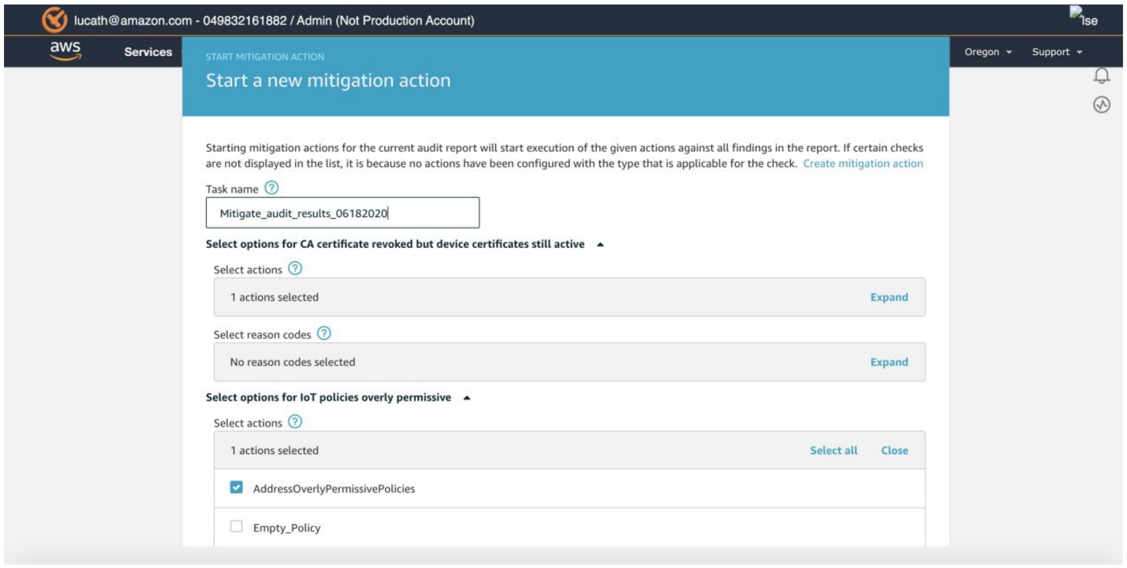
▾ Mitigation actions

0 of 0

Created date	Task name	Status

You don't have any Audit action tasks yet.

5. Under **Select actions**, select the appropriate actions for each non-compliant finding to address the issues.



lucath@amazon.com - 049832161882 / Admin (Not Production Account)

START MITIGATION ACTION

Start a new mitigation action

Starting mitigation actions for the current audit report will start execution of the given actions against all findings in the report. If certain checks are not displayed in the list, it is because no actions have been configured with the type that is applicable for the check. Create mitigation action

Task name: Mitigate_audit_results_06182020

Select options for CA certificate revoked but device certificates still active

Select actions: 1 actions selected

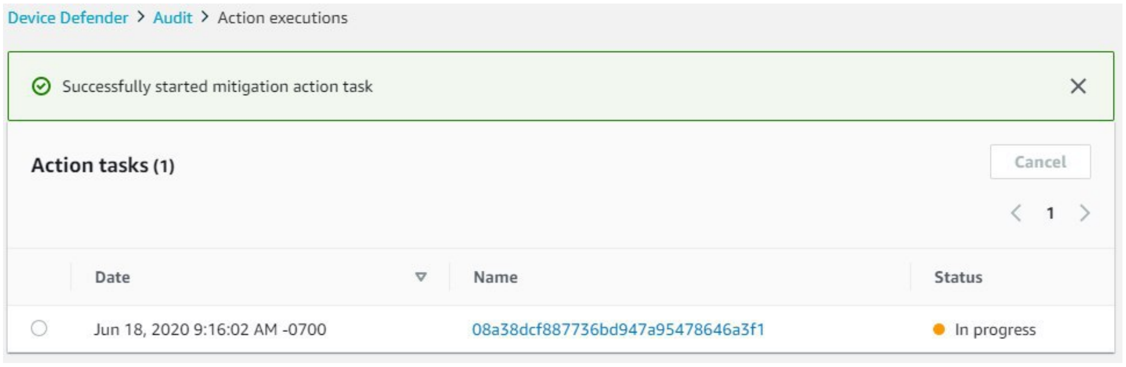
Select reason codes: No reason codes selected

Select options for IoT policies overly permissive

Select actions: 1 actions selected

- [x] AddressOverlyPermissivePolicies
- [ ] Empty_Policy

6. Select **Confirm**.
7. Once the mitigation action is started, it may take a few minutes for it to run.



Device Defender > Audit > Action executions

Successfully started mitigation action task

Action tasks (1)

	Date	Name	Status
	Jun 18, 2020 9:16:02 AM -0700	08a38dcf887736bd947a95478646a3f1	In progress

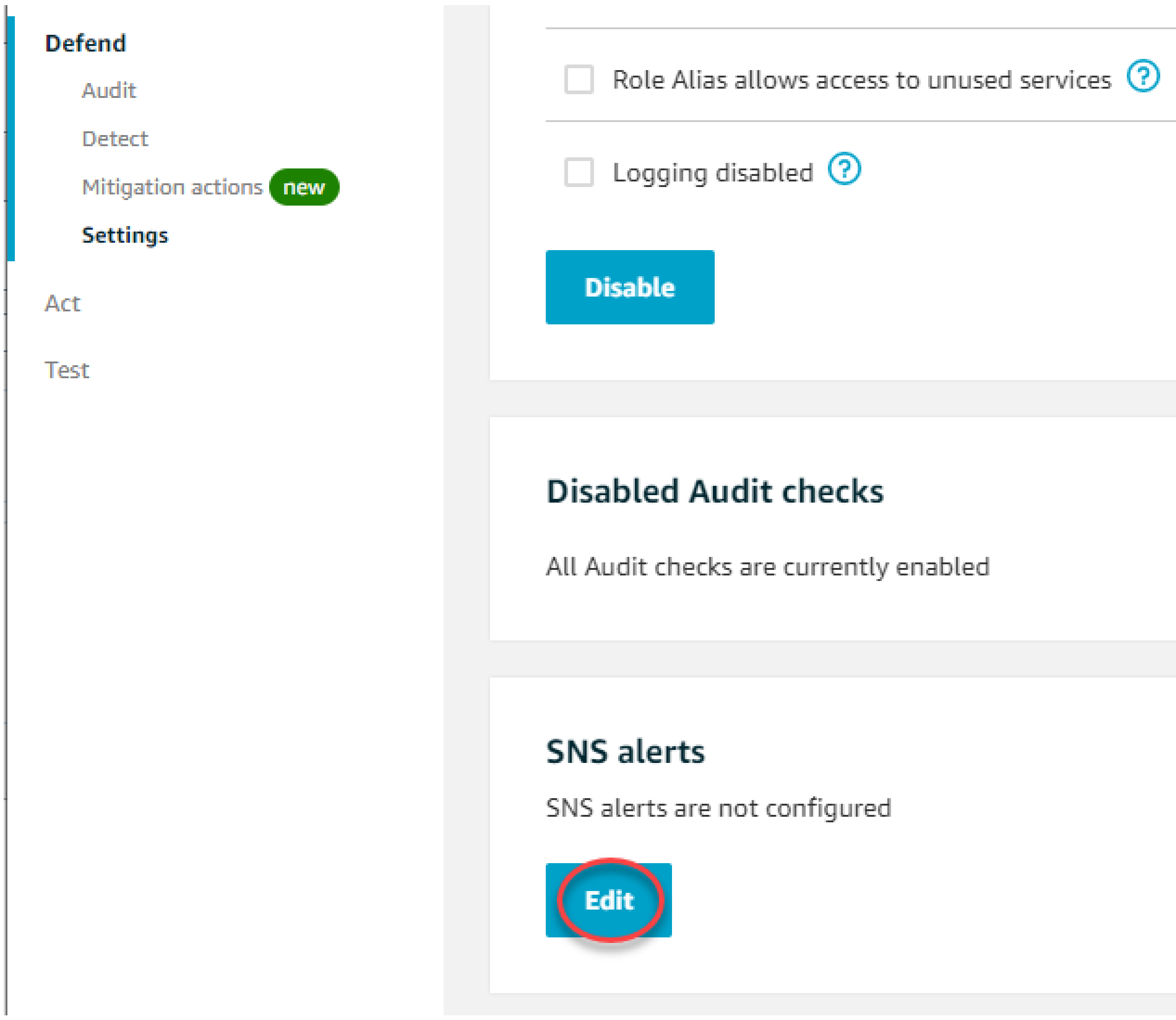
To check that the mitigation action worked

1. In the AWS console, in the navigation pane, select **Settings**.
2. Confirm that **Logs** are `Enabled` and the **Level of verbosity** is `Error`.

Enable SNS notifications (optional)

In the following procedure, you enable Simple Notifications Service (SNS) notifications to alert you when your audits identifies any non-compliant resources. In this tutorial you will set up notifications for the audit checks enabled in the Enable audit checks (p. 716) tutorial.

1. First, you need to create an IAM policy that provides access to Amazon SNS via the AWS Management Console. You can do this by following the Creating a AWS IoT Device Defender Audit IAM role (optional) (p. 734) process, but selecting **AmazonSNSRole** in step 8.
2. In the AWS IoT console, in the navigation pane, expand **Defend** and select **Settings**.
3. Under **SNS alerts**, select **Edit**.



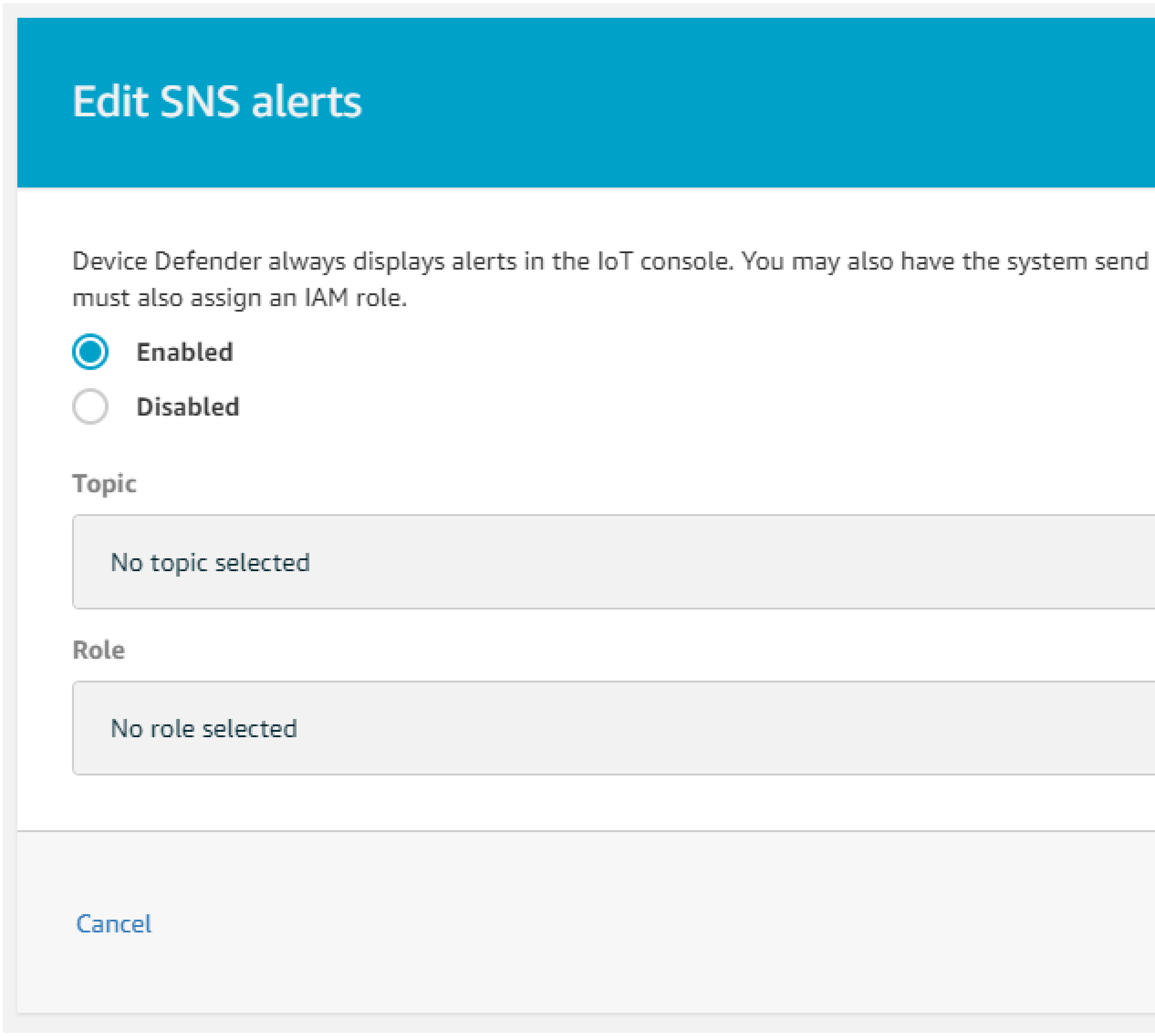
Disabled Audit checks

All Audit checks are currently enabled

SNS alerts

SNS alerts are not configured

4. On the **Edit SNS alerts** screen, select **Enabled**. Under **Topic**, select **Create**. Name the topic *`IoTDDNotifications`* and select **Create**. Under **Role**, select the role you created called *`AWSIoTDeviceDefenderAudit`*.



Edit SNS alerts

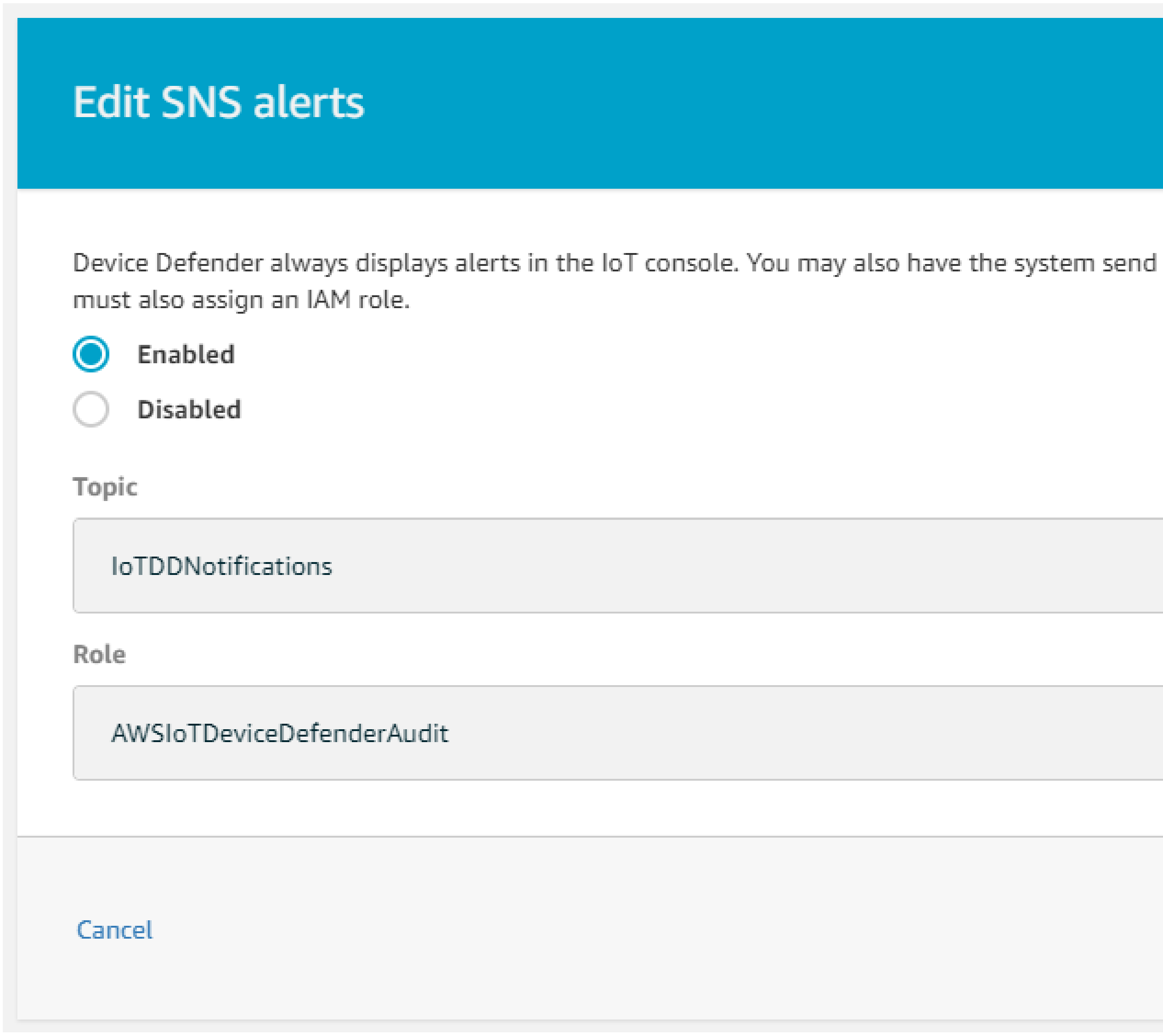
Device Defender always displays alerts in the IoT console. You may also have the system send
must also assign an IAM role.

[✓ Enabled] Disabled

Topic: No topic selected

Role: No role selected

Select **Update**.


Edit SNS alerts

Device Defender always displays alerts in the IoT console. You may also have the system send
must also assign an IAM role.

Enabled [✓]

Disabled

Topic

IoTDDNotifications

Role

AWSIoTDeviceDefenderAudit

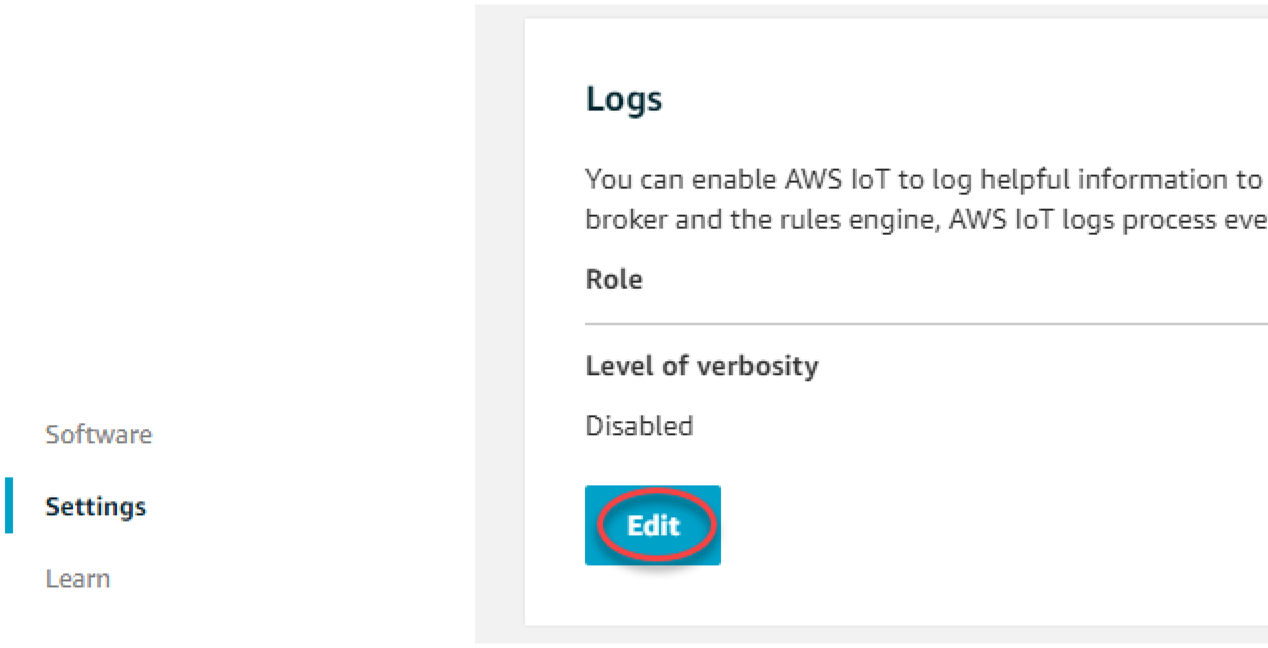
If you'd like to receive email or text in your Ops platforms through SNS, see Using Amazon SNS for user notifications.

Enable logging (optional)

This procedure describes how to enable AWS IoT to log information to CloudWatch Logs. This will allow you to view your audit results. Enabling logging may result in incurred charges.

To enable logging

1. In the AWS console, in the navigation pane, select **Settings**.
2. Under **Logs**, select **Edit**.


Software
Settings
Learn

Logs

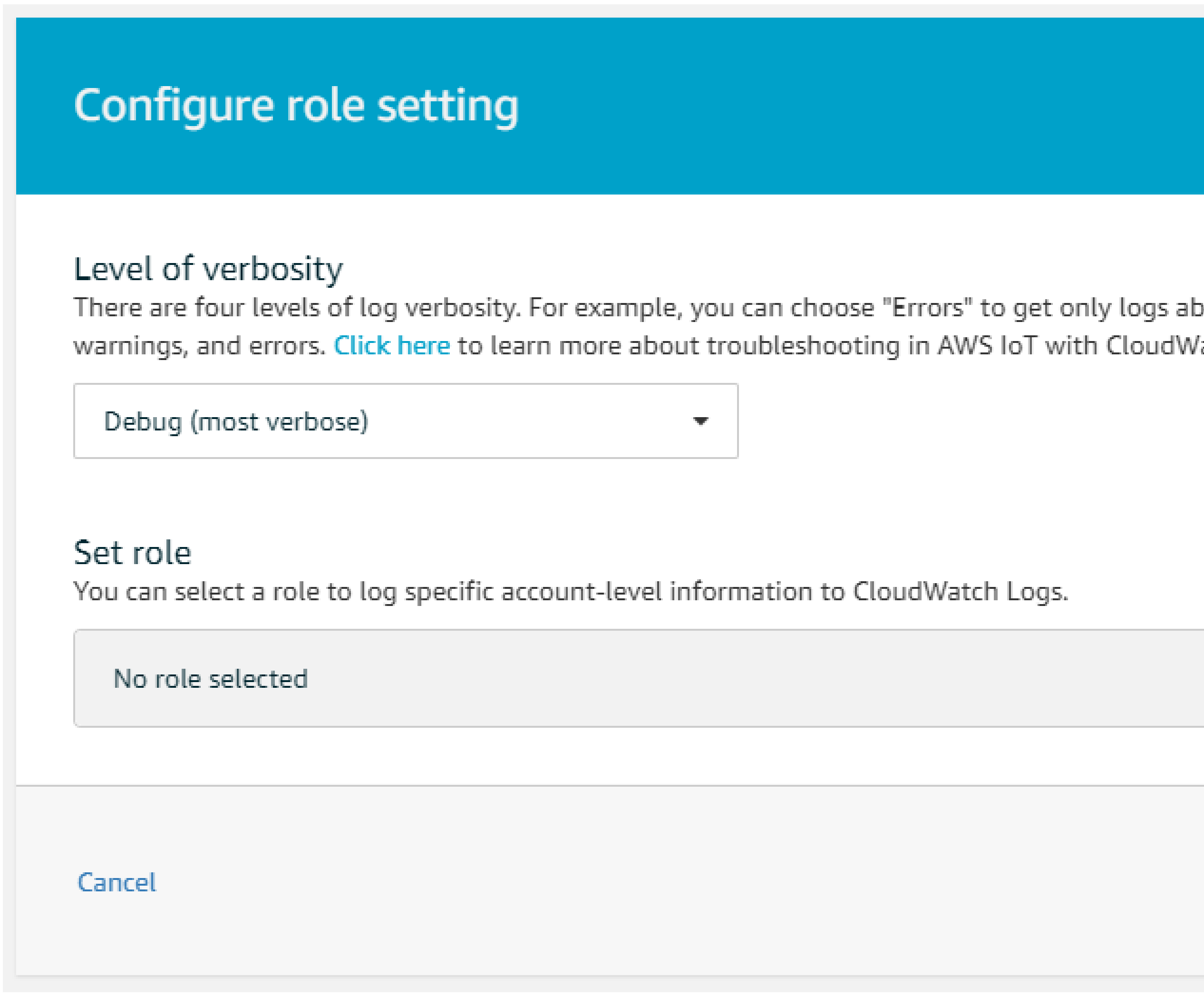
You can enable AWS IoT to log helpful information to
broker and the rules engine, AWS IoT logs process eve

Role

Level of verbosity

Disabled

3. Under **Level of verbosity**, select **Debug (most verbose)**.
4. Under **Set role**, select **Create Role** and name the role *`AWSIoTLoggingRole`*. A policy will automatically be attached.


Configure role setting

Level of verbosity

There are four levels of log verbosity. For example, you can choose "Errors" to get only logs ab
warnings, and errors. Click here to learn more about troubleshooting in AWS IoT with CloudWa

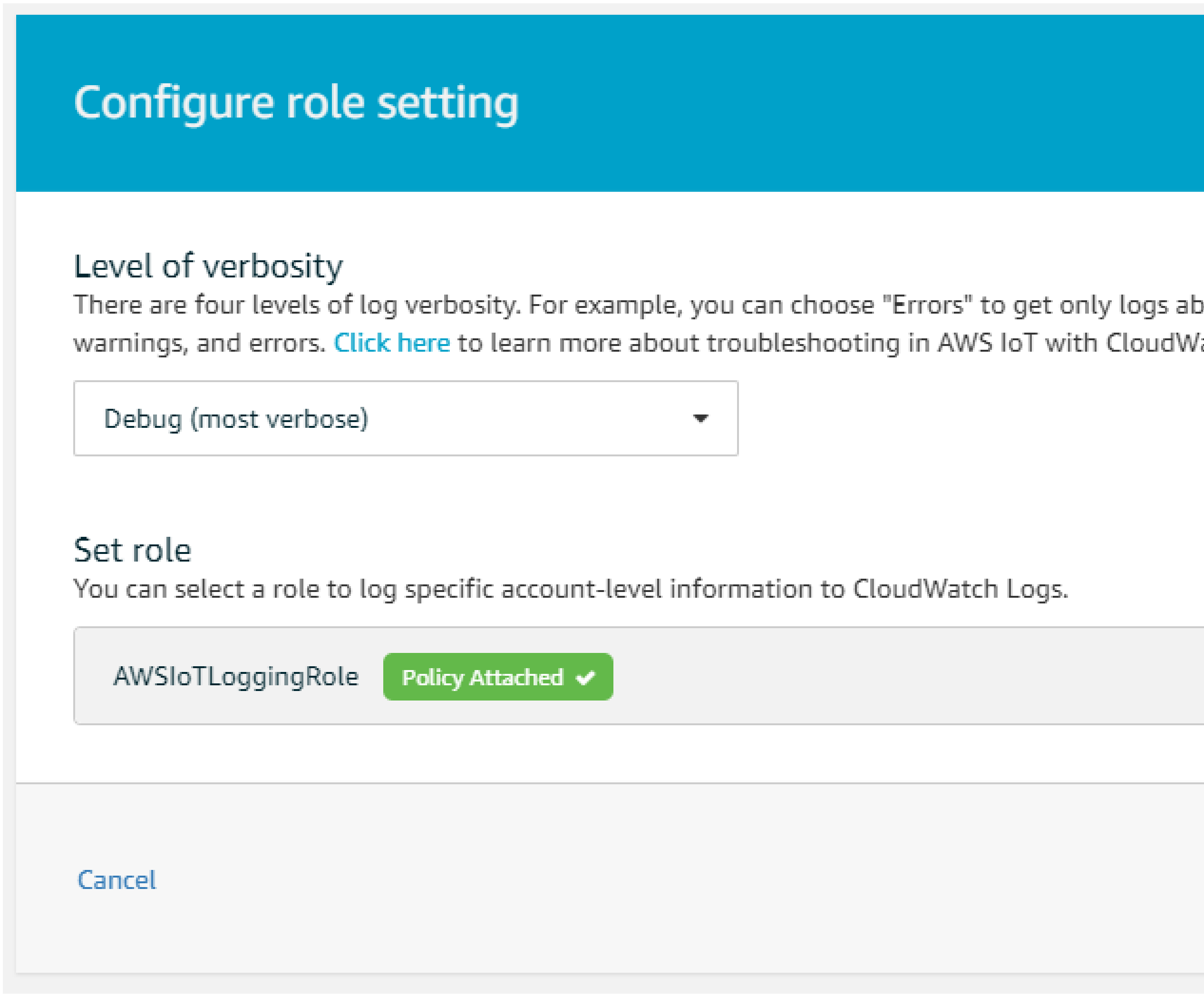
Debug (most verbose)

Set role

You can select a role to log specific account-level information to CloudWatch Logs.

No role selected

Select **Update**.



Configure role setting

Level of verbosity

There are four levels of log verbosity. For example, you can choose "Errors" to get only logs ab
warnings, and errors. Click here to learn more about troubleshooting in AWS IoT with CloudWa

Debug (most verbose)

Set role

You can select a role to log specific account-level information to CloudWatch Logs.

AWSIoTLoggingRole Policy Attached ✓

Cancel

Creating a AWS IoT Device Defender Audit IAM role (optional)

In the following procedure, you create a AWS IoT Device Defender Audit IAM role that provides AWS IoT Device Defender read access to AWS IoT.

1. Navigate to the IAM console at https://console.aws.amazon.com/iam/
2. In the navigation pane, chose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Attach existing policies directly**.
8. In the policy list, select the check box for **AWSIoTDeviceDefenderAudit**.
9. Choose **Next: Tags**.

10. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

ML Detect guide

ML Detect is in preview release for AWS IoT Device Defender and is subject to change.

In this Getting Started guide, you create an ML Detect Security Profile that uses machine learning (ML) to create models of expected behavior based on historical metric data from your devices. While ML Detect is creating the ML model, you can monitor its progress. Once the ML model is built, you can view and investigate alarms on an ongoing basis and mitigate identified issues.

For more information about ML Detect and its API and CLI commands, see ML Detect reference (p. 839).

This chapter contains the following sections:

Prerequisites

- An AWS account. If you don't have this, see Setting up.

How to use ML Detect in the console

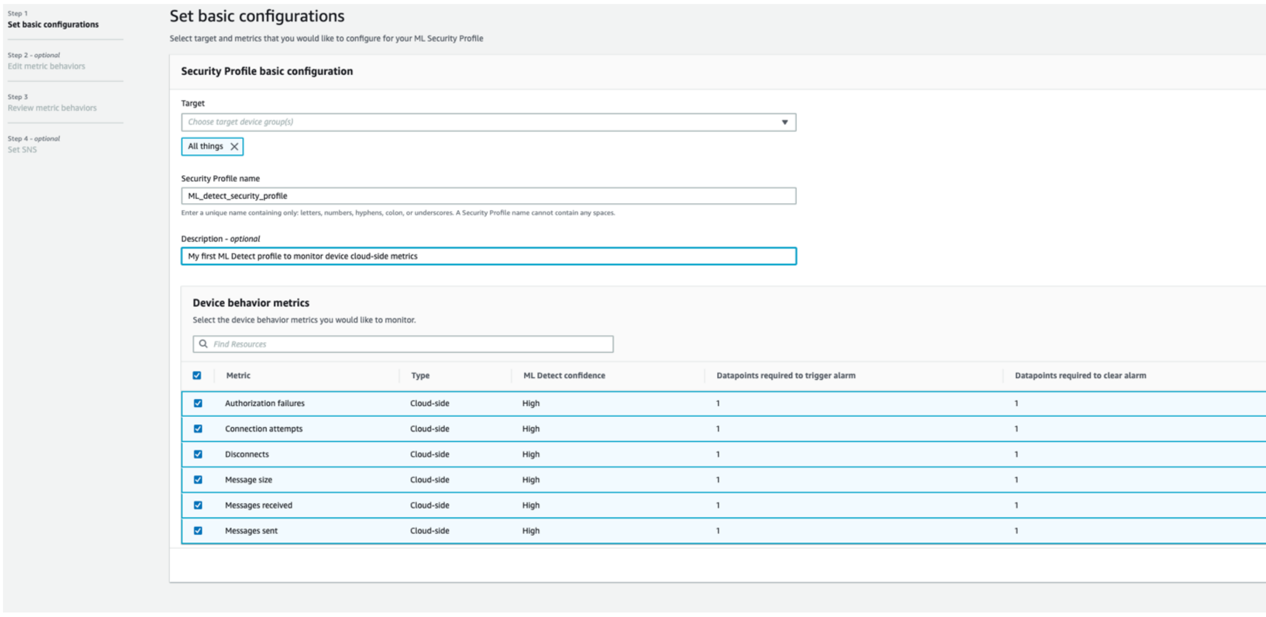
Tutorials

Enable ML Detect

The following procedures detail how to set up ML Detect in the console.

1. First, make sure your devices will create the minimum datapoints required as defined in ML Detect minimum requirements (p. 840) for ongoing training and refreshing of the model. For data collection to progress, ensure your Security Profile is attached to a target (for example, a thing group).
2. In the AWS IoT console, in the navigation pane, expand **Defend**. Choose **Detect**, **Security profiles**, **Create security profile**, and then **Create ML anomaly Detect profile**.
3. On the **Set basic configurations** page, do the following.
   - Under **Target**, choose your target device groups.
   - Under **Security profile name**, enter a name for your Security Profile.
   - Under **Device behavior metrics**, choose the metrics you'd like to monitor.

- (Optional) If you don't want to customize your metric configurations, select **Skip edit metric behaviors** and then select **Next**.



Step 1
Set basic configurations

Step 2 - *optional*
Edit metric behaviors

Step 3
Review metric behaviors

Step 4 - *optional*
Set SNS

Set basic configurations

Select target and metrics that you would like to configure for your ML Security Profile

Security Profile basic configuration

Target: *Choose target device group(s)*; All things

Security Profile name: ML_detect_security_profile

Enter a unique name containing only: letters, numbers, hyphens, colon, or underscores. A Security Profile name cannot contain any spaces.

Description - *optional*: My first ML Detect profile to monitor device cloud-side metrics

Device behavior metrics

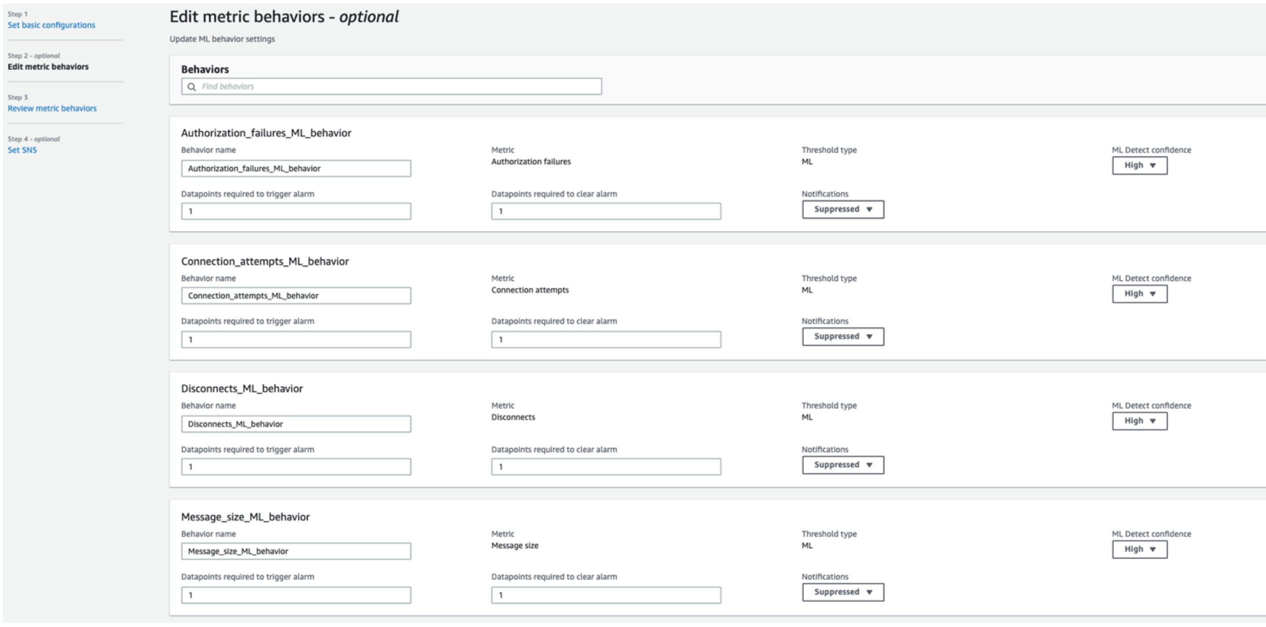
Select the device behavior metrics you would like to monitor.

Find Resources

✓	Metric	Type	ML Detect confidence	Datapoints required to trigger alarm	Datapoints required to clear alarm
✓	Authorization failures	Cloud-side	High	1	1
✓	Connection attempts	Cloud-side	High	1	1
✓	Disconnects	Cloud-side	High	1	1
✓	Message size	Cloud-side	High	1	1
✓	Messages received	Cloud-side	High	1	1
✓	Messages sent	Cloud-side	High	1	1

When you're done, choose **Next**.

4. On the **Update configurations** page, you can optionally customize your ML behavior settings.



Step 1
Set basic configurations

Step 2 - *optional*
Edit metric behaviors

Step 3
Review metric behaviors

Step 4 - *optional*
Set SNS

Edit metric behaviors - *optional*

Update ML behavior settings

Behaviors

Find behaviors

Authorization_failures_ML_behavior

Behavior name	Metric	Threshold type	ML Detect confidence
Authorization_failures_ML_behavior	Authorization failures	ML	High
Datapoints required to trigger alarm	**Datapoints required to clear alarm**	**Notifications**	
1	1	Suppressed	

Connection_attempts_ML_behavior

Behavior name	Metric	Threshold type	ML Detect confidence
Connection_attempts_ML_behavior	Connection attempts	ML	High
Datapoints required to trigger alarm	**Datapoints required to clear alarm**	**Notifications**	
1	1	Suppressed	

Disconnects_ML_behavior

Behavior name	Metric	Threshold type	ML Detect confidence
Disconnects_ML_behavior	Disconnects	ML	High
Datapoints required to trigger alarm	**Datapoints required to clear alarm**	**Notifications**	
1	1	Suppressed	

Message_size_ML_behavior

Behavior name	Metric	Threshold type	ML Detect confidence
Message_size_ML_behavior	Message size	ML	High
Datapoints required to trigger alarm	**Datapoints required to clear alarm**	**Notifications**	
1	1	Suppressed	

5. When you're done, choose **Next**.

6. On the **Review metric behaviors** screen, verify the behaviors you'd like machine learning to monitor.

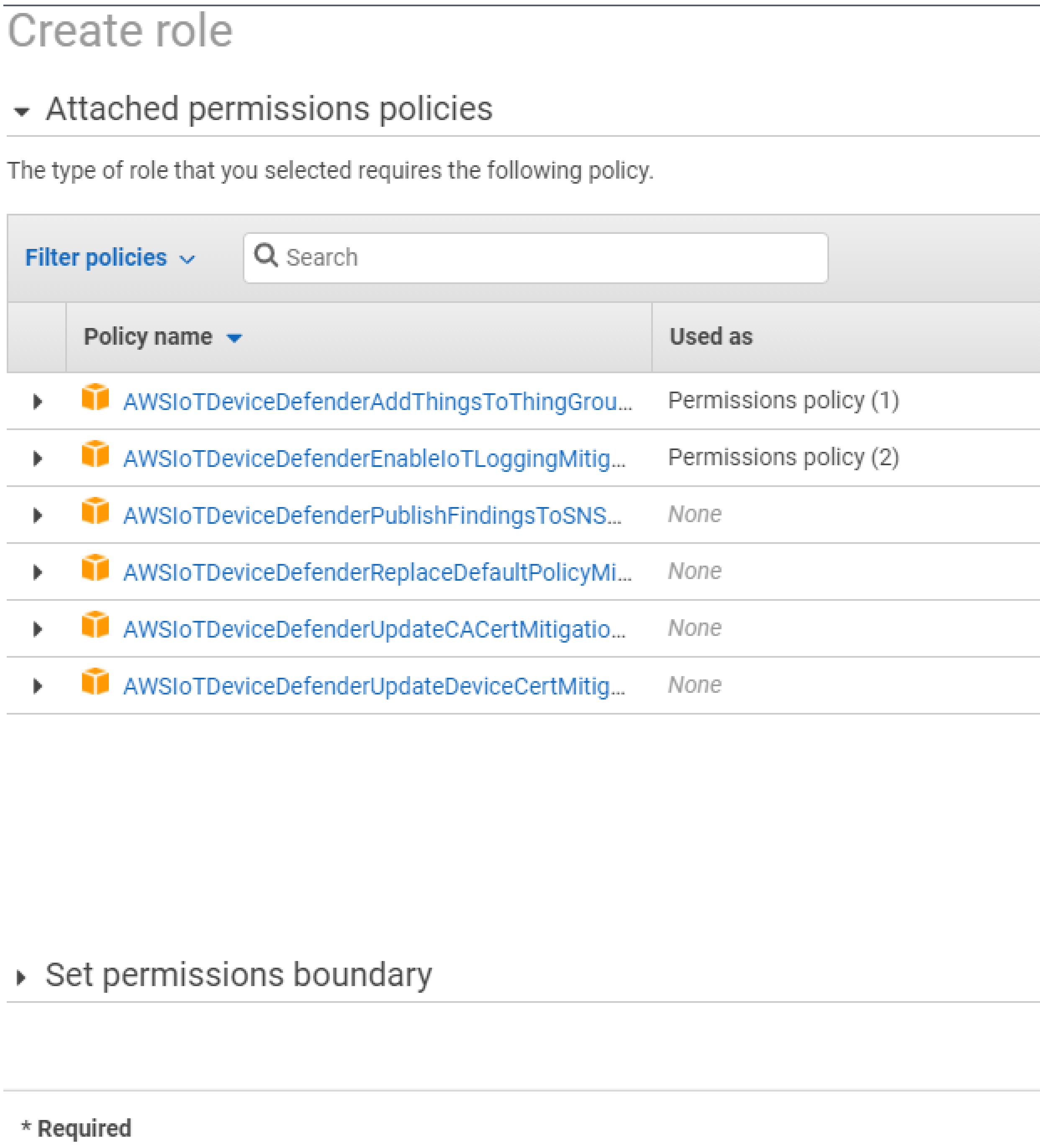
   (Optional) If you don't want to enable SNS, select **Skip next step and finish**.

   When you're done, choose **Next**.

7. (Optional) On the **Set SNS (optional)** page, specify an SNS topic for alarms when a device violates a behavior in your profile. Select an IAM role you will use to publish on the selected SNS topic.

   If you don't have an SNS role yet, use the following steps to create a role with the proper permissions and trust relationships required.

   - Navigate to the IAM console. In the navigation pane, choose **Roles** and then choose **Create role**.
   - Under **Select type of trusted entity**, select **AWS Service**. Then, under **Choose a use case**, choose **IoT** and under **Select your use case**, choose **IoT - Device Defender Mitigation Actions**. Choose **Next: Permissions** when done.
   - Under **Attached permissions policies**, ensure that **AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction** is selected then choose **Next: Tags**.



Create role

Attached permissions policies

The type of role that you selected requires the following policy.

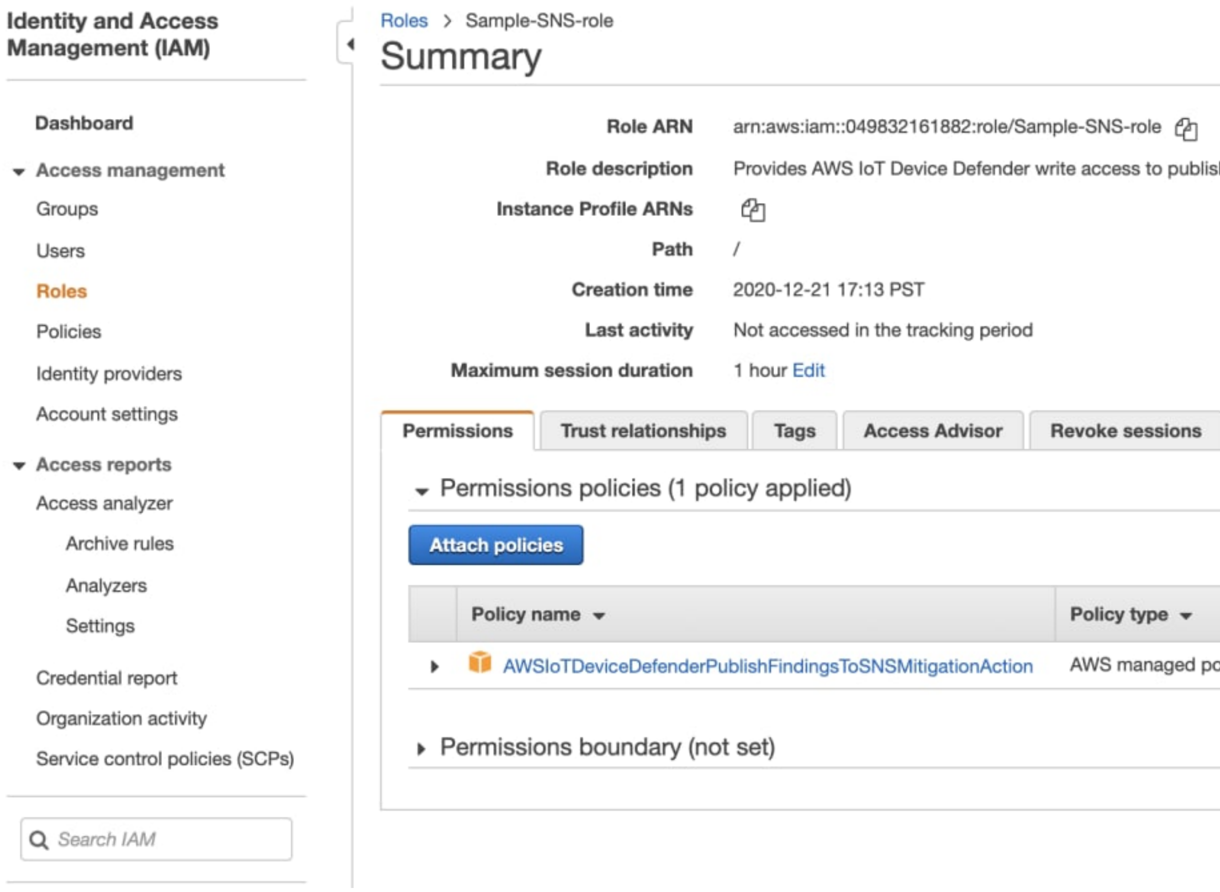
Filter policies | Search

Policy name	Used as
AWSIoTDeviceDefenderAddThingsToThingGrou...	Permissions policy (1)
AWSIoTDeviceDefenderEnableIoTLoggingMitig...	Permissions policy (2)
AWSIoTDeviceDefenderPublishFindingsToSNS...	None
AWSIoTDeviceDefenderReplaceDefaultPolicyMi...	None
AWSIoTDeviceDefenderUpdateCACertMitigatio...	None
AWSIoTDeviceDefenderUpdateDeviceCertMitig...	None

Set permissions boundary

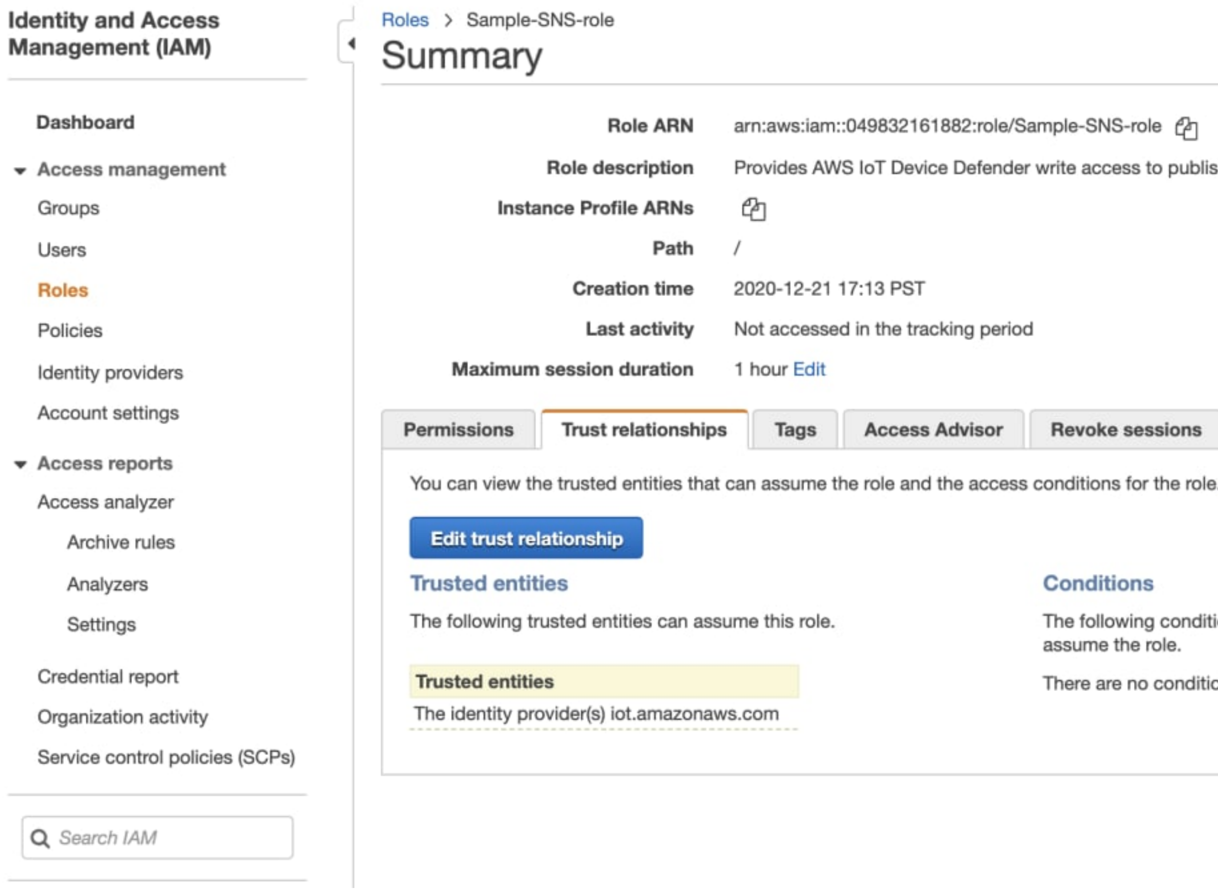
* Required

- Under **Add tags (optional)**, you can add any tags you'd like to associate with your role. Choose **Next: Review** when done.
- Under **Review**, give your role a name and ensure that **AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction** is listed under **Permissions** and **AWS service: iot.amazonaws.com** is listed under **Trust relationships**. Choose **Create role** when done.



Role ARN	arn:aws:iam::049832161882:role/Sample-SNS-role
Role description	Provides AWS IoT Device Defender write access to publis
Instance Profile ARNs	
Path	/
Creation time	2020-12-21 17:13 PST
Last activity	Not accessed in the tracking period
Maximum session duration	1 hour Edit

Policy name	Policy type
AWSIoTDeviceDefenderPublishFindingsToSNSMitigationAction	AWS managed po


Identity and Access Management (IAM)

Dashboard

Access management
Groups
Users
Roles
Policies
Identity providers
Account settings

Access reports
Access analyzer
Archive rules
Analyzers
Settings
Credential report
Organization activity
Service control policies (SCPs)

Search IAM

Roles > Sample-SNS-role

Summary

Role ARN	arn:aws:iam::049832161882:role/Sample-SNS-role
Role description	Provides AWS IoT Device Defender write access to publis
Instance Profile ARNs	
Path	/
Creation time	2020-12-21 17:13 PST
Last activity	Not accessed in the tracking period
Maximum session duration	1 hour Edit

You can view the trusted entities that can assume the role and the access conditions for the role

Trusted entities

The following trusted entities can assume this role.

Trusted entities
The identity provider(s) iot.amazonaws.com

Conditions

The following conditi
assume the role.

There are no conditio

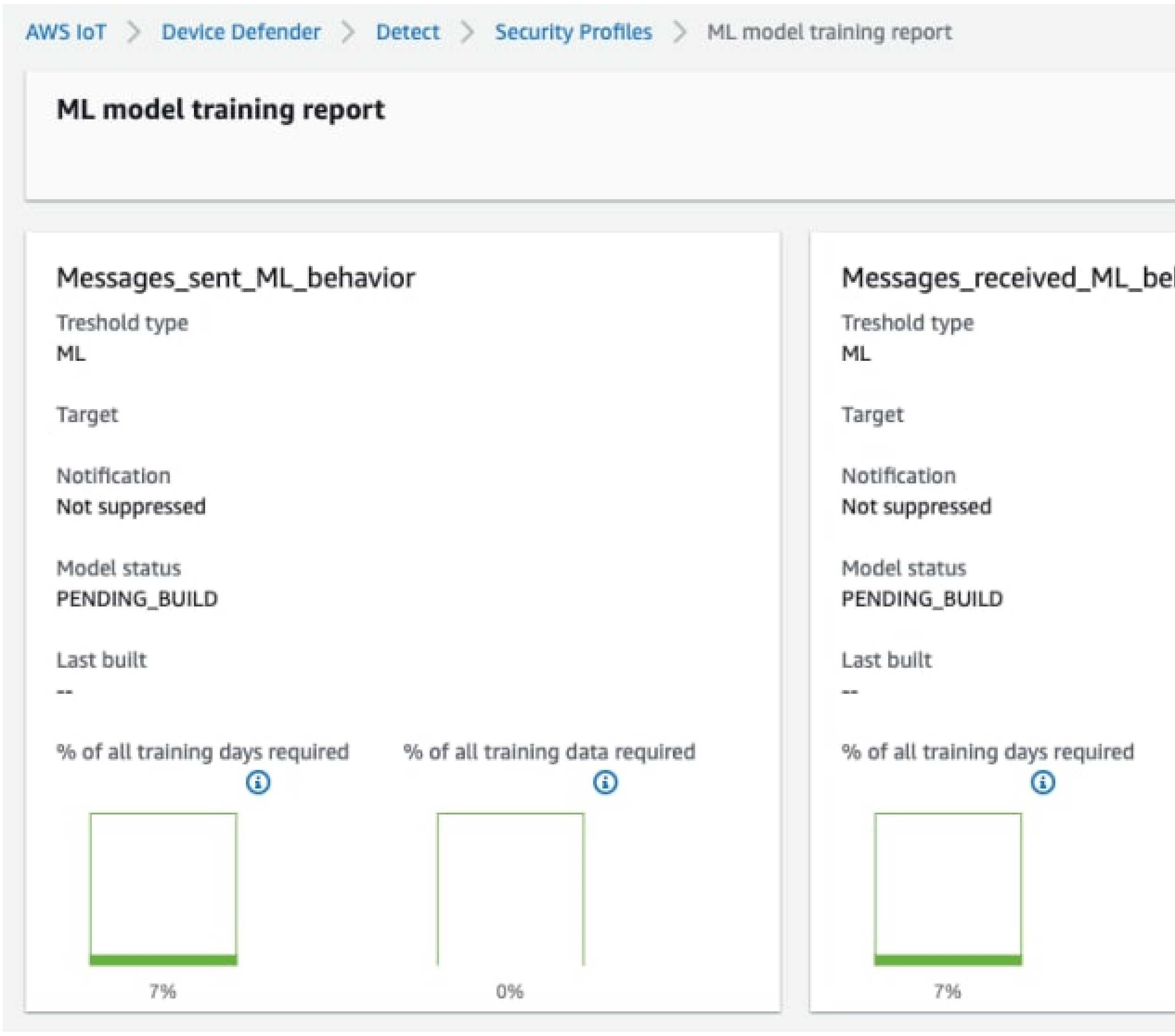
8. After you've created your Security Profile, you're redirected to the **Security Profiles** page, where newly created Security Profile is displayed.

   Note
   The initial ML model training and creation takes 14 days to complete. You can expect to see alarms after it's complete if there is any anomalous activity on your devices.

Monitor your ML model status

While your ML models are in the initial training period, you can monitor their progress at any time by taking the following steps.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.
2. On the **Security Profiles** page, select the check box next to the Security Profile you'd like to review. Then, choose **Other actions** and **View ML model training report**.
3. On the **ML model training report** page, check the training progress of your ML models.


AWS IoT > Device Defender > Detect > Security Profiles > ML model training report

ML model training report

Messages_sent_ML_behavior

Treshold type
ML

Target

Notification
Not suppressed

Model status
PENDING_BUILD

Last built
--

% of all training days required

7%

% of all training data required

0%

Messages_received_ML_be

Treshold type
ML

Target

Notification
Not suppressed

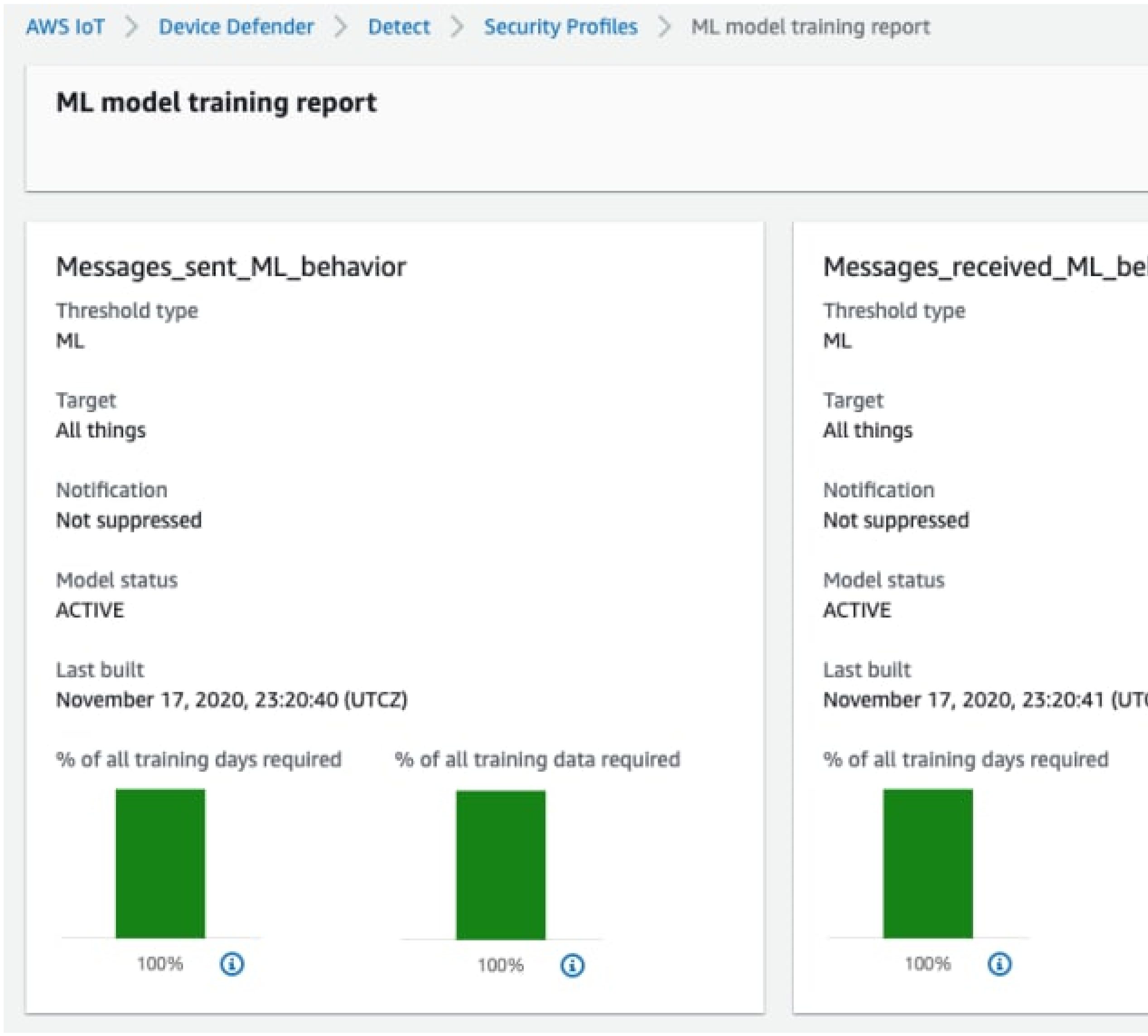
Model status
PENDING_BUILD

Last built
--

% of all training days required

7%

Once your model status is **Active**, it'll start making Detect decisions based on your usage and update the profile every day.


AWS IoT > Device Defender > Detect > Security Profiles > ML model training report

ML model training report

Messages_sent_ML_behavior

Threshold type
ML

Target
All things

Notification
Not suppressed

Model status
ACTIVE

Last built
November 17, 2020, 23:20:40 (UTCZ)

% of all training days required
100%

% of all training data required
100%

Messages_received_ML_be

Threshold type
ML

Target
All things

Notification
Not suppressed

Model status
ACTIVE

Last built
November 17, 2020, 23:20:41 (UT

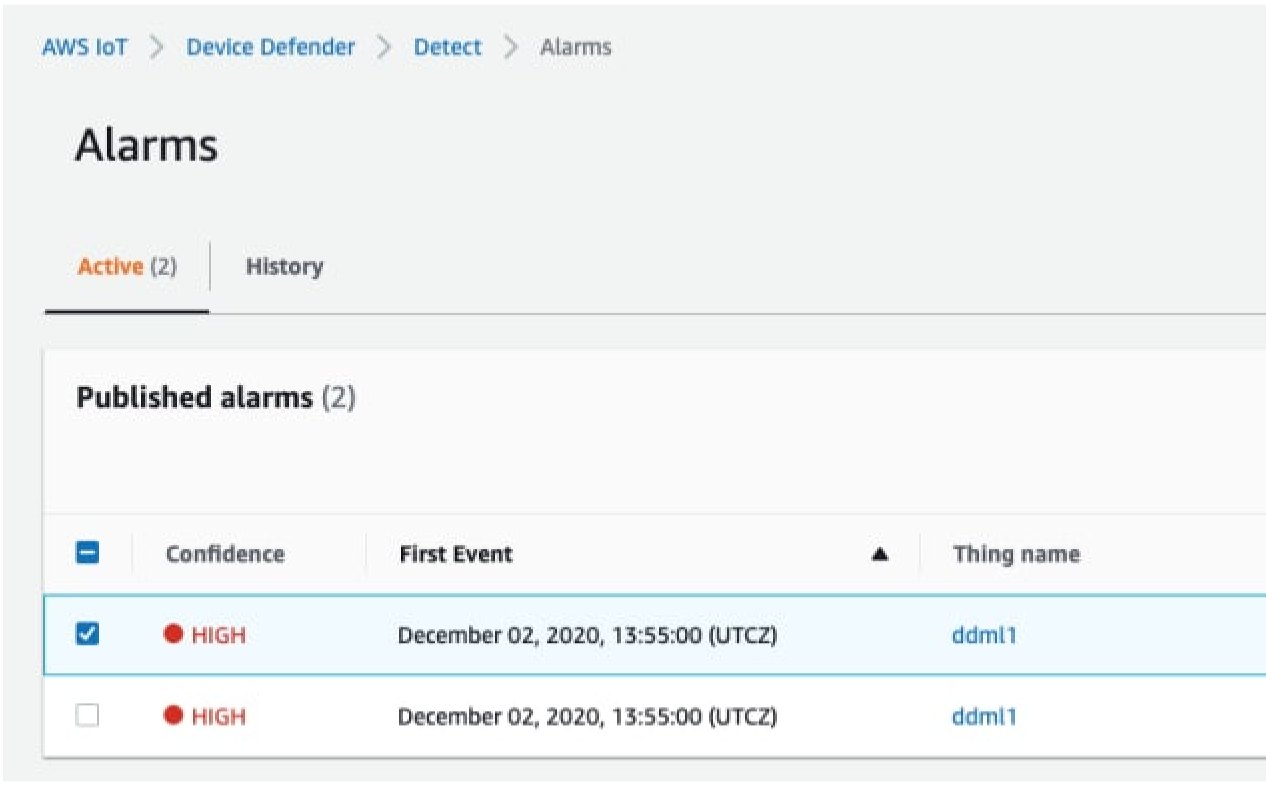
% of all training days required
100%

Note
If your model doesn't progress as expected, make sure your devices are meeting the Minimum requirements (p. 840).

Review your ML Detect alarms

After your ML models are built and ready for data inference, you can regularly view and investigate alarms that are identified by the models.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Alarms**.



AWS IoT > Device Defender > Detect > Alarms

Alarms

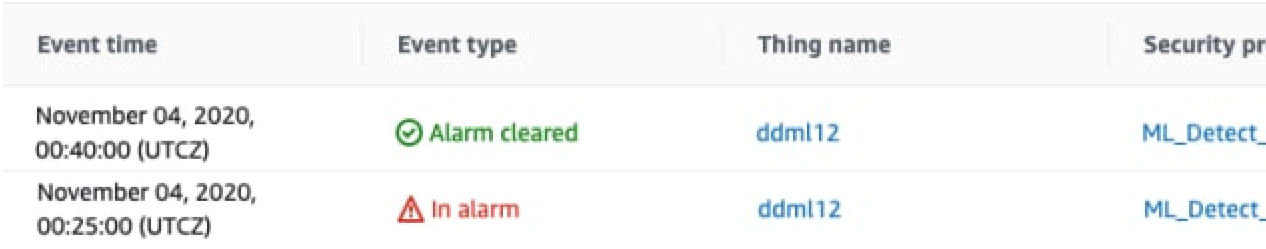
Active (2) | History

Published alarms (2)

	Confidence	First Event	Thing name
✓	● HIGH	December 02, 2020, 13:55:00 (UTCZ)	ddml1
	● HIGH	December 02, 2020, 13:55:00 (UTCZ)	ddml1

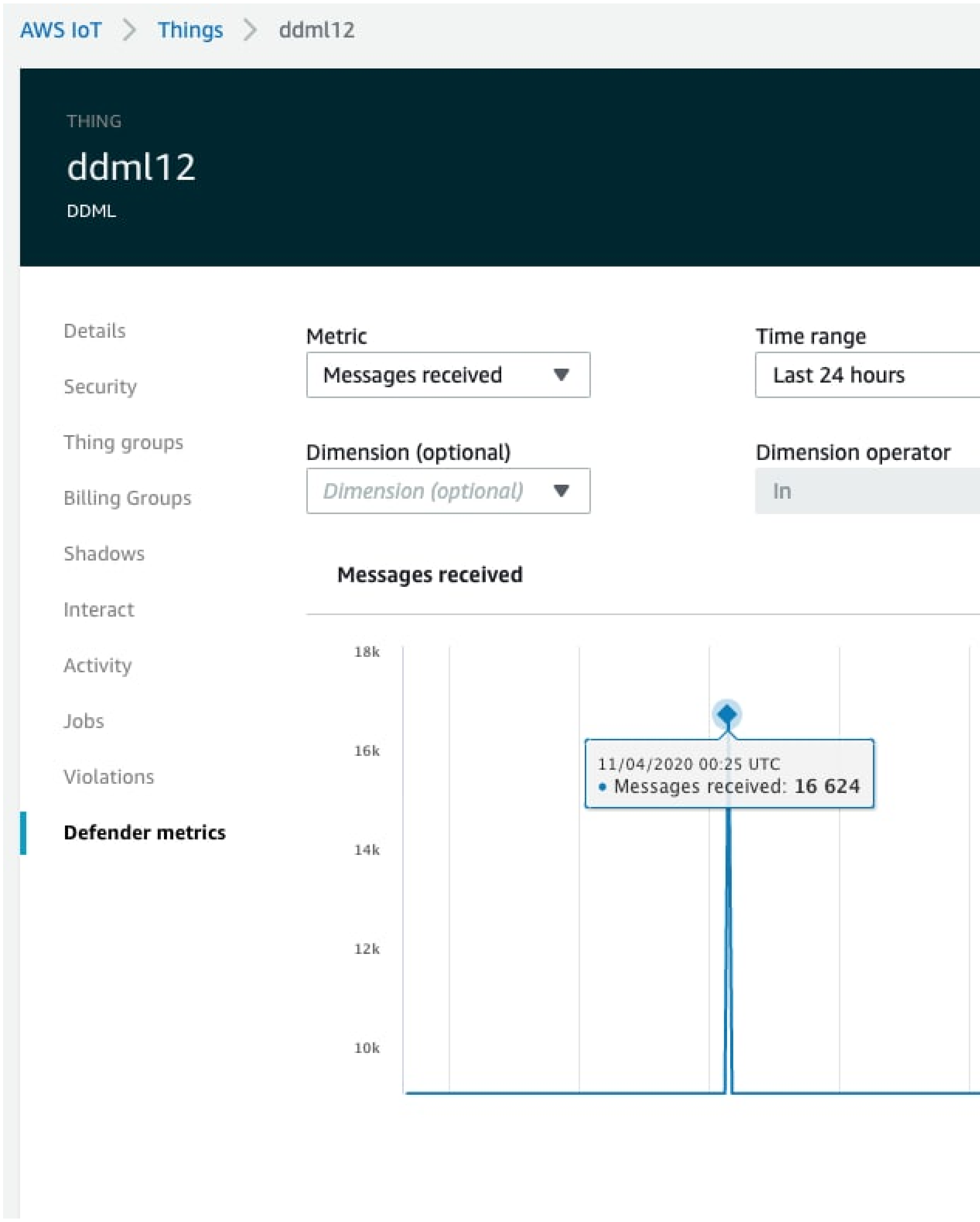
2. If you navigate to the **History** tab and scroll down, you can view details about your alarms and their state.

   Here, you can see thing `ddml12` received too many messages and cleared subsequently, which cleared the alarm state.



Event time	Event type	Thing name	Security pr
November 04, 2020, 00:40:00 (UTCZ)	Alarm cleared	ddml12	ML_Detect_
November 04, 2020, 00:25:00 (UTCZ)	In alarm	ddml12	ML_Detect_

   To dive deeper, choose the **Thing name** you'd like to see more details for. Here, we select `ddml12` and then navigate to **Defender metrics**. You can access the **Defender metrics graph** and perform your investigation on any thing in alarm from the **Active** tab. In this case, the graph shows a spike in messages received, which triggered the alarm. However, the alarm cleared subsequently.


AWS IoT > Things > ddml12

THING
ddml12
DDML

Details
Security
Thing groups
Billing Groups
Shadows
Interact
Activity
Jobs
Violations
Defender metrics

Metric: Messages received
Time range: Last 24 hours
Dimension (optional): *Dimension (optional)*
Dimension operator: In

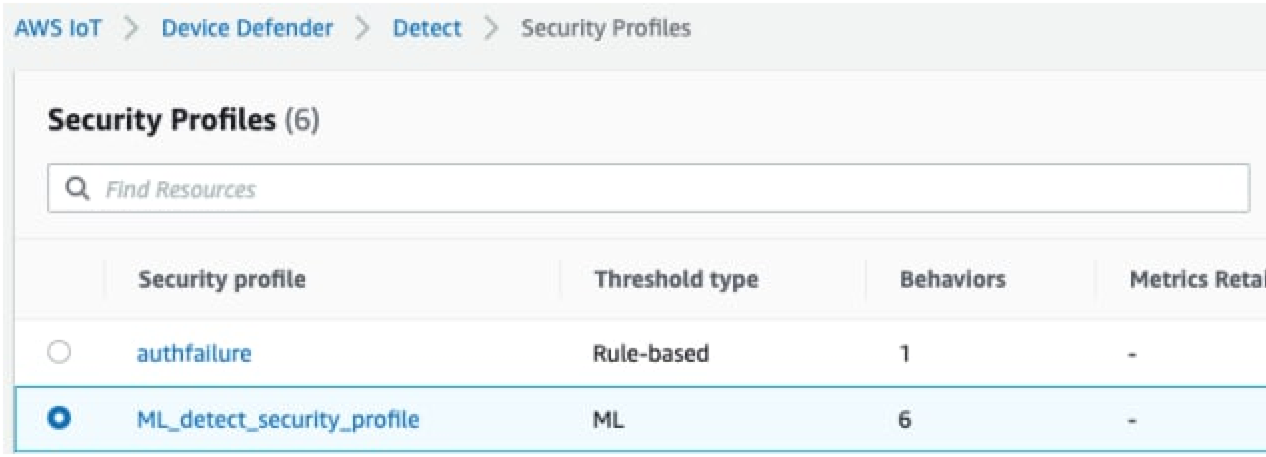
Messages received

11/04/2020 00:25 UTC
• Messages received: 16 624

Fine-tune your ML alarms

After your ML models are built, you can update your Security Profile's ML behavior settings by using the following steps.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.
2. On the **Security Profiles** page, select the check box next to the Security Profile you'd like to review. Then, choose **Actions**, **Edit**.



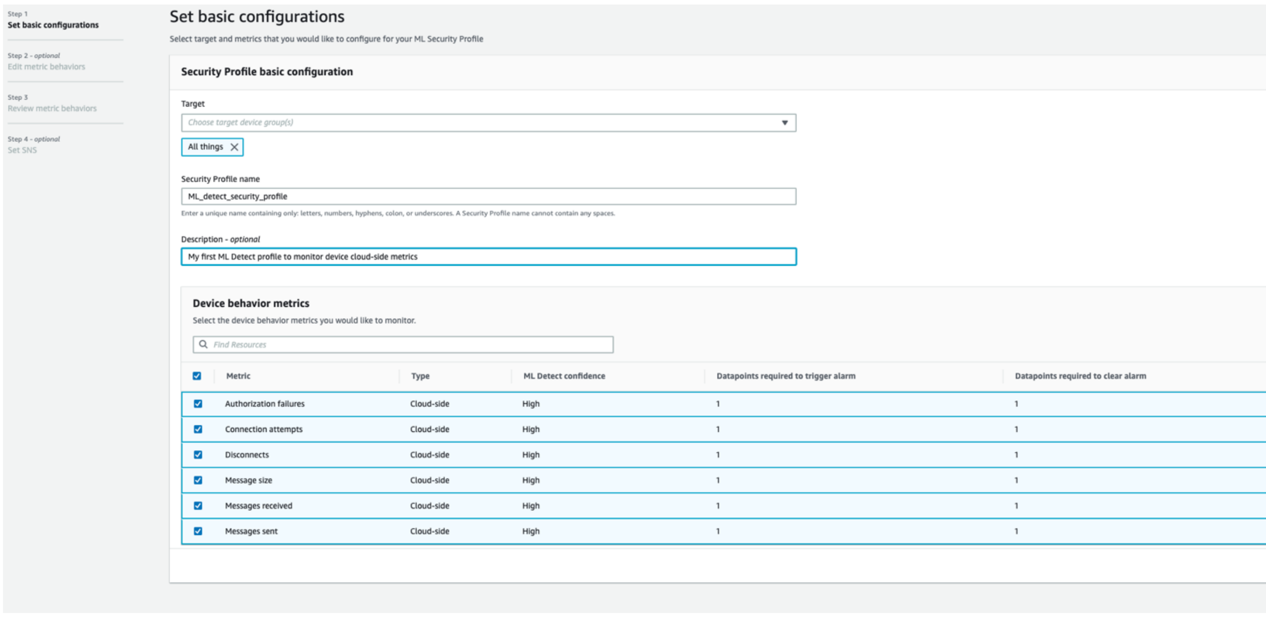
AWS IoT > Device Defender > Detect > Security Profiles

Security Profiles (6)

Find Resources

	Security profile	Threshold type	Behaviors	Metrics Retai
○	authfailure	Rule-based	1	-
●	ML_detect_security_profile	ML	6	-

3. Under **Set basic configurations**, you can change what metrics you want to monitor as well as attach them to other things or thing groups.



Step 1
Set basic configurations

Step 2 - *optional*
Edit metric behaviors

Step 3
Review metric behaviors

Step 4 - *optional*
Set SNS

Set basic configurations

Select target and metrics that you would like to configure for your ML Security Profile

Security Profile basic configuration

Target

Choose target device group(s)

All things

Security Profile name

ML_detect_security_profile

Enter a unique name containing only: letters, numbers, hyphens, colon, or underscores. A Security Profile name cannot contain any spaces.

Description - *optional*

My first ML Detect profile to monitor device cloud-side metrics

Device behavior metrics

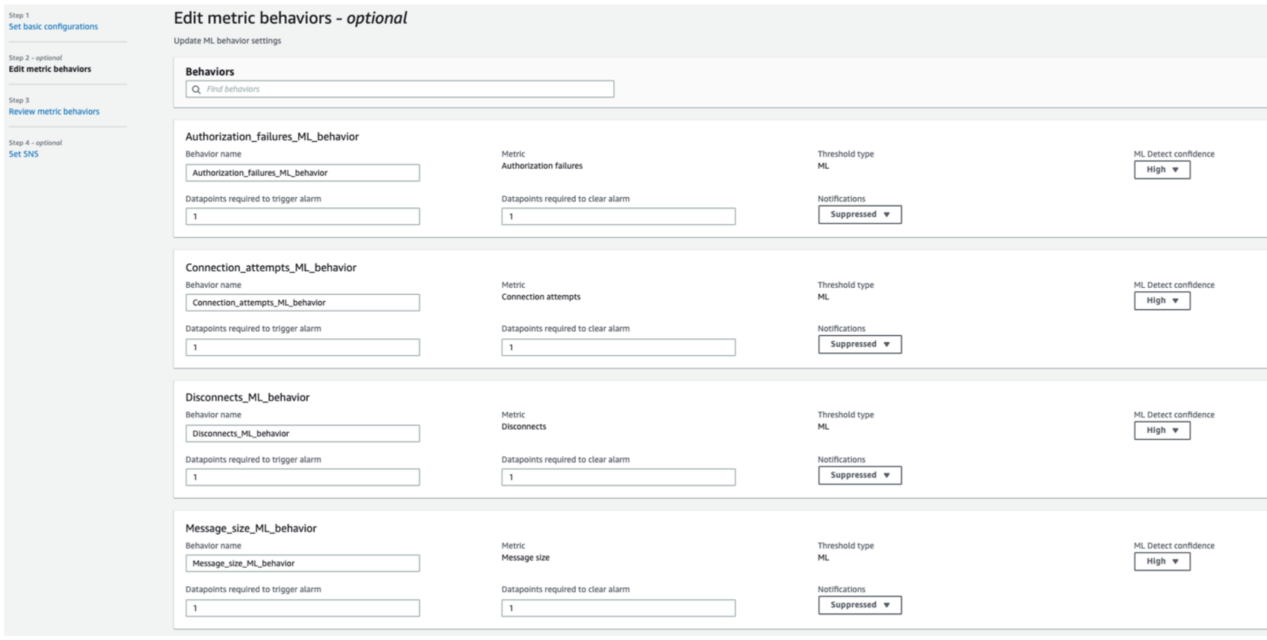
Select the device behavior metrics you would like to monitor.

Find Resources

✓	Metric	Type	ML Detect confidence	Datapoints required to trigger alarm	Datapoints required to clear alarm
✓	Authorization failures	Cloud-side	High	1	1
✓	Connection attempts	Cloud-side	High	1	1
✓	Disconnects	Cloud-side	High	1	1
✓	Message size	Cloud-side	High	1	1
✓	Messages received	Cloud-side	High	1	1
✓	Messages sent	Cloud-side	High	1	1

4. You can update any of the following by navigating to **Edit metric behaviors (optional)**.

   - Your ML model datapoints required to trigger alarm
   - Your ML model datapoints required to clear alarm
   - Your ML Detect confidence level
   - Your ML Detect notifications (for example, **On**, **Suppressed**)



Step 1
Set basic configurations

Step 2 - *optional*
Edit metric behaviors

Step 3
Review metric behaviors

Step 4 - *optional*
Set SNS

Edit metric behaviors - *optional*

Update ML behavior settings

Behaviors

Find behaviors

Authorization_failures_ML_behavior

Behavior name: Authorization_failures_ML_behavior
Metric: Authorization failures
Threshold type: ML
ML Detect confidence: High
Datapoints required to trigger alarm: 1
Datapoints required to clear alarm: 1
Notifications: Suppressed

Connection_attempts_ML_behavior

Behavior name: Connection_attempts_ML_behavior
Metric: Connection attempts
Threshold type: ML
ML Detect confidence: High
Datapoints required to trigger alarm: 1
Datapoints required to clear alarm: 1
Notifications: Suppressed

Disconnects_ML_behavior

Behavior name: Disconnects_ML_behavior
Metric: Disconnects
Threshold type: ML
ML Detect confidence: High
Datapoints required to trigger alarm: 1
Datapoints required to clear alarm: 1
Notifications: Suppressed

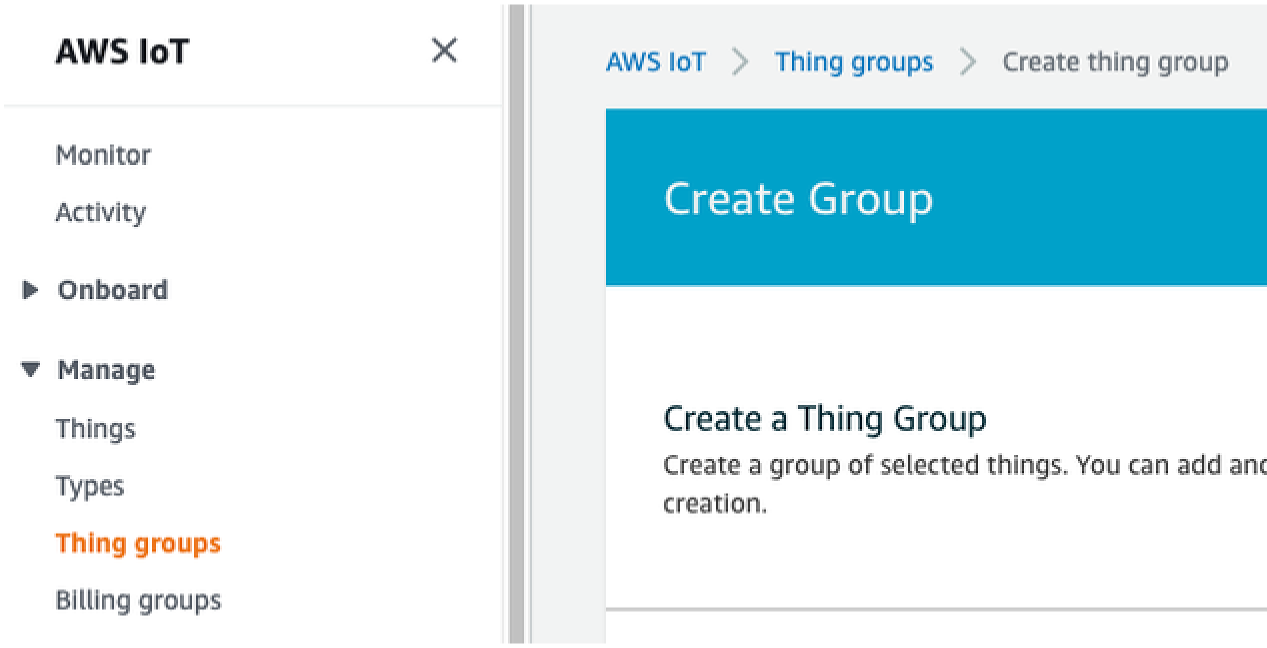
Message_size_ML_behavior

Behavior name: Message_size_ML_behavior
Metric: Message size
Threshold type: ML
ML Detect confidence: High
Datapoints required to trigger alarm: 1
Datapoints required to clear alarm: 1
Notifications: Suppressed

Mitigate identified device issues

1. *(Optional)* Before setting up quarantine mitigation actions, let's set up a quarantine group where we'll move the device in violation to. You can use any existing group if you have it.
2. Navigate to **Manage**, **Thing groups**, and then **Create Thing Group**. Name your thing group. For this tutorial, we'll name our thing group `Quarantine_group`. Under **Thing group**, **Security**, apply the following policy to the thing group.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iot:*",
      "Resource": "*",
    }
  ]
}
```



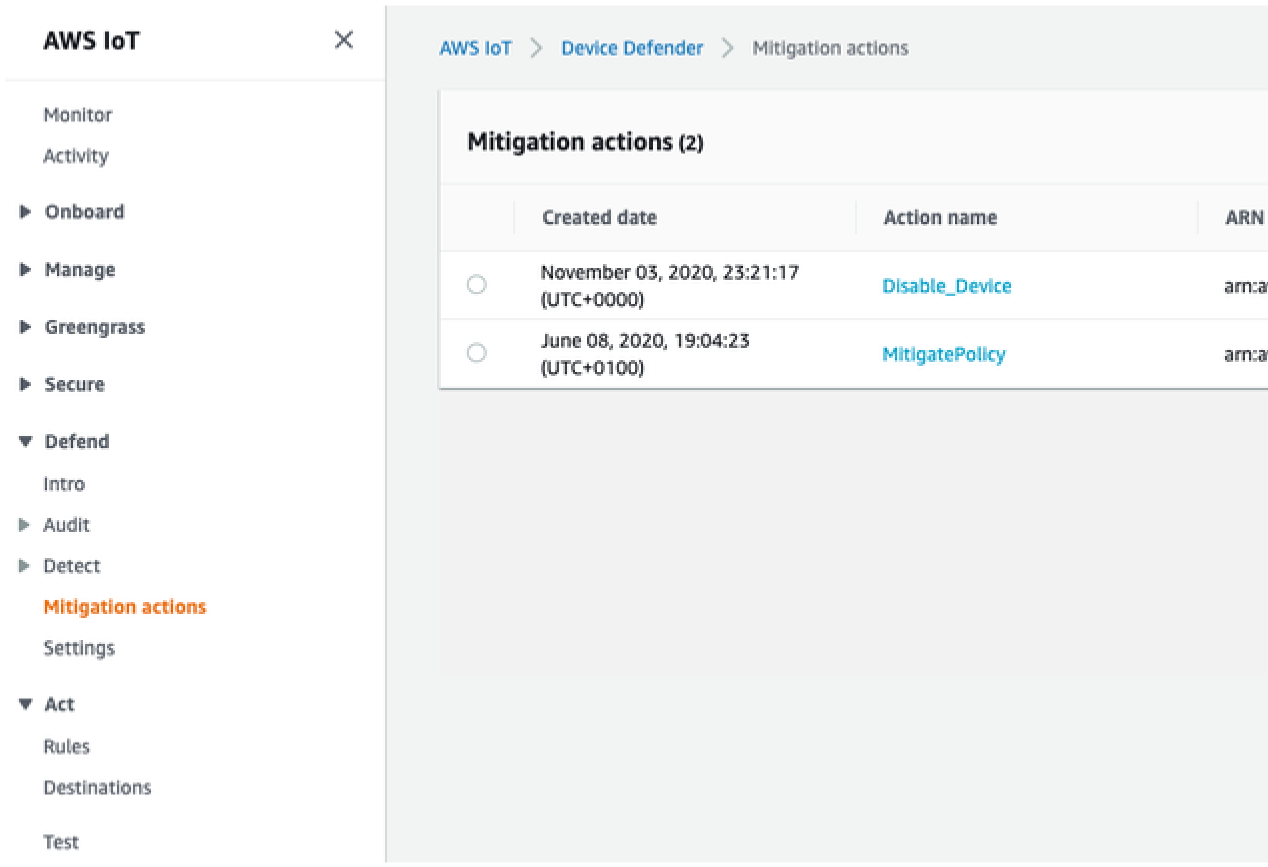
Create a Thing Group

Create a group of selected things. You can add anc
creation.

Choose **Create thing group** when done.

3. Now that we've created a thing group, let's create a mitigation action that move devices in alarm into `Quarantine_group`.

   Under **Defend**, **Mitigation actions**, choose **Create**.

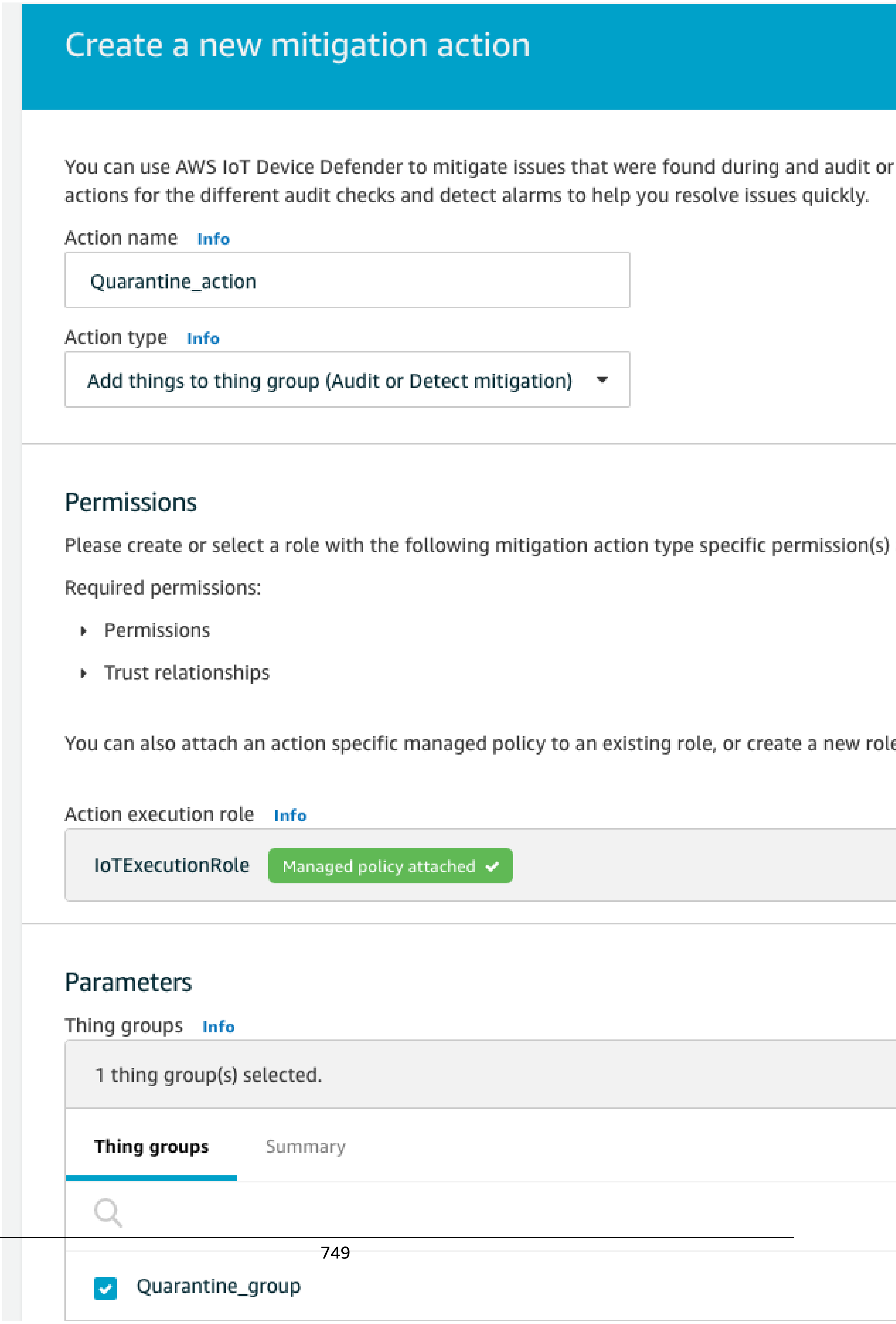


Mitigation actions (2)

	Created date	Action name	ARN
○	November 03, 2020, 23:21:17 (UTC+0000)	Disable_Device	arn:a
○	June 08, 2020, 19:04:23 (UTC+0100)	MitigatePolicy	arn:a

4. On the **Create a new mitigation action** screen, fill out the following.

   - **Action name**: Give your mitigation action a name, such as **`Quarantine_action`**.
   - **Action type**: Select the type of action. We'll select **Add things to thing group (Audit or Detect mitigation)**.
   - **Action execution role**: Create a role or select an existing role if you created one earlier.
   - **Parameters**: Select a thing group. We can use `Quarantine_group`, which we created earlier.



Create a new mitigation action

You can use AWS IoT Device Defender to mitigate issues that were found during and audit or
actions for the different audit checks and detect alarms to help you resolve issues quickly.

Action name Info

Quarantine_action

Action type Info

Add things to thing group (Audit or Detect mitigation)

Permissions

Please create or select a role with the following mitigation action type specific permission(s)

Required permissions:

- Permissions
- Trust relationships

You can also attach an action specific managed policy to an existing role, or create a new role

Action execution role Info

IoTExecutionRole Managed policy attached ✓

Parameters

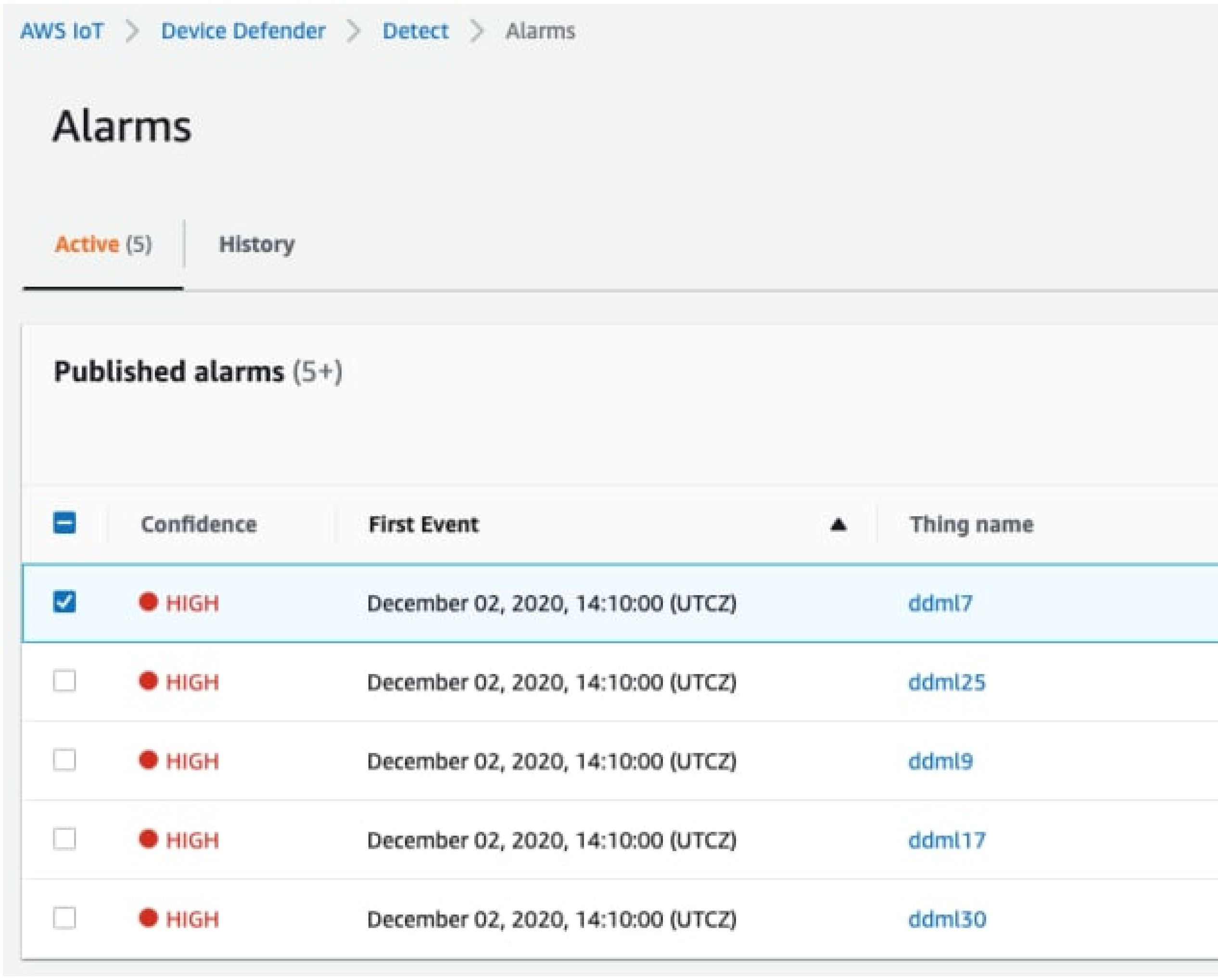
Thing groups Info

1 thing group(s) selected.

✓ Quarantine_group

Once you're done, choose **Save**. You'll now have a mitigation action that moves devices in alarm to a quarantine thing group and a mitigation action to isolate the device while you investigate.

5. Navigate to **Defender**, **Detect**, **Alarms**. You'll be able to see which devices are in alarm state under **Active**.



AWS IoT > Device Defender > Detect > Alarms

Alarms

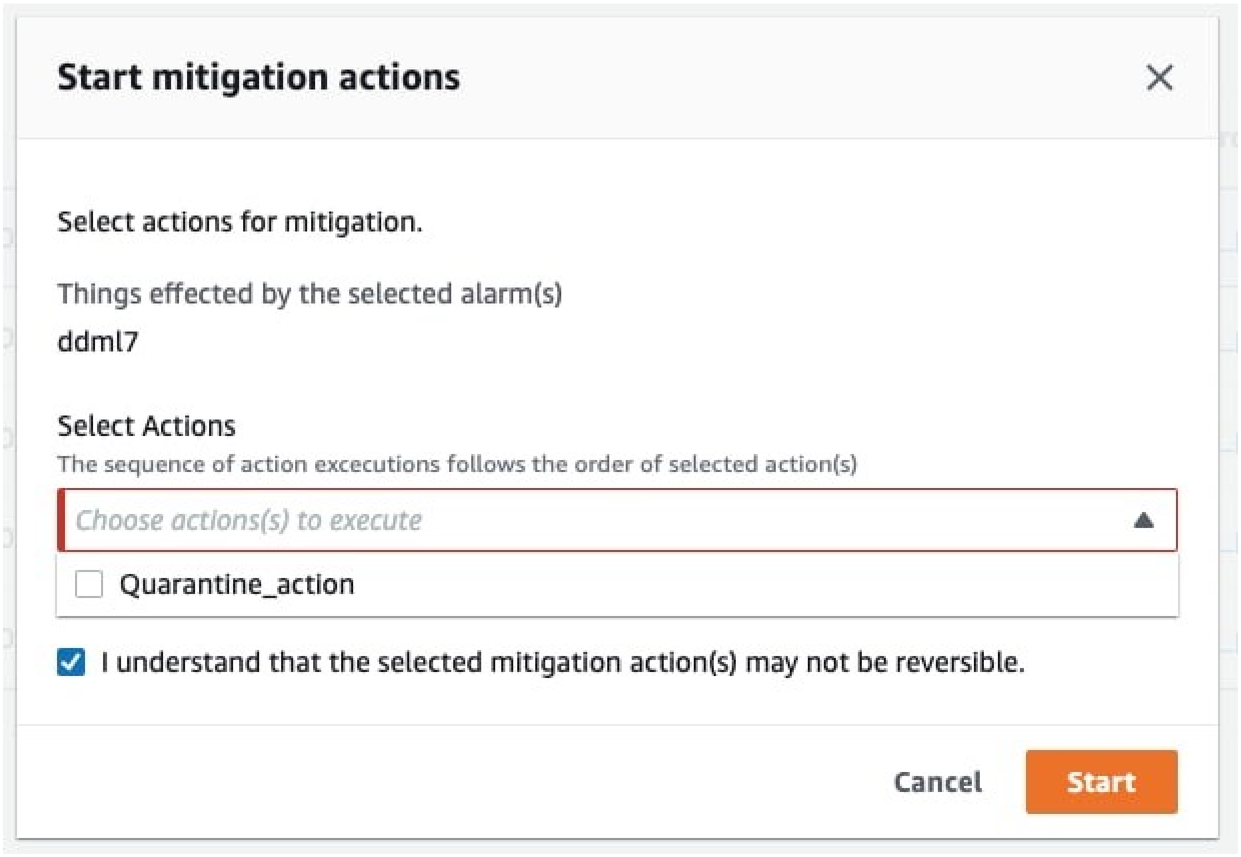
Active (5) | History

Published alarms (5+)

	Confidence	First Event	Thing name
✓	● HIGH	December 02, 2020, 14:10:00 (UTCZ)	ddml7
	● HIGH	December 02, 2020, 14:10:00 (UTCZ)	ddml25
	● HIGH	December 02, 2020, 14:10:00 (UTCZ)	ddml9
	● HIGH	December 02, 2020, 14:10:00 (UTCZ)	ddml17
	● HIGH	December 02, 2020, 14:10:00 (UTCZ)	ddml30

Select the device you want to move to the quarantine group and choose **Start Mitigation Actions**.

6. Under **Start mitigation actions**, **Start Actions** select the mitigation action you created earlier. For example, we'll choose `Quarantine_action`. Then choose **Start**.



Start mitigation actions

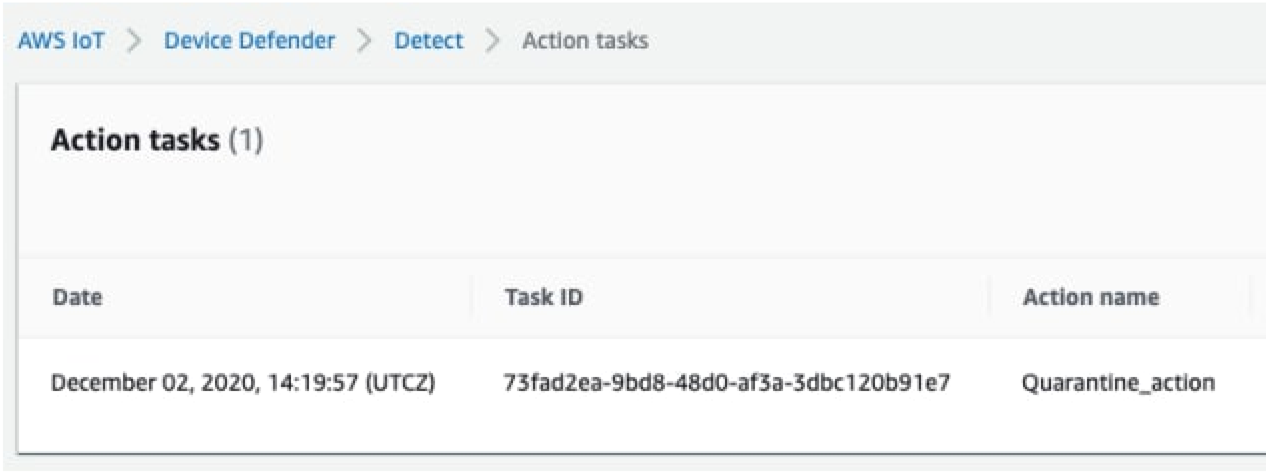
Select actions for mitigation.

Things effected by the selected alarm(s)
ddml7

Select Actions
The sequence of action excecutions follows the order of selected action(s)

I understand that the selected mitigation action(s) may not be reversible.

7. You'll be redirected to **Action tasks**.



Action tasks (1)

Date	Task ID	Action name
December 02, 2020, 14:19:57 (UTCZ)	73fad2ea-9bd8-48d0-af3a-3dbc120b91e7	Quarantine_action

The device is now isolated in **`Quarantine_group`** and you can investigate the root cause of the issue that set off the alarm. Once the investigation is complete, you can always move the device out of the thing group or take further actions.

How to use ML Detect on the CLI

The following shows you how to set up ML Detect using the CLI.

Tutorials

Enable ML Detect

The following procedure shows you how to enable ML Detect in the AWS CLI.

1. First, make sure your devices will create the minimum datapoints required as defined in ML Detect minimum requirements (p. 840) for ongoing training and refreshing of the model. For data collection to progress, ensure your things are in a thing group attached to a Security Profile.
2. First, create an ML Detect Security Profile by using the `create-security-profile` command. The following example creates a Security Profile named *`security-profile-for-smart-lights`* that checks for number of messages sent, number of authorization failures, number of connection attempts, and number of disconnects. The example uses `mlDetectionConfig` to establish that the metric will use the ML Detect model.

```
aws iot create-security-profile \
    --security-profile-name security-profile-for-smart-lights \
    --behaviors \
     '[{
      "name": "num-messages-sent-ml-behavior",
      "metric": "aws:num-messages-sent",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "MEDIUM"
          }
      },
      "suppressAlerts": true
  },
  {
      "name": "num-authorization-failures-ml-behavior",
      "metric": "aws:num-authorization-failures",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "MEDIUM"
          }
      },
      "suppressAlerts": true
  },
  {
      "name": "num-connection-attempts-ml-behavior",
      "metric": "aws:num-connection-attempts",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "MEDIUM"
          }
      },
      "suppressAlerts": true
  },
  {
      "name": "num-disconnects-ml-behavior",
      "metric": "aws:num-disconnects",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "MEDIUM"
          }
      },
      "suppressAlerts": true

  }]'
```

Output:

```
{
    "securityProfileName": "security-profile-for-smart-lights",
    "securityProfileArn": "arn:aws:iot:eu-west-1:123456789012:securityprofile/security-
profile-for-smart-lights"
  }
```

3. Next, associate your Security Profile with one or multiple thing groups. Use the `attach-security-profile` command to attach a thing group to your Security Profile. The following example associates a thing group named *`ML_Detect_beta_static_group`* with the *`security-profile-for-smart-lights`* Security Profile.

   ```
   aws iot attach-security-profile \
   --security-profile-name security-profile-for-smart-lights \
   --security-profile-target-arn arn:aws:iot:eu-
   west-1:123456789012:thinggroup/ML_Detect_beta_static_group
   ```

   Output:

   None.

4. After you've created your complete Security Profile, the ML model will begin training. The initial ML model training and building takes 14 days to complete. After 14 days, if there's anomalous activity on your device, you can expect to see alarms.

Monitor your ML model status

The following procedure shows you how to monitor your ML models in-progress training.

- Use the `get-behavior-model-training-summaries` command to view your ML model's progress. The following example gets the ML model training progress summary for the *`security-profile-for-smart-lights`* Security Profile. `modelStatus` will show you if a model has completed training or still pending build for a particular behavior.

  ```
  aws iot get-behavior-model-training-summaries \
     --security-profile-name security-profile-for-smart-lights
  ```

  Output:

  ```
  {
      "summaries": [
          {
              "securityProfileName": "security-profile-for-smart-lights",
              "behaviorName": "Messages_sent_ML_behavior",
              "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
              "modelStatus": "ACTIVE",
              "datapointsCollectionPercentage": 29.408,
              "lastModelRefreshDate": "2020-12-07T14:35:19.237000-08:00"
          },
          {
              "securityProfileName": "security-profile-for-smart-lights",
              "behaviorName": "Messages_received_ML_behavior",
              "modelStatus": "PENDING_BUILD",
  ```

```
            "datapointsCollectionPercentage": 0.0
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Authorization_failures_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 35.464,
            "lastModelRefreshDate": "2020-12-07T14:29:44.396000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Message_size_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 29.332,
            "lastModelRefreshDate": "2020-12-07T14:30:44.113000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Connection_attempts_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 32.891999999999996,
            "lastModelRefreshDate": "2020-12-07T14:29:43.121000-08:00"
        },
        {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "Disconnects_ML_behavior",
            "trainingDataCollectionStartDate": "2020-11-30T14:00:00-08:00",
            "modelStatus": "ACTIVE",
            "datapointsCollectionPercentage": 35.46,
            "lastModelRefreshDate": "2020-12-07T14:29:55.556000-08:00"
        }
    ]
}
```

Note
If your model doesn't progress as expected, make sure your devices are meeting the ML Detect minimum requirements (p. 840).

Review your ML Detect alarms

After your ML models are built and ready for data evaluations, you can regularly view any alarms that are inferred by the models. The following procedure shows you how to view your alarms in the AWS CLI.

- To see all active alarms, use the `list-active-violations` command.

```
aws iot list-active-violations \
--max-results 2
```

Output:

```
{
    "activeViolations": []
}
```

Alternatively, you can view all violations discovered during a given time period by using the `list-violation-events` command. The following example lists violation events from September 22, 2020 5:42:13 GMT to October 26, 2020 5:42:13 GMT.

```
aws iot list-violation-events \
    --start-time 1599500533 \
    --end-time 1600796533 \
    --max-results 2
```

Output:

```
{
    "violationEvents": [
        {
            "violationId": "1448be98c09c3d4ab7cb9b6f3ece65d6",
            "thingName": "lightbulb-1",
            "securityProfileName": "security-profile-for-smart-lights",
            "behavior": {
                "name": "LowConfidence_MladBehavior_MessagesSent",
                "metric": "aws:num-messages-sent",
                "criteria": {
                    "mlDetectionConfig": {
                        "confidenceLevel": "MEDIUM"
                    }
                },
                "suppressAlerts": false
            },
            "violationEventType": "alarm-invalidated",
            "violationEventTime": 1600780245.29
        },
        {
            "violationId": "df4537569ef23efb1c029a433ae84b52",
            "thingName": "lightbulb-2",
            "securityProfileName": "security-profile-for-smart-lights",
            "behavior": {
                "name": "LowConfidence_MladBehavior_MessagesSent",
                "metric": "aws:num-messages-sent",
                "criteria": {
                    "mlDetectionConfig": {
                        "confidenceLevel": "MEDIUM"
                    }
                },
                "suppressAlerts": false
            },
            "violationEventType": "alarm-invalidated",
            "violationEventTime": 1600780245.281
        }
    ],
    "nextToken":
 "Amo6XIUrsOohsojuIG6TuwSR3X9iUvH2OCksBZg6bed2j21VSnD1uP1pflxKX1+a3cvBRSosIB0xFv40kM6RYBknZ/
vxabMe/ZW31Ps/WiZHlr9Wg7R7eEGli59IJ/U0iBQ1McP/ht0E2XA2TTIvYeMmKQQPsRj/
eoV9j7P/wveu7skNGepU/mvpV0O2Ap7hnV5U+Prx/9+iJA/341va
+pQww7jpUeHmJN9Hw4MqW0ysw0Ry3w38hOQWEpz2xwFWAxAARxeIxCxt5c37RK/lRZBlhYqoB
+w2PZ74730h8pICGY4gktJxkwHyyRabpSM/G/f5DFrD9O5v8idkTZzBxW2jrbzSUIdafPtsZHL/
yAMKr3HAKtaABz2nTsOBNre7X2d/jIjjarhon0Dh9l+8I9Y5Ey
+DIFBcqFTvhibKAafQt3gs6CUiqHdWiCenfJyb8whmDE2qxvdxGElGmRb
+k6kuN5jrZxxw95gzfYDgRHv11iEn8h1qZLD0czkIFBpMppHj9cetHPvM
+qffXGAzKi8tL6eQuCdMLXmVE3jbqcJcjk9ItnaYJi5zKDz9FVbrz9qZZPtZJFHp"
}
```

Fine-tune your ML alarms

Once your ML models are built and ready for data evaluations, you can update your Security Profile's ML behavior settings to change the configuration. The following procedure shows you how to update your Security Profile's ML behavior settings in the AWS CLI.

- Use the `update-security-profile` command to change your Security Profile's ML behavior settings. The following example updates the *`security-profile-for-smart-lights`* Security Profile's behaviors by changing the `confidenceLevel` of a few of the behaviors and unsuppresses notifications.

```
aws iot update-security-profile \
    --security-profile-name security-profile-for-smart-lights \
    --behaviors
     '[{
      "name": "num-messages-sent-ml-behavior",
      "metric": "aws:num-messages-sent",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
  },
  {
      "name": "num-authorization-failures-ml-behavior",
      "metric": "aws:num-authorization-failures",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
  },
  {
      "name": "num-connection-attempts-ml-behavior",
      "metric": "aws:num-connection-attempts",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "HIGH"
          }
      },
      "suppressAlerts": false
  },
  {
      "name": "num-disconnects-ml-behavior",
      "metric": "aws:num-disconnects",
      "criteria": {
          "mlDetectionConfig": {
              "confidenceLevel" : "LOW"
          }
      },
      "suppressAlerts": true

  }]'
```

Output:

```
 {
    "securityProfileName": "security-profile-for-smart-lights",
    "securityProfileArn": "arn:aws:iot:eu-west-1:123456789012:securityprofile/security-
profile-for-smart-lights",
    "behaviors": [
        {
            "name": "num-messages-sent-ml-behavior",
            "metric": "aws:num-messages-sent",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            }
        },
        {
            "name": "num-authorization-failures-ml-behavior",
            "metric": "aws:num-authorization-failures",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            }
        },
        {
            "name": "num-connection-attempts-ml-behavior",
            "metric": "aws:num-connection-attempts",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "HIGH"
                }
            },
            "suppressAlerts": false
        },
        {
            "name": "num-disconnects-ml-behavior",
            "metric": "aws:num-disconnects",
            "criteria": {
                "mlDetectionConfig": {
                    "confidenceLevel": "LOW"
                }
            },
            "suppressAlerts": true
        }
    ],
    "version": 2,
    "creationDate": 1600799559.249,
    "lastModifiedDate": 1600800516.856
}
```

Mitigate identified device issues

1. Use the `create-thing-group` command to create a thing group for the mitigation action. In the following example we create a thing group called **ThingGroupForDetectMitigationAction**

```
aws iot create-thing-group —thing-group-name ThingGroupForDetectMitigationAction
```

   Output:

```
{
 "thingGroupName": "ThingGroupForDetectMitigationAction",
 "thingGroupArn": "arn:aws:iot:us-
east-1:123456789012:thinggroup/ThingGroupForDetectMitigationAction",
 "thingGroupId": "4139cd61-10fa-4c40-b867-0fc6209dca4d"
}
```

2. Next, use the `create-mitigation-action` command to create a mitigation action. In the following example, we create a mitigation action called **detect_mitigation_action** with the ARN of

the IAM role that is used to apply the mitigation action. We also define the type of action and the parameters for that action. In this case, our mitigation will move things to our previously created thing group called **ThingGroupForDetectMitigationAction**.

```
aws iot create-mitigation-action --action-name detect_mitigation_action \
--role-arn arn:aws:iam::123456789012:role/MitigationActionValidRole \
--action-params
'{
     "addThingsToThingGroupParams": {
         "thingGroupNames": ["ThingGroupForDetectMitigationAction"],
         "overrideDynamicGroups": false
     }
 }'
```

Output:

```
{
 "actionArn": "arn:aws:iot:us-
east-1:123456789012:mitigationaction/detect_mitigation_action",
 "actionId": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3"
}
```

3. Use the `start-detect-mitigation-actions-task` command to start your mitigation actions task. `task-id`, `target` and `actions` are required parameters.

```
aws iot start-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction \
    --target '{ "violationIds" : [ "violationId-1", "violationId-2" ] }' \
    --actions "detect_mitigation_action" \
    --include-only-active-violations \
    --include-suppressed-alerts \
```

Output:

```
{
    "taskId": "taskIdForMitigationAction"
}
```

4. (Optional) Use the `describe-detect-mitigation-actions-task` command to get information about a mitigation action task.

```
aws iot describe-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction
```

Output:

```
{
    "taskSummary": {
        "taskId": "taskIdForMitigationAction",
        "taskStatus": "SUCCESSFUL",
        "taskStartTime": 1609988361.224,
        "taskEndTime": 1609988362.281,
        "target": {
            "securityProfileName": "security-profile-for-smart-lights",
            "behaviorName": "num-messages-sent-ml-behavior"
        },
        "violationEventOccurrenceRange": {
            "startTime": 1609986633.0,
            "endTime": 1609987833.0
        },
        "onlyActiveViolationsIncluded": true,
        "suppressedAlertsIncluded": true,
        "actionsDefinition": [
            {
                "name": "detect_mitigation_action",
                "id": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3",
                "roleArn": "arn:aws:iam::123456789012:role/MitigatioActionValidRole",
                "actionParams": {
                    "addThingsToThingGroupParams": {
                        "thingGroupNames": [
                            "ThingGroupForDetectMitigationAction"
                        ],
                        "overrideDynamicGroups": false
                    }
                }
            }
        ],
        "taskStatistics": {
            "actionsExecuted": 0,
            "actionsSkipped": 0,
            "actionsFailed": 0
        }
    }
}
```

5. (Optional) To view mitigation actions executions included in a task, use the `list-detect-mitigation-actions-executions` command.

```
aws iot list-detect-mitigation-actions-executions \
    --task-id taskIdForMitigationAction \
    --max-items 5 \
    --page-size 4
```

Output:

```
{
    "actionsExecutions": [
        {
            "taskId": "e56ee95e - f4e7 - 459 c - b60a - 2701784290 af",
            "violationId": "214_fe0d92d21ee8112a6cf1724049d80",
            "actionName": "underTest_MAThingGroup71232127",
            "thingName": "cancelDetectMitigationActionsTaskd143821b",
            "executionStartDate": "Thu Jan 07 18: 35: 21 UTC 2021",
            "executionEndDate": "Thu Jan 07 18: 35: 21 UTC 2021",
            "status": "SUCCESSFUL",
        }
    ]
}
```

6. (Optional) To get a list of your mitigation actions tasks, use the `list-detect-mitigation-actions-tasks` command.

```
aws iot list-detect-mitigation-actions-tasks \
    --start-time 1609985315 \
    --end-time 1609988915 \
    --max-items 5 \
    --page-size 4
```

Output:

```
{
    "tasks": [
        {
            "taskId": "taskIdForMitigationAction",
            "taskStatus": "SUCCESSFUL",
            "taskStartTime": 1609988361.224,
            "taskEndTime": 1609988362.281,
            "target": {
                "securityProfileName": "security-profile-for-smart-lights",
                "behaviorName": "num-messages-sent-ml-behavior"
            },
            "violationEventOccurrenceRange": {
                "startTime": 1609986633.0,
                "endTime": 1609987833.0
            },
            "onlyActiveViolationsIncluded": true,
            "suppressedAlertsIncluded": true,
            "actionsDefinition": [
                {
                    "name": "detect_mitigation_action",
                    "id": "5939e3a0-bf4c-44bb-a547-1ab59ffe67c3",
                    "roleArn": "arn:aws:iam::123456789012:role/
MitigatioActionValidRole",
                    "actionParams": {
                        "addThingsToThingGroupParams": {
                            "thingGroupNames": [
                                "ThingGroupForDetectMitigationAction"
                            ],
                            "overrideDynamicGroups": false
                        }
                    }
                }
            ],
            "taskStatistics": {
                "actionsExecuted": 0,
                "actionsSkipped": 0,
                "actionsFailed": 0
            }
        }
    ]
}
```

7. (Optional) To cancel a mitigation actions task, use the `cancel-detect-mitigation-actions-task` command.

```
aws iot cancel-detect-mitigation-actions-task \
    --task-id taskIdForMitigationAction
```

Output:

None.

Customize when and how you view AWS IoT Device Defender audit results

AWS IoT Device Defender audit provides periodic security checks to confirm AWS IoT devices and resources are following best practices. For each check, the audit results are categorized as compliant or non-compliant, where non-compliance results in console warning icons. To reduce noise from repeating known issues, the audit finding suppression feature allows you to temporarily silence these non-compliance notifications.

You can suppress select audit checks for a specific resource or account for a predetermined time period. An audit check result that has been suppressed is categorized as a suppressed finding, separate from the compliant and non-compliant categories. This new category doesn't trigger an alarm like a non-compliant result. This allows you to reduce non-compliance notification disturbances during known maintenance periods or until an update is scheduled to be completed.

Getting started

The following sections detail how you can use audit finding suppressions to suppress a `Device certificate expiring` check in the console and CLI. If you'd like to follow either of the demonstrations, you must first create two expiring certificates for Device Defender to detect.

Use the following to create your certificates.

- Create and register a CA certificate (p. 207)
- Create a client certificate using your CA certificate. In step 3, set your `days` parameter to **1**.

If you're using the CLI to create your certificates, enter the following command.

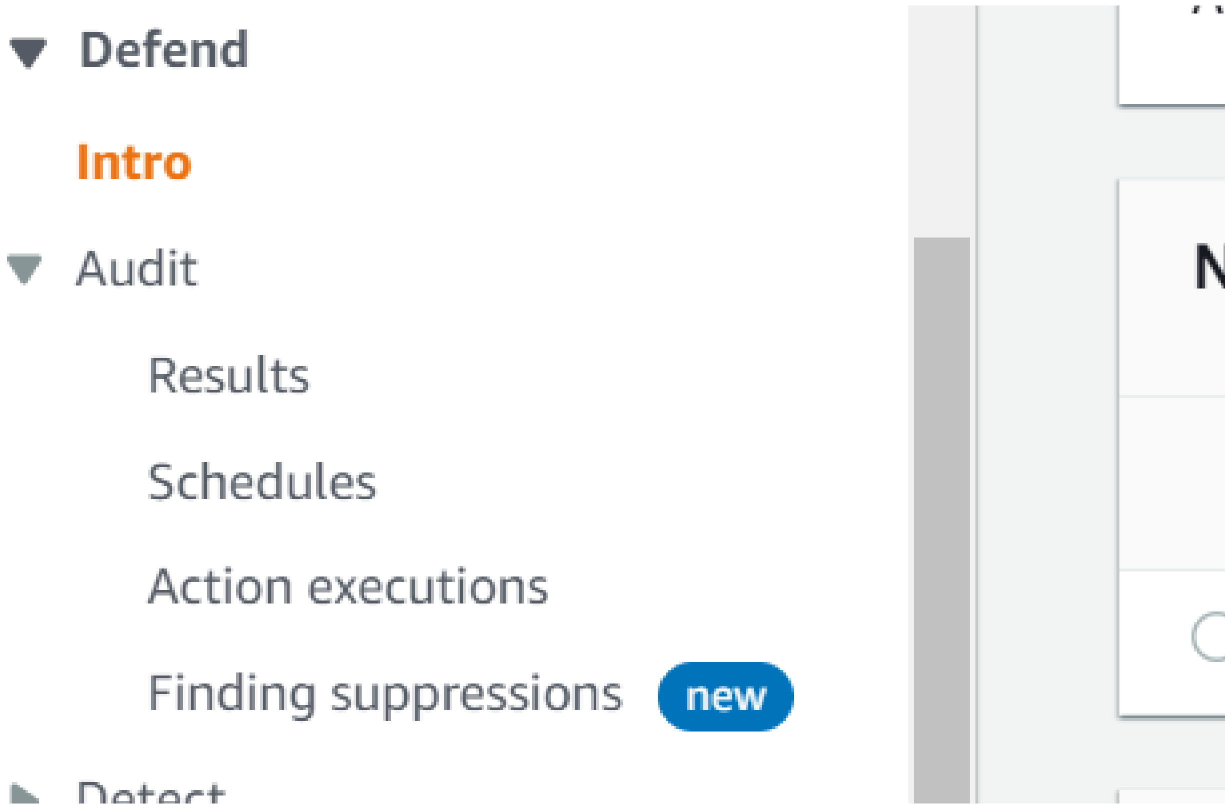
```
openssl x509 -req \
    -in device_cert_csr_filename \
    -CA root_ca_pem_filename \
    -CAkey root_ca_key_filename \
    -CAcreateserial \
    -out device_cert_pem_filename \
    -days 1 -sha256
```

Customize your audit findings in the console

The following walkthrough uses an account with two expired device certificates that trigger a non-compliant audit check. In this scenario, we want to disable the warning because our developers are testing a new feature that'll address the problem. We create an audit finding suppression for each certificate to stop the audit result from being non-compliant for the next week.

1. We will first run an on-demand audit to show that the expired device certificate check is non-compliant.

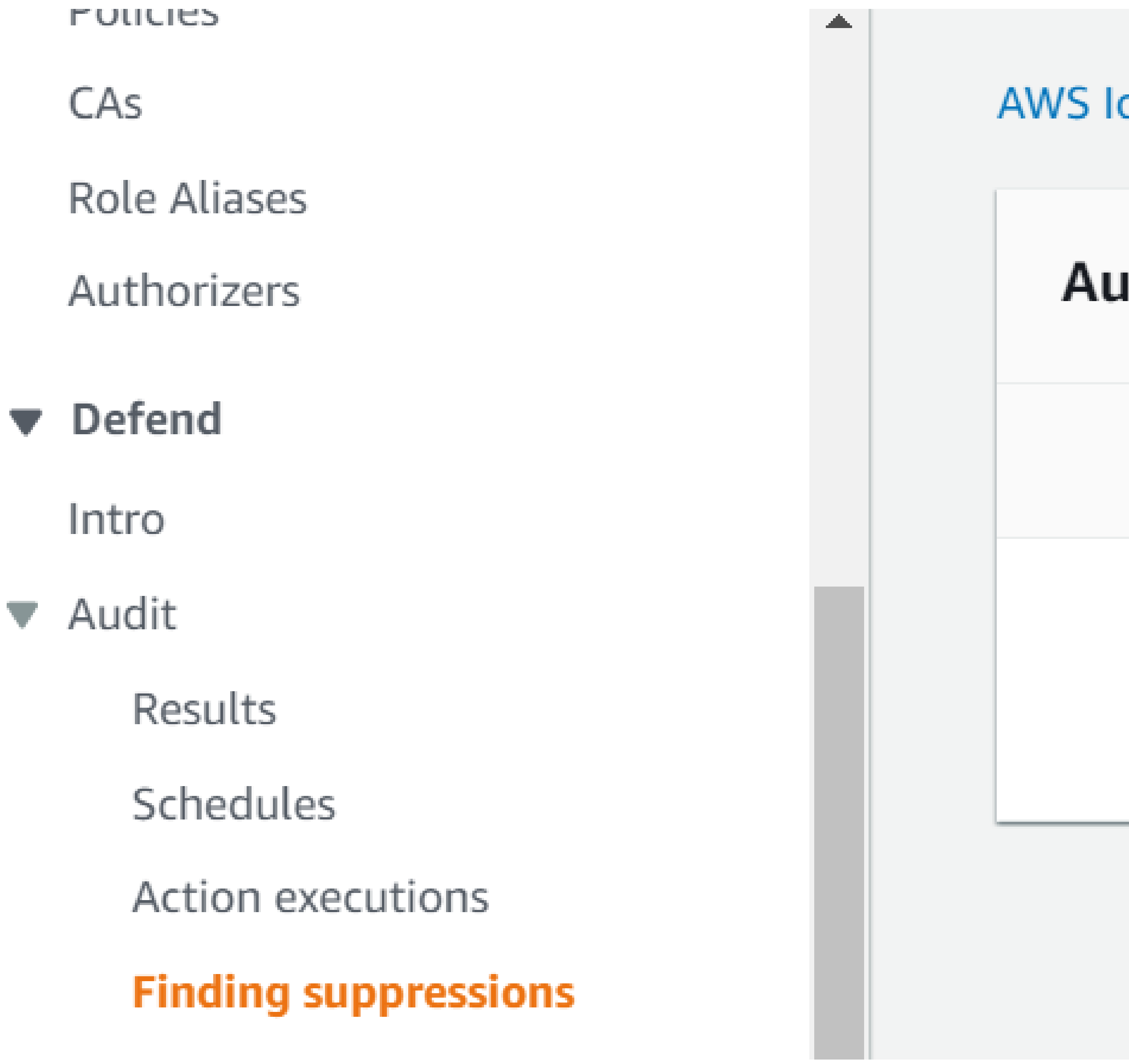
   From the AWS IoT console, choose **Defend** from the left sidebar, then **Audit**, and then **Results**. On the **Audit Results** page, choose **Create**. The **Create a new audit** window opens. Choose **Create**.



From the on-demand audit results, we can see that "Device certificate expiring" is non-compliant for two resources.

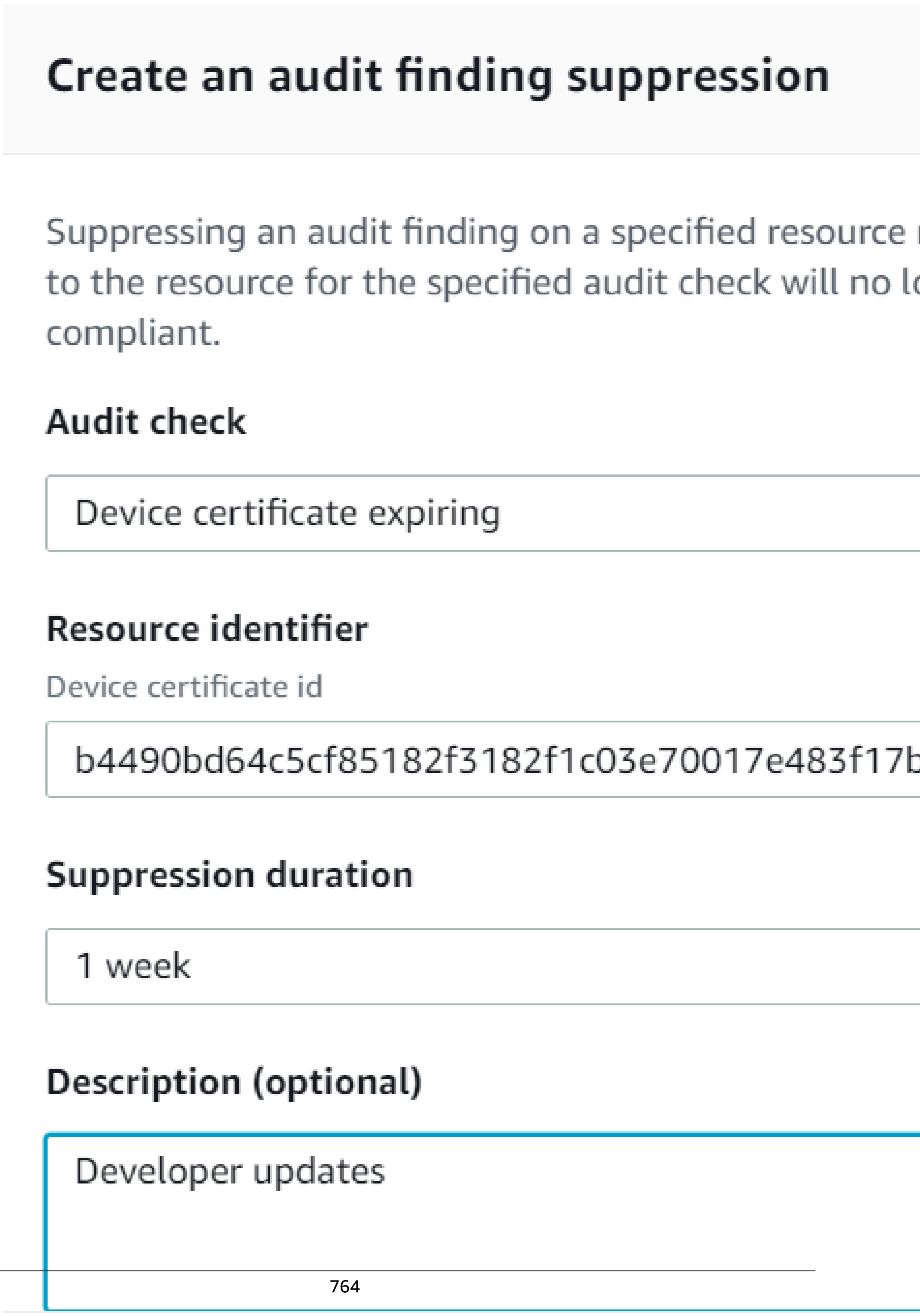
2. Now, we'd like to disable the "Device certificate expiring" non-compliant check warning because our developers are testing new features that will fix the warning.

   From the left sidebar under **Defend**, choose **Audit**, and then choose **Finding suppressions**. On the **Audit finding suppressions** page, choose **Create**.



3. On the **Create an audit finding suppression** window, we need to fill out the following.

   - **Audit check**: We select `Device certificate expiring`, because that is the audit check we'd like to suppress.
   - **Resource identifier**: We input the device certificate ID of one of the certificates we'd like to suppress audit findings for.
   - **Suppression duration**: We select `1 week`, because that's how long we'd like to suppress the `Device certificate expiring` audit check for.
   - **Description (optional)**: We add a note that describes why we're suppressing this audit finding.



Create an audit finding suppression

Suppressing an audit finding on a specified resource
to the resource for the specified audit check will no l
compliant.

Audit check

Device certificate expiring

Resource identifier

Device certificate id

b4490bd64c5cf85182f3182f1c03e70017e483f17

Suppression duration

1 week

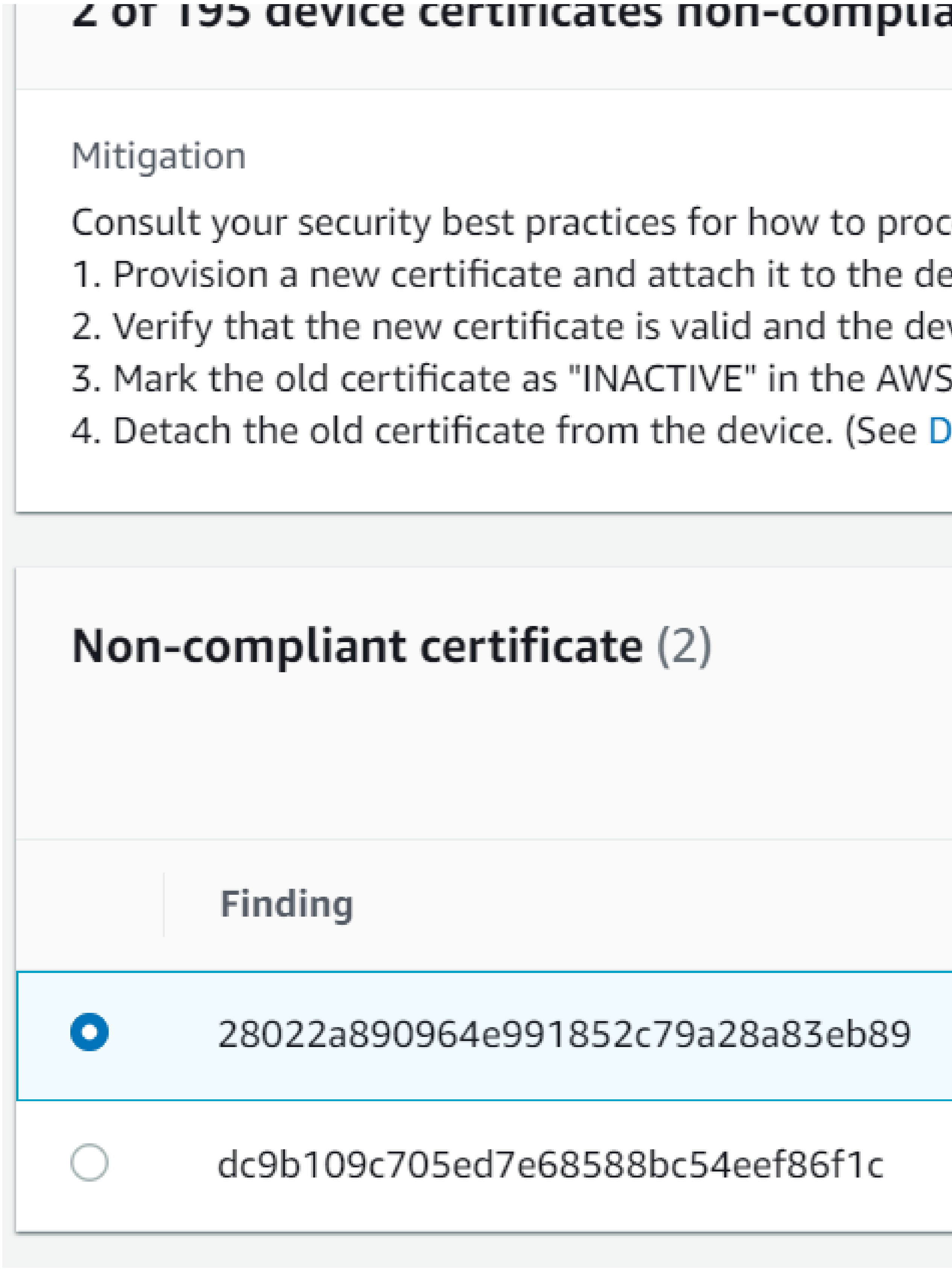
Description (optional)

Developer updates

After we've filled out the fields, choose **Create**. We see a success banner after the audit finding suppression has been created.

4. We've suppressed an audit finding for one of the certificates and now we need to suppress the audit finding for the second certificate. We could use the same suppression method that we used in step 3, but we will be using a different method for demonstration purposes.

   From the left sidebar under **Defend**, choose **Audit**, and then choose **Results**. On the **Audit results** page, choose the audit with the non-compliant resource. Then, select the resource under **Non-compliant checks**. In our case, we select "Device certificate expiring".
5. On the **Device certificate expiring** page, under **Non-compliant policy** choose the option button next to the finding that needs to be suppressed. Next, choose the **Actions** dropdown menu, and then choose the duration for which you'd like finding to be suppressed. In our case, we choose `1 week` as we did for the other certificate. On the **Confirm suppression** window, choose **Enable suppression**.



2 of 195 device certificates non-complia

Mitigation

Consult your security best practices for how to proc
1. Provision a new certificate and attach it to the de
2. Verify that the new certificate is valid and the dev
3. Mark the old certificate as "INACTIVE" in the AWS
4. Detach the old certificate from the device. (See D

Non-compliant certificate (2)

	Finding
●	28022a890964e991852c79a28a83eb89
○	dc9b109c705ed7e68588bc54eef86f1c

We see a success banner after the audit finding suppression has been created. Now, both audit findings have been suppressed for 1 week while our developers work on a solution to address the warning.

Customize your audit findings in the CLI

The following walkthrough uses an account with an expired device certificate that trigger a non-compliant audit check. In this scenario, we want to disable the warning because our developers are testing a new feature that'll address the problem. We create an audit finding suppression for the certificate to stop the audit result from being non-compliant for the next week.

We use the following CLI commands.

- create-audit-suppression
- describe-audit-suppression
- update-audit-suppression
- delete-audit-suppression
- list-audit-suppressions

1. Use the following command to enable the audit.

   ```
   aws iot update-account-audit-configuration \
      --audit-check-configurations "{\"DEVICE_CERTIFICATE_EXPIRING_CHECK\":{\"enabled
   \":true}}"
   ```

   Output:

   None.
2. Use the following command to run an on-demand audit that targets the `DEVICE_CERTIFICATE_EXPIRING_CHECK` audit check.

   ```
   aws iot start-on-demand-audit-task \
       --target-check-names DEVICE_CERTIFICATE_EXPIRING_CHECK
   ```

   Output:

   ```
   {
       "taskId": "787ed873b69cb4d6cdbae6ddd06996c5"
   }
   ```
3. Use the describe-account-audit-configuration command to describe the audit configuration. We want to confirm that we've turned on the audit check for `DEVICE_CERTIFICATE_EXPIRING_CHECK`.

   ```
   aws iot describe-account-audit-configuration
   ```

   Output:

   ```

   {
       "roleArn": "arn:aws:iam::<accountid>:role/service-role/project",
       "auditNotificationTargetConfigurations": {
           "SNS": {
   ```

```
            "targetArn": "arn:aws:sns:us-east-1:<accountid>:project_sns",
            "roleArn": "arn:aws:iam::<accountid>:role/service-role/project",
            "enabled": true
        }
    },
    "auditCheckConfigurations": {
        "AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK": {
            "enabled": false
        },
        "CA_CERTIFICATE_EXPIRING_CHECK": {
            "enabled": false
        },
        "CA_CERTIFICATE_KEY_QUALITY_CHECK": {
            "enabled": false
        },
        "CONFLICTING_CLIENT_IDS_CHECK": {
            "enabled": false
        },
        "DEVICE_CERTIFICATE_EXPIRING_CHECK": {
            "enabled": true
        },
        "DEVICE_CERTIFICATE_KEY_QUALITY_CHECK": {
            "enabled": false
        },
        "DEVICE_CERTIFICATE_SHARED_CHECK": {
            "enabled": false
        },
        "IOT_POLICY_OVERLY_PERMISSIVE_CHECK": {
            "enabled": true
        },
        "IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK": {
            "enabled": false
        },
        "IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK": {
            "enabled": false
        },
        "LOGGING_DISABLED_CHECK": {
            "enabled": false
        },
        "REVOKED_CA_CERTIFICATE_STILL_ACTIVE_CHECK": {
            "enabled": false
        },
        "REVOKED_DEVICE_CERTIFICATE_STILL_ACTIVE_CHECK": {
            "enabled": false
        },
        "UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK": {
            "enabled": false
        }
    }
}
```

`DEVICE_CERTIFICATE_EXPIRING_CHECK` should have a value of `true`.

4. Use the list-audit-task command to identify the completed audit tasks.

```
aws iot list-audit-tasks \
    --task-status "COMPLETED" \
    --start-time 2020-07-31 \
    --end-time 2020-08-01
```

Output:

```
{
    "tasks": [
        {
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "taskStatus": "COMPLETED",
            "taskType": "SCHEDULED_AUDIT_TASK"
        }
    ]
}
```

The `taskId` of the audit you ran in step 1 should have a `taskStatus` of `COMPLETED`.

5. Use the describe-audit-task command to get details about the completed audit using the `taskId` output from the previous step. This command lists details about your audit.

```
aws iot describe-audit-task \
    --task-id "787ed873b69cb4d6cdbae6ddd06996c5"
```

Output:

```
{
    "taskStatus": "COMPLETED",
    "taskType": "SCHEDULED_AUDIT_TASK",
    "taskStartTime": 1596168096.157,
    "taskStatistics": {
        "totalChecks": 1,
        "inProgressChecks": 0,
        "waitingForDataCollectionChecks": 0,
        "compliantChecks": 0,
        "nonCompliantChecks": 1,
        "failedChecks": 0,
        "canceledChecks": 0
    },
    "scheduledAuditName": "AWSIoTDeviceDefenderDailyAudit",
    "auditDetails": {
        "DEVICE_CERTIFICATE_EXPIRING_CHECK": {
            "checkRunStatus": "COMPLETED_NON_COMPLIANT",
            "checkCompliant": false,
            "totalResourcesCount": 195,
            "nonCompliantResourcesCount": 2
        }
    }
}
```

6. Use the list-audit-findings command to find the non-compliant certificate ID so that we can suspend the audit alerts for this resource.

```
aws iot list-audit-findings \
    --start-time 2020-07-31 \
    --end-time 2020-08-01
```

Output:

```
{
    "findings": [
        {
            "findingId": "296ccd39f806bf9d8f8de20d0ceb33a1",
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "taskStartTime": 1596168096.157,
            "findingTime": 1596168096.651,
            "severity": "MEDIUM",
            "nonCompliantResource": {
                "resourceType": "DEVICE_CERTIFICATE",
                "resourceIdentifier": {
                    "deviceCertificateId": "b4490<shortened>"
                },
                "additionalInfo": {
                "EXPIRATION_TIME": "1582862626000"
                }
            },
            "reasonForNonCompliance": "Certificate is past its expiration.",
            "reasonForNonComplianceCode": "CERTIFICATE_PAST_EXPIRATION",
            "isSuppressed": false
        },
        {
            "findingId": "37ecb79b7afb53deb328ec78e647631c",
            "taskId": "787ed873b69cb4d6cdbae6ddd06996c5",
            "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "taskStartTime": 1596168096.157,
            "findingTime": 1596168096.651,
            "severity": "MEDIUM",
            "nonCompliantResource": {
                "resourceType": "DEVICE_CERTIFICATE",
                "resourceIdentifier": {
                    "deviceCertificateId": "c7691<shortened>"
                },
                "additionalInfo": {
                "EXPIRATION_TIME": "1583424717000"
                }
            },
            "reasonForNonCompliance": "Certificate is past its expiration.",
            "reasonForNonComplianceCode": "CERTIFICATE_PAST_EXPIRATION",
            "isSuppressed": false
        }
    ]
}
```

7. Use the create-audit-suppression command to suppress notifications for the `DEVICE_CERTIFICATE_EXPIRING_CHECK` audit check for a device certificate with the id *`c7691e<shortened>`* until *`2020-08-20`*.

```
aws iot create-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId="c7691e<shortened>" \
    --no-suppress-indefinitely \
    --expiration-date 2020-08-20
```

8. Use the list-audit-suppression command to confirm the audit suppression setting and get details about the suppression.

```
aws iot list-audit-suppressions
```

Output:

```
{
    "suppressions": [
        {
        "checkName": "DEVICE_CERTIFICATE_EXPIRING_CHECK",
            "resourceIdentifier": {
                "deviceCertificateId": "c7691e<shortened>"
            },
        "expirationDate": 1597881600.0,
        "suppressIndefinitely": false
        }
    ]
}
```

9. The update-audit-suppression command can be used to update the audit finding suppression. The example below updates the `expiration-date` to `08/21/20`.

```
aws iot update-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId=c7691e<shortened> \
    --no-suppress-indefinitely \
    --expiration-date 2020-08-21
```

10. The delete-audit-suppression command can be used to remove an audit finding suppression.

```
aws iot delete-audit-suppression \
    --check-name DEVICE_CERTIFICATE_EXPIRING_CHECK \
    --resource-identifier deviceCertificateId="c7691e<shortened>"
```

    To confirm deletion, use the list-audit-suppressions command.

```
aws iot list-audit-suppressions
```

    Output:

```
{
 "suppressions": []
}
```

In this tutorial, we showed you how to suppress a `Device certificate expiring` check in the console and CLI. For more information about audit finding suppressions, see Audit finding suppressions (p. 823)

Audit

An AWS IoT Device Defender audit looks at account- and device-related settings and policies to ensure security measures are in place. An audit can help you detect any drifts from security best practices or access policies (for example, multiple devices using the same identity, or overly permissive policies that allow one device to read and update data for many other devices). You can run audits as needed (*on-demand audits*) or schedule them to be run periodically (*scheduled audits*).

An AWS IoT Device Defender audit runs a set of predefined checks for common IoT security best practices and device vulnerabilities. Examples of predefined checks include policies that grant permission to read or update data on multiple devices, devices that share an identity (X.509 certificate), or certificates that are expiring or have been revoked but are still active.

Issue severity

Issue severity indicates the level of concern associated with each identified instance of noncompliance and the recommended time to remediation.

Critical

Non compliant audit checks with this severity identify issues that require urgent attention. Critical issues often allow bad actors with little sophistication and no insider knowledge or special credentials to easily gain access to or control of your assets.

High

Non compliant audit checks with this severity require urgent investigation and remediation planning after critical issues are addressed. Like critical issues, high severity issues often provide bad actors with access to or control of your assets. However, high severity issues are often more difficult to exploit. They might require special tools, insider knowledge, or specific setups.

Medium

Non compliant audit checks with this severity present issues that need attention as part of your continuous security posture maintenance. Medium severity issues might cause negative operational impact, such as unplanned outages due to malfunction of security controls. These issues might also provide bad actors with limited access to or control of your assets, or might facilitate parts of their malicious actions.

Low

Non compliant audit checks with this severity often indicate security best practices were overlooked or bypassed. Although they might not cause an immediate security impact on their own, these lapses can be exploited by bad actors. Like medium severity issues, low severity issues require attention as part of your continuous security posture maintenance.

Next steps

To understand the types of audit checks that can be performed, see Audit checks (p. 772). For information about service quotas that apply to audits, see Service Quotas.

Audit checks

Note
When you enable a check, data collection starts immediately. If there is a large amount of data in your account to collect, results of the check might not be available for some time after you enabled it.

The following audit checks are supported:

- CA certificate revoked but device certificates still active (p. 772)
- Device certificate shared (p. 773)
- Device certificate key quality (p. 774)
- CA certificate key quality (p. 775)
- Unauthenticated Cognito role overly permissive (p. 776)
- Authenticated Cognito role overly permissive (p. 782)
- AWS IoT policies overly permissive (p. 788)
- Role alias overly permissive (p. 792)
- Role alias allows access to unused services (p. 793)
- CA certificate expiring (p. 793)
- Conflicting MQTT client IDs (p. 794)
- Device certificate expiring (p. 795)
- Revoked device certificate still active (p. 796)
- Logging disabled (p. 796)

CA certificate revoked but device certificates still active

A CA certificate was revoked, but is still active in AWS IoT.

This check appears as `REVOKED_CA_CERT_CHECK` in the CLI and API.

Severity: **Critical**

Details

A CA certificate is marked as revoked in the certificate revocation list maintained by the issuing authority, but is still marked as ACTIVE or PENDING_TRANSFER in AWS IoT.

The following reason codes are returned when this check finds a noncompliant CA certificate:

- CERTIFICATE_REVOKED_BY_ISSUER

Why it matters

A revoked CA certificate should no longer be used to sign device certificates. It might have been revoked because it was compromised. Newly added devices with certificates signed using this CA certificate might pose a security threat.

How to fix it

1. Use UpdateCACertificate to mark the CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
    - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.
    - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

    For more information, see Mitigation actions (p. 877).
2. Review the device certificate registration activity for the time after the CA certificate was revoked and consider revoking any device certificates that might have been issued with it during this time. You can use ListCertificatesByCA to list the device certificates signed by the CA certificate and UpdateCertificate to revoke a device certificate.

Device certificate shared

Multiple, concurrent connections use the same X.509 certificate to authenticate with AWS IoT.

This check appears as `DEVICE_CERTIFICATE_SHARED_CHECK` in the CLI and API.

Severity: **Critical**

Details

When performed as part of an on-demand audit, this check looks at the certificates and client IDs that were used by devices to connect during the 31 days before the start of the audit up to 2 hours before the check is run. For scheduled audits, this check looks at data from 2 hours before the last time the audit was run to 2 hours before the time this instance of the audit started. If you have taken steps to mitigate this condition during the time checked, note when the concurrent connections were made to determine if the problem persists.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_SHARED_BY_MULTIPLE_DEVICES

In addition, the findings returned by this check include the ID of the shared certificate, the IDs of the clients using the certificate to connect, and the connect/disconnect times. Most recent results are listed first.

Why it matters

Each device should have a unique certificate to authenticate with AWS IoT. When multiple devices use the same certificate, this might indicate that a device has been compromised. Its identity might have been cloned to further compromise the system.

How to fix it

Verify that the device certificate has not been compromised. If it has, follow your security best practices to mitigate the situation.

If you use the same certificate on multiple devices, you might want to:

1. Provision new, unique certificates and attach them to each device.
2. Verify that the new certificates are valid and the devices can use them to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to do the following:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).
4. Detach the old certificate from each of the devices.

Device certificate key quality

AWS IoT customers often rely on TLS mutual authentication using X.509 certificates for authenticating to AWS IoT message broker. These certificates and their certificate authority certificates must be registered in their AWS IoT account before they are used. AWS IoT performs basic sanity checks on these certificates when they are registered. These checks include:

- They must be in a valid format.
- They must be signed by a registered certificate authority.
- They must still be within their validity period (in other words, they haven't expired).
- Their cryptographic key sizes must meet a minimum required size (for RSA keys, they must be 2048 bits or larger).

This audit check provides the following additional tests of the quality of your cryptographic key:

- CVE-2008-0166 – Check whether the key was generated using OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on a Debian-based operating system. Those versions of OpenSSL use a random number generator that generates predictable numbers, making it easier for remote attackers to conduct brute force guessing attacks against cryptographic keys.
- CVE-2017-15361 – Check whether the key was generated by the Infineon RSA library 1.02.013 in Infineon Trusted Platform Module (TPM) firmware, such as versions before 0000000000000422 – 4.34, before 000000000000062b – 6.43, and before 0000000000008521 – 133.33. That library mishandles RSA key generation, making it easier for attackers to defeat some cryptographic protection mechanisms through targeted attacks. Examples of affected technologies include BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5) PGP key generation, and the Cached User Data encryption feature in Chrome OS.

AWS IoT Device Defender reports certificates as noncompliant if they fail these tests.

This check appears as `DEVICE_CERTIFICATE_KEY_QUALITY_CHECK` in the CLI and API.

Severity: **Critical**

Details

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_KEY_VULNERABILITY_CVE-2017-15361
- CERTIFICATE_KEY_VULNERABILITY_CVE-2008-0166

Why it matters

When a device uses a vulnerable certificate, attackers can more easily compromise that device.

How to fix it

Update your device certificates to replace those with known vulnerabilities.

If you are using the same certificate on multiple devices, you might want to:

1. Provision new, unique certificates and attach them to each device.
2. Verify that the new certificates are valid and the devices can use them to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).
4. Detach the old certificate from each of the devices.

CA certificate key quality

AWS IoT customers often rely on TLS mutual authentication using X.509 certificates for authenticating to AWS IoT message broker. These certificates and their certificate authority certificates must be registered in their AWS IoT account before they are used. AWS IoT performs basic sanity checks on these certificates when they are registered, including:

- The certificates are in a valid format.
- The certificates are within their validity period (in other words, not expired).
- Their cryptographic key sizes meet a minimum required size (for RSA keys, they must be 2048 bits or larger).

This audit check provides the following additional tests of the quality of your cryptographic key:

- CVE-2008-0166 – Check whether the key was generated using OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on a Debian-based operating system. Those versions of OpenSSL use a random number generator that generates predictable numbers, making it easier for remote attackers to conduct brute force guessing attacks against cryptographic keys.
- CVE-2017-15361 – Check whether the key was generated by the Infineon RSA library 1.02.013 in Infineon Trusted Platform Module (TPM) firmware, such as versions before 0000000000000422 – 4.34, before 000000000000062b – 6.43, and before 0000000000008521 – 133.33. That library mishandles RSA key generation, making it easier for attackers to defeat some cryptographic protection mechanisms through targeted attacks. Examples of affected technologies include BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5) PGP key generation, and the Cached User Data encryption feature in Chrome OS.

AWS IoT Device Defender reports certificates as noncompliant if they fail these tests.

This check appears as `CA_CERTIFICATE_KEY_QUALITY_CHECK` in the CLI and API.

Severity: **Critical**

Details

This check applies to CA certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant certificate:

- CERTIFICATE_KEY_VULNERABILITY_CVE-2017-15361
- CERTIFICATE_KEY_VULNERABILITY_CVE-2008-0166

Why it matters

Newly added devices signed using this CA certificate might pose a security threat.

How to fix it

1. Use UpdateCACertificate to mark the CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).
2. Review the device certificate registration activity for the time after the CA certificate was revoked and consider revoking any device certificates that might have been issued with it during this time. (Use ListCertificatesByCA to list the device certificates signed by the CA certificate and UpdateCertificate to revoke a device certificate.)

Unauthenticated Cognito role overly permissive

A policy attached to an unauthenticated Amazon Cognito identity pool role is considered too permissive because it grants permission to perform any of the following AWS IoT actions:

- Manage or modify things.
- Read thing administrative data.
- Manage non-thing related data or resources.

Or, because it grants permission to perform the following AWS IoT actions on a broad set of devices:

- Use MQTT to connect, publish, or subscribe to reserved topics (including shadow or job execution data).
- Use API commands to read or modify shadow or job execution data.

In general, devices that connect using an unauthenticated Amazon Cognito identity pool role should have only limited permission to publish and subscribe to thing-specific MQTT topics or use the API commands to read and modify thing-specific data related to shadow or job execution data.

This check appears as `UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

For this check, AWS IoT Device Defender audits all Amazon Cognito identity pools that have been used to connect to the AWS IoT message broker during the 31 days before the audit execution. All Amazon Cognito identity pools from which either an authenticated or unauthenticated Amazon Cognito identity connected are included in the audit.

The following reason codes are returned when this check finds a noncompliant unauthenticated Amazon Cognito identity pool role:

- ALLOWS_ACCESS_TO_IOT_ADMIN_ACTIONS
- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

Because unauthenticated identities are never authenticated by the user, they pose a much greater risk than authenticated Amazon Cognito identities. If an unauthenticated identity is compromised, it can use administrative actions to modify account settings, delete resources, or gain access to sensitive data. Or, with broad access to device settings, it can access or modify shadows and jobs for all devices in your account. A guest user might use the permissions to compromise your entire fleet or launch a DDOS attack with messages.

How to fix it

A policy attached to an unauthenticated Amazon Cognito identity pool role should grant only those permissions required for a device to do its job. We recommend the following steps:

1. Create a new compliant role.
2. Create a Amazon Cognito identity pool and attach the compliant role to it.
3. Verify that your identities can access AWS IoT using the new pool.
4. After verification is complete, attach the compliant role to the Amazon Cognito identity pool that was flagged as noncompliant.

You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Manage or modify things

The following AWS IoT API actions are used to manage or modify things. Permission to perform these actions should not be granted to devices that connect through an unauthenticated Amazon Cognito identity pool.

- `AddThingToThingGroup`
- `AttachThingPrincipal`
- `CreateThing`
- `DeleteThing`
- `DetachThingPrincipal`
- `ListThings`
- `ListThingsInThingGroup`
- `RegisterThing`
- `RemoveThingFromThingGroup`
- `UpdateThing`
- `UpdateThingGroupsForThing`

Any role that grants permission to perform these actions on even a single resource is considered noncompliant.

Read thing administrative data

The following AWS IoT API actions are used to read or modify thing data. Devices that connect through an unauthenticated Amazon Cognito identity pool should not be given permission to perform these actions.

- `DescribeThing`
- `ListJobExecutionsForThing`
- `ListThingGroupsForThing`
- `ListThingPrincipals`

Example

- noncompliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing"
      ]
    }
  ]
}
```

This allows the device to perform the specified actions even though it is granted for one thing only.

Manage non-things

Devices that connect through an unauthenticated Amazon Cognito identity pool should not be given permission to perform AWS IoT API actions other than those discussed in these sections. You can manage your account with an application that connects through an unauthenticated Amazon Cognito identity pool by creating a separate identity pool not used by devices.

Subscribe/publish to MQTT topics

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

```
arn:aws:iot:region:account-id:client/*
```

  The wildcard * allows any device to connect to AWS IoT.

```
arn:aws:iot:region:account-id:client/${iot:ClientId}
```

  Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * in the previous example.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Connect" ],
      "Resource": [
        "arn:aws:iot:region:account-id:client/${iot:Connection.Thing.ThingName}"
      ],
      "Condition": {
        "Bool": { "iot:Connection.Thing.IsAttached": "true" }
      }
    }
  ]
}
```

  The resource specification contains a variable that matches the device name used to connect. The condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
```

  This allows the device to update the shadow of any device (* = all devices).

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This allows the device to read, update, or delete the shadow of any device.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Publish" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
      ],
    }
  ]
}
```

The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

This allows the device to subscribe to reserved shadow or job topics for all devices.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

The same as the previous example, but using the # wildcard.

```
arn:aws:iot:region:account-id:topic/$aws/things/+/shadow/update
```

This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Subscribe" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
      ],
    }
  ]
}
```

The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

  This is allowed because the device can receive messages only from topics on which it has permission to subscribe.

Read/modify shadow or job data

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Example

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.
- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DeleteThingShadow",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:DescribeJobExecution",
          "iot:GetPendingJobExecutions",
          "iot:StartNextPendingJobExecution",
          "iot:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing1",
        "arn:aws:iot:region:account-id:/thing/MyThing2"
      ]
    }
  ]
}
```

  This allows the device to perform the specified actions on two things only.

Authenticated Cognito role overly permissive

A policy attached to an authenticated Amazon Cognito identity pool role is considered too permissive because it grants permission to perform the following AWS IoT actions:

- Manage or modify things.
- Manage non-thing related data or resources.

Or, because it grants permission to perform the following AWS IoT actions on a broad set of devices:

- Read thing administrative data.
- Use MQTT to connect/publish/subscribe to reserved topics (including shadow or job execution data).
- Use API commands to read or modify shadow or job execution data.

In general, devices that connect using an authenticated Amazon Cognito identity pool role should have only limited permission to read thing-specific administrative data, publish and subscribe to thing-specific MQTT topics, or use the API commands to read and modify thing-specific data related to shadow or job execution data.

This check appears as `AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

For this check, AWS IoT Device Defender audits all Amazon Cognito identity pools that have been used to connect to the AWS IoT message broker during the 31 days before the audit execution. All Amazon Cognito identity pools from which either an authenticated or unauthenticated Amazon Cognito identity connected are included in the audit.

The following reason codes are returned when this check finds a noncompliant authenticated Amazon Cognito identity pool role:

- ALLOWS_BROAD_ACCESS_TO_IOT_THING_ADMIN_READ_ACTIONS
- ALLOWS_ACCESS_TO_IOT_NON_THING_ADMIN_ACTIONS
- ALLOWS_ACCESS_TO_IOT_THING_ADMIN_WRITE_ACTIONS

Why it matters

If an authenticated identity is compromised, it can use administrative actions to modify account settings, delete resources, or gain access to sensitive data.

How to fix it

A policy attached to an authenticated Amazon Cognito identity pool role should grant only those permissions required for a device to do its job. We recommend the following steps:

1. Create a new compliant role.
2. Create a Amazon Cognito identity pool and attach the compliant role to it.
3. Verify that your identities can access AWS IoT using the new pool.
4. After verification is complete, attach the role to the Amazon Cognito identity pool that was flagged as noncompliant.

You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Manage or modify things

The following AWS IoT API actions are used to manage or modify things so permission to perform these should not be granted to devices connecting through an authenticated Amazon Cognito identity pool:

- `AddThingToThingGroup`
- `AttachThingPrincipal`
- `CreateThing`
- `DeleteThing`
- `DetachThingPrincipal`
- `ListThings`
- `ListThingsInThingGroup`
- `RegisterThing`
- `RemoveThingFromThingGroup`
- `UpdateThing`
- `UpdateThingGroupsForThing`

Any role that grants permission to perform these actions on even a single resource is considered noncompliant.

Manage non-things

Devices that connect through an authenticated Amazon Cognito identity pool should not be given permission to perform AWS IoT API actions other than those discussed in these sections. To manage your account with an application that connects through an authenticated Amazon Cognito identity pool, create a separate identity pool not used by devices.

Read thing administrative data

The following AWS IoT API actions are used to read thing data, so devices that connect through an authenticated Amazon Cognito identity pool should be given permission to perform these on a limited set of things only:

- `DescribeThing`
- `ListJobExecutionsForThing`
- `ListThingGroupsForThing`
- `ListThingPrincipals`

- noncompliant:

  ```
  arn:aws:iot:region:account-id:thing/*
  ```

  This allows the device to perform the specified action on any thing.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing"
      ]
    }
  ]
}
```

  This allows the device to perform the specified actions on only one thing.
- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing*"
      ]
    }
  ]
}
```

  This is compliant because, although the resource is specified with a wildcard (*), it is preceded by a specific string, and that limits the set of things accessed to those with names that have the given prefix.

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

  This allows the device to perform the specified action on any thing.
- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing"
      ]
    }
  ]
}
```

This allows the device to perform the specified actions on only one thing.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DescribeThing",
          "iot:ListJobExecutionsForThing",
          "iot:ListThingGroupsForThing",
          "iot:ListThingPrincipals"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing*"
      ]
    }
  ]
}
```

This is compliant because, although the resource is specified with a wildcard (*), it is preceded by a specific string, and that limits the set of things accessed to those with names that have the given prefix.

Subscribe/publish to MQTT topics

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many different actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

```
arn:aws:iot:region:account-id:client/*
```

The wildcard * allows any device to connect to AWS IoT.

```
arn:aws:iot:region:account-id:client/${iot:ClientId}
```

Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * in the previous example.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Connect" ],
      "Resource": [
        "arn:aws:iot:region:account-id:client/${iot:Connection.Thing.ThingName}"
      ],
      "Condition": {
        "Bool": { "iot:Connection.Thing.IsAttached": "true" }
      }
    }
  ]
}
```

The resource specification contains a variable that matches the device name used to connect, and the condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
```

This allows the device to update the shadow of any device (* = all devices).

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This allows the device to read/update/delete the shadow of any device.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Publish" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
      ],
    }
  ]
}
```

The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

This allows the device to subscribe to reserved shadow or job topics for all devices.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/#
```

The same as the previous example, but using the # wildcard.

```
arn:aws:iot:region:account-id:topic/$aws/things/+/shadow/update
```

This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Subscribe" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
      ],
    }
  ]
}
```

The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

This is compliant because the device can receive messages only from topics on which it has permission to subscribe.

Read or modify shadow or job data

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Examples

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

This allows the device to perform the specified action on any thing.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DeleteThingShadow",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:DescribeJobExecution",
          "iot:GetPendingJobExecutions",
          "iot:StartNextPendingJobExecution",
          "iot:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing1",
        "arn:aws:iot:region:account-id:/thing/MyThing2"
      ]
    }
  ]
}
```

This allows the device to perform the specified actions on only two things.

AWS IoT policies overly permissive

An AWS IoT policy gives permissions that are too broad or unrestricted. It grants permission to send or receive MQTT messages for a broad set of devices, or grants permission to access or modify shadow and job execution data for a broad set of devices.

In general, a policy for a device should grant access to resources associated with just that device and no or very few other devices. With some exceptions, using a wildcard (for example, "*") to specify resources in such a policy is considered too broad or unrestricted.

This check appears as `IOT_POLICY_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

The following reason code is returned when this check finds a noncompliant AWS IoT policy:

- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

A certificate, Amazon Cognito identity, or thing group with an overly permissive policy can, if compromised, impact the security of your entire account. An attacker could use such broad access to read or modify shadows, jobs, or job executions for all your devices. Or an attacker could use a compromised certificate to connect malicious devices or launch a DDOS attack on your network.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Use CreatePolicyVersion to create a new, compliant version of the policy. Set the `setAsDefault` flag to true. (This makes this new version operative for all entities that use the policy.)

2. Use ListTargetsForPolicy to get a list of targets (certificates, thing groups) that the policy is attached to and determine which devices are included in the groups or which use the certificates to connect.
3. Verify that all associated devices are able to connect to AWS IoT. If a device is unable to connect, use SetPolicyVersion to roll back the default policy to the previous version, revise the policy, and try again.

You can use mitigation actions to:

- Apply the `REPLACE_DEFAULT_POLICY_VERSION` mitigation action on your audit findings to make this change.
- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Use AWS IoT policy variables (p. 238) to dynamically reference AWS IoT resources in your policies.

MQTT permissions

MQTT messages are sent through the AWS IoT message broker and are used by devices to perform many actions, including accessing and modifying shadow state and job execution state. A policy that grants permission to a device to connect, publish, or subscribe to MQTT messages should restrict these actions to specific resources as follows:

Connect

- noncompliant:

  ```
  arn:aws:iot:region:account-id:client/*
  ```

  The wildcard * allows any device to connect to AWS IoT.

  ```
  arn:aws:iot:region:account-id:client/${iot:ClientId}
  ```

  Unless `iot:Connection.Thing.IsAttached` is set to true in the condition keys, this is equivalent to the wildcard * as in the previous example.
- compliant:

  ```
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [ "iot:Connect" ],
        "Resource": [
          "arn:aws:iot:region:account-id:client/${iot:Connection.Thing.ThingName}"
        ],
        "Condition": {
          "Bool": { "iot:Connection.Thing.IsAttached": "true" }
        }
      }
    ]
  }
  ```

  The resource specification contains a variable that matches the device name used to connect. The condition statement further restricts the permission by checking that the certificate used by the MQTT client matches that attached to the thing with the name used.

Publish

- noncompliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*/shadow/update
```

This allows the device to update the shadow of any device (* = all devices).

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This allows the device to read, update, or delete the shadow of any device.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Publish" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
      ],
    }
  ]
}
```

The resource specification contains a wildcard, but it only matches any shadow-related topic for the device whose thing name is used to connect.

Subscribe

- noncompliant:

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

This allows the device to subscribe to reserved shadow or job topics for all devices.

```
arn:aws:iot:region:account-id:topicfilter/$aws/things/*
```

The same as the previous example, but using the # wildcard.

```
arn:aws:iot:region:account-id:topic/$aws/things/+/shadow/update
```

This allows the device to see shadow updates on any device (+ = all devices).

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [ "iot:Subscribe" ],
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
        "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/jobs/*"
```

```
        ],
      }
    ]
}
```

The resource specifications contain wildcards, but they only match any shadow-related topic and any job-related topic for the device whose thing name is used to connect.

Receive

- compliant:

```
arn:aws:iot:region:account-id:topic/$aws/things/*
```

This is compliant because the device can only receive messages from topics on which it has permission to subscribe.

Shadow and job permissions

A policy that grants permission to a device to perform an API action to access or modify device shadows or job execution data should restrict these actions to specific resources. The following are the API actions:

- `DeleteThingShadow`
- `GetThingShadow`
- `UpdateThingShadow`
- `DescribeJobExecution`
- `GetPendingJobExecutions`
- `StartNextPendingJobExecution`
- `UpdateJobExecution`

Examples

- noncompliant:

```
arn:aws:iot:region:account-id:thing/*
```

This allows the device to perform the specified action on any thing.

- compliant:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "iot:DeleteThingShadow",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:DescribeJobExecution",
          "iot:GetPendingJobExecutions",
          "iot:StartNextPendingJobExecution",
          "iot:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:/thing/MyThing1",
        "arn:aws:iot:region:account-id:/thing/MyThing2"
      ]
    }
  ]
}
```

This allows the device to perform the specified actions on only two things.

Role alias overly permissive

AWS IoT role alias provides a mechanism for connected devices to authenticate to AWS IoT using X.509 certificates and then obtain short-lived AWS credentials from an IAM role that is associated with an AWS IoT role alias. The permissions for these credentials must be scoped down using access policies with authentication context variables. If your policies are not configured correctly, you could leave yourself exposed to an escalation of privilege attack. This audit check ensures that the temporary credentials provided by AWS IoT role aliases are not overly permissive.

This check is triggered if one of the following conditions are found:

- The policy provides administrative permissions to any services used in the past year by this role alias (for example, "iot:*", "dynamodb:*", "iam:*", and so on).
- The policy provides broad access to thing metadata actions, access to restricted AWS IoT actions, or broad access to AWS IoT data plane actions.
- The policy provides access to security auditing services such as "iam", "cloudtrail", "guardduty", "inspector", or "trustedadvisor".

This check appears as `IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK` in the CLI and API.

Severity: **Critical**

Details

The following reason codes are returned when this check finds a noncompliant IoT policy:

- ALLOWS_BROAD_ACCESS_TO_USED_SERVICES
- ALLOWS_ACCESS_TO_SECURITY_AUDITING_SERVICES
- ALLOWS_BROAD_ACCESS_TO_IOT_THING_ADMIN_READ_ACTIONS
- ALLOWS_ACCESS_TO_IOT_NON_THING_ADMIN_ACTIONS
- ALLOWS_ACCESS_TO_IOT_THING_ADMIN_WRITE_ACTIONS
- ALLOWS_BROAD_ACCESS_TO_IOT_DATA_PLANE_ACTIONS

Why it matters

By limiting permissions to those that are required for a device to perform its normal operations, you reduce the risks to your account if a device is compromised.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Follow the steps in Authorizing direct calls to AWS services (p. 269) to apply a more restrictive policy to your role alias.

You can use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom action in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Role alias allows access to unused services

AWS IoT role alias provides a mechanism for connected devices to authenticate to AWS IoT using X.509 certificates and then obtain short-lived AWS credentials from an IAM role that is associated with an AWS IoT role alias. The permissions for these credentials must be scoped down using access policies with authentication context variables. If your policies are not configured correctly, you could leave yourself exposed to an escalation of privilege attack. This audit check ensures that the temporary credentials provided by AWS IoT role aliases are not overly permissive.

This check is triggered if the role alias has access to services that haven't been used for the AWS IoT device in the last year. For example, the audit reports if you have an IAM role linked to the role alias that has only used AWS IoT in the past year but the policy attached to the role also grants permission to `"iam:getRole"` and `"dynamodb:PutItem"`.

This check appears as `IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK` in the CLI and API.

Severity: **Medium**

Details

The following reason codes are returned when this check finds a noncompliant AWS IoT policy:

- ALLOWS_ACCESS_TO_UNUSED_SERVICES

Why it matters

By limiting permissions to those services that are required for a device to perform its normal operations, you reduce the risks to your account if a device is compromised.

How to fix it

Follow these steps to fix any noncompliant policies attached to things, thing groups, or other entities:

1. Follow the steps in Authorizing direct calls to AWS services (p. 269) to apply a more restrictive policy to your role alias.

You can use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom action in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

CA certificate expiring

A CA certificate is expiring within 30 days or has expired.

This check appears as `CA_CERT_APPROACHING_EXPIRATION_CHECK` in the CLI and API.

Severity: **Medium**

Details

This check applies to CA certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant CA certificate:

- CERTIFICATE_APPROACHING_EXPIRATION
- CERTIFICATE_PAST_EXPIRATION

Why it matters

An expired CA certificate should not be used to sign new device certificates.

How to fix it

Consult your security best practices for how to proceed. You might want to:

1. Register a new CA certificate with AWS IoT.
2. Verify that you are able to sign device certificates using the new CA certificate.
3. Use UpdateCACertificate to mark the old CA certificate as INACTIVE in AWS IoT. You can also use mitigation actions to do the following:
   - Apply the `UPDATE_CA_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).

Conflicting MQTT client IDs

Multiple devices connect using the same client ID.

This check appears as `CONFLICTING_CLIENT_IDS_CHECK` in the CLI and API.

Severity: **High**

Details

Multiple connections were made using the same client ID, causing an already connected device to be disconnected. The MQTT specification allows only one active connection per client ID, so when another device connects using the same client ID, it knocks the previous one off the connection.

When performed as part of an on-demand audit, this check looks at how client IDs were used to connect during the 31 days prior to the start of the audit. For scheduled audits, this check looks at data from the last time the audit was run to the time this instance of the audit started. If you have taken steps to mitigate this condition during the time checked, note when the connections/disconnections were made to determine if the problem persists.

The following reason codes are returned when this check finds noncompliance:

- DUPLICATE_CLIENT_ID_ACROSS_CONNECTIONS

The findings returned by this check also include the client ID used to connect, principal IDs, and disconnect times. The most recent results are listed first.

Why it matters

Devices with conflicting IDs are forced to constantly reconnect, which might result in lost messages or leave a device unable to connect.

This might indicate that a device or a device's credentials have been compromised, and might be part of a DDoS attack. It is also possible that devices are not configured correctly in the account or a device has a bad connection and is forced to reconnect several times per minute.

How to fix it

Register each device as a unique thing in AWS IoT, and use the thing name as the client ID to connect. Or use a UUID as the client ID when connecting the device over MQTT. You can also use mitigation actions to:

- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Device certificate expiring

A device certificate is expiring within 30 days or has expired.

This check appears as `DEVICE_CERTIFICATE_EXPIRING_CHECK` in the CLI and API.

Severity: **Medium**

Details

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds a noncompliant device certificate:

- CERTIFICATE_APPROACHING_EXPIRATION
- CERTIFICATE_PAST_EXPIRATION

Why it matters

A device certificate should not be used after it expires.

How to fix it

Consult your security best practices for how to proceed. You might want to:

1. Provision a new certificate and attach it to the device.
2. Verify that the new certificate is valid and the device is able to use it to connect.
3. Use UpdateCertificate to mark the old certificate as INACTIVE in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).
4. Detach the old certificate from the device. (See DetachThingPrincipal.)

Revoked device certificate still active

A revoked device certificate is still active.

This check appears as `REVOKED_DEVICE_CERT_CHECK` in the CLI and API.

Severity: **Medium**

Details

A device certificate is in its CA's certificate revocation list, but it is still active in AWS IoT.

This check applies to device certificates that are ACTIVE or PENDING_TRANSFER.

The following reason codes are returned when this check finds noncompliance:

- CERTIFICATE_REVOKED_BY_ISSUER

Why it matters

A device certificate is usually revoked because it has been compromised. It is possible that it has not yet been revoked in AWS IoT due to an error or oversight.

How to fix it

Verify that the device certificate has not been compromised. If it has, follow your security best practices to mitigate the situation. You might want to:

1. Provision a new certificate for the device.
2. Verify that the new certificate is valid and the device is able to use it to connect.
3. Use UpdateCertificate to mark the old certificate as REVOKED in AWS IoT. You can also use mitigation actions to:
   - Apply the `UPDATE_DEVICE_CERTIFICATE` mitigation action on your audit findings to make this change.
   - Apply the `ADD_THINGS_TO_THING_GROUP` mitigation action to add the device to a group where you can take action on it.
   - Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

   For more information, see Mitigation actions (p. 877).
4. Detach the old certificate from the device. (See DetachThingPrincipal.)

Logging disabled

AWS IoT logs are not enabled in Amazon CloudWatch. Verifies both V1 and V2 logging.

This check appears as `LOGGING_DISABLED_CHECK` in the CLI and API.

Severity: **Low**

Details

The following reason codes are returned when this check finds noncompliance:

- LOGGING_DISABLED

Why it matters

AWS IoT logs in CloudWatch provide visibility into behaviors in AWS IoT, including authentication failures and unexpected connects and disconnects that might indicate that a device has been compromised.

How to fix it

Enable AWS IoT logs in CloudWatch. See Monitoring Tools (p. 303). You can also use mitigation actions to:

- Apply the `ENABLE_IOT_LOGGING` mitigation action on your audit findings to make this change.
- Apply the `PUBLISH_FINDINGS_TO_SNS` mitigation action if you want to implement a custom response in response to the Amazon SNS message.

For more information, see Mitigation actions (p. 877).

Audit commands

Manage audit settings

Use `UpdateAccountAuditConfiguration` to configure audit settings for your account. This command allows you to enable those checks you want to be available for audits, set up optional notifications, and configure permissions.

Check these settings with `DescribeAccountAuditConfiguration`.

Use `DeleteAccountAuditConfiguration` to delete your audit settings. This restores all default values, and effectively disables audits because all checks are disabled by default.

UpdateAccountAuditConfiguration

Configures or reconfigures the Device Defender audit settings for this account. Settings include how audit notifications are sent and which audit checks are enabled or disabled.

Synopsis

```
aws iot  update-account-audit-configuration \
    [--role-arn <value>] \
    [--audit-notification-target-configurations <value>] \
    [--audit-check-configurations <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "roleArn": "string",
  "auditNotificationTargetConfigurations": {
    "string": {
      "targetArn": "string",
      "roleArn": "string",
      "enabled": "boolean"
    }
  },
  "auditCheckConfigurations": {
    "string": {
      "enabled": "boolean"
    }
  }
}
```

`cli-input-json` Fields

Name	Type	Description
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to AWS IoT to access information about your devices, policies, certificates, and other items when performing an audit.
auditNotificationTargetConfigurations	map	Information about the targets to which audit notifications are sent.
targetArn	string	The ARN of the target (SNS topic) to which audit notifications are sent.
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to send notifications to the target.
enabled	boolean	True if notifications to the target are enabled.
auditCheckConfigurations	map	Specifies which audit checks are enabled and disabled for this account. Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are currently enabled. Some data collection might start immediately when certain checks are enabled. When a check is disabled, any data collected so far in relation to the check is deleted. You cannot disable a check if it is used by any scheduled audit. You must first delete the check from the scheduled audit or delete the scheduled audit itself. On the first call to `UpdateAccountAuditConfiguration`, this parameter is required

Name	Type	Description
		and must specify at least one enabled check.
enabled	boolean	True if this audit check is enabled for this account.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DescribeAccountAuditConfiguration

Gets information about the Device Defender audit settings for this account. Settings include how audit notifications are sent and which audit checks are enabled or disabled.

Synopsis

```
aws iot  describe-account-audit-configuration  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
}
```

Output

```
{
  "roleArn": "string",
  "auditNotificationTargetConfigurations": {
    "string": {
      "targetArn": "string",
      "roleArn": "string",
      "enabled": "boolean"
    }
  },
  "auditCheckConfigurations": {
    "string": {
      "enabled": "boolean"
    }
  }
}
```

CLI output fields

Name	Type	Description
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to AWS IoT to access information about your devices, policies, certificates, and other items when performing an audit. On the first call to `UpdateAccountAuditConfiguration`, this parameter is required.
auditNotificationTargetConfigurations	map	Information about the targets to which audit notifications are sent for this account.
targetArn	string	The ARN of the target (SNS topic) to which audit notifications are sent.
roleArn	string length- max:2048 min:20	The ARN of the role that grants permission to send notifications to the target.
enabled	boolean	True if notifications to the target are enabled.
auditCheckConfigurations	map	Which audit checks are enabled and disabled for this account.
enabled	boolean	True if this audit check is enabled for this account.

Errors

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DeleteAccountAuditConfiguration

Restores the default settings for Device Defender audits for this account. Any configuration data you entered is deleted and all audit checks are reset to disabled.

Synopsis

```
aws iot  delete-account-audit-configuration \
    [--delete-scheduled-audits | --no-delete-scheduled-audits]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "deleteScheduledAudits": "boolean"
}
```

`cli-input-json` Fields

Name	Type	Description
deleteScheduledAudits	boolean	If true, all scheduled audits are deleted.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Schedule audits

Use `CreateScheduledAudit` to create one or more scheduled audits. This command allows you to specify the checks you want to perform during an audit and how often the audit should be run.

Keep track of your scheduled audits with `ListScheduledAudits` and `DescribeScheduledAudit`.

Change an existing scheduled audit with `UpdateScheduledAudit` or delete it with `DeleteScheduledAudit`.

CreateScheduledAudit

Creates a scheduled audit that is run at a specified time interval.

Synopsis

```
aws iot  create-scheduled-audit \
    --frequency <value> \
    [--day-of-month <value>] \
    [--day-of-week <value>] \
    --target-check-names <value> \
    [--tags <value>] \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. Can be one of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. This field is required if the `frequency` parameter is set to MONTHLY. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. Can be one of SUN, MON, TUE,WED, THU, FRI, or SAT. This field is required if the `frequency` parameter is set to WEEKLY or BIWEEKLY. enum: SUN | MON | TUE | WED | THU | FRI | SAT

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or `UpdateAccountAuditConfiguration` to select which checks are enabled.)
tags	list member: Tag java class: java.util.List	Metadata that can be used to manage the scheduled audit.
Key	string	The tag's key.
Value	string	The tag's value.
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name you want to give to the scheduled audit. (Maximum of 128 characters)

Output

```
{
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

`LimitExceededException`

A limit has been exceeded.

ListScheduledAudits

Lists all of your scheduled audits.

Synopsis

```
aws iot  list-scheduled-audits \
    [--next-token <value>] \
    [--max-results <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "nextToken": "string",
  "maxResults": "integer"
}
```

`cli-input-json` Fields

Name	Type	Description
nextToken	string	The token for the next set of results.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.

Output

```
{
  "scheduledAudits": [
    {
      "scheduledAuditName": "string",
      "scheduledAuditArn": "string",
      "frequency": "string",
      "dayOfMonth": "string",
      "dayOfWeek": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAudits	list member: ScheduledAuditMetadata java class: java.util.List	The list of scheduled audits.
scheduledAuditName	string	The name of the scheduled audit.

Name	Type	Description
	length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	
scheduledAuditArn	string	The ARN of the scheduled audit.
frequency	string	How often the scheduled audit takes place. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit is run (if the `frequency` is MONTHLY). If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit is run (if the `frequency` is WEEKLY or BIWEEKLY). enum: SUN | MON | TUE | WED | THU | FRI | SAT
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no more results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DescribeScheduledAudit

Gets information about a scheduled audit.

Synopsis

```
aws iot  describe-scheduled-audit \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit whose information you want to get.

Output

```
{
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "scheduledAuditName": "string",
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. One of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. One of SUN, MON, TUE, WED, THU, FRI, or SAT. enum: SUN | MON | TUE | WED | THU | FRI | SAT

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.)
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit.
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

UpdateScheduledAudit

Updates a scheduled audit, including which checks are performed and how often the audit takes place.

Synopsis

```
aws iot  update-scheduled-audit \
    [--frequency <value>] \
    [--day-of-month <value>] \
    [--day-of-week <value>] \
    [--target-check-names <value>] \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "frequency": "string",
  "dayOfMonth": "string",
  "dayOfWeek": "string",
  "targetCheckNames": [
    "string"
  ],
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
frequency	string	How often the scheduled audit takes place. Can be one of DAILY, WEEKLY, BIWEEKLY, or MONTHLY. The actual start time of each audit is determined by the system. enum: DAILY | WEEKLY | BIWEEKLY | MONTHLY
dayOfMonth	string pattern: ^([1-9]|[12][0-9]|3[01])$\|^LAST$	The day of the month on which the scheduled audit takes place. Can be 1 through 31 or LAST. This field is required if the `frequency` parameter is set to MONTHLY. If days 29-31 are specified, and the month does not have that many days, the audit takes place on the LAST day of the month.
dayOfWeek	string	The day of the week on which the scheduled audit takes place. Can be one of SUN, MON, TUE, WED, THU, FRI, or SAT. This field is required if the `frequency` parameter is set to WEEKLY or BIWEEKLY. enum: SUN | MON | TUE | WED | THU | FRI | SAT
targetCheckNames	list member: AuditCheckName	Which checks are performed during the scheduled audit. Checks must be enabled for your account. (Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.)
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit. (Maximum of 128 characters)

Output

```
{
  "scheduledAuditArn": "string"
}
```

CLI output fields

Name	Type	Description
scheduledAuditArn	string	The ARN of the scheduled audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

DeleteScheduledAudit

Deletes a scheduled audit.

Synopsis

```
aws iot  delete-scheduled-audit \
    --scheduled-audit-name <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "scheduledAuditName": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit you want to delete.

Output

None

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Run an On-Demand audit

Use `StartOnDemandAuditTask` to specify the checks you want to perform and start an audit running right away.

StartOnDemandAuditTask

Starts an on-demand Device Defender audit.

Synopsis

```
aws iot  start-on-demand-audit-task \
    --target-check-names <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "targetCheckNames": [
    "string"
  ]
}
```

`cli-input-json` Fields

Name	Type	Description
targetCheckNames	list member: AuditCheckName	Which checks are performed during the audit. The checks you specify must be enabled for your account or an exception occurs. Use `DescribeAccountAuditConfiguration` to see the list of all checks, including those that are enabled or use `UpdateAccountAuditConfiguration` to select which checks are enabled.

Output

```
{
  "taskId": "string"
}
```

CLI output fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the on-demand audit you started.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

`LimitExceededException`

A limit has been exceeded.

Manage audit instances

Use `DescribeAuditTask` to get information about a specific audit instance. If it has already run, the results include which checks failed and which passed, those that the system was unable to complete, and if the audit is still in progress, those it is still working on.

Use `ListAuditTasks` to find the audits that were run during a specified time interval.

Use `CancelAuditTask` to halt an audit in progress.

DescribeAuditTask

Gets information about a Device Defender audit.

Synopsis

```
aws iot  describe-audit-task \
    --task-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit whose information you want to get.

Output

```
{
  "taskStatus": "string",
  "taskType": "string",
  "taskStartTime": "timestamp",
  "taskStatistics": {
    "totalChecks": "integer",
    "inProgressChecks": "integer",
    "waitingForDataCollectionChecks": "integer",
    "compliantChecks": "integer",
    "nonCompliantChecks": "integer",
    "failedChecks": "integer",
    "canceledChecks": "integer"
  },
  "scheduledAuditName": "string",
  "auditDetails": {
    "string": {
      "checkRunStatus": "string",
      "checkCompliant": "boolean",
      "totalResourcesCount": "long",
      "nonCompliantResourcesCount": "long",
      "errorCode": "string",
      "message": "string"
    }
  }
}
```

CLI output fields

Name	Type	Description
taskStatus	string	The status of the audit: one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED. enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
taskType	string	The type of audit: ON_DEMAND_AUDIT_TASK or SCHEDULED_AUDIT_TASK. enum: ON_DEMAND_AUDIT_TASK | SCHEDULED_AUDIT_TASK

Name	Type	Description
taskStartTime	timestamp	The time the audit started.
taskStatistics	TaskStatistics	Statistical information about the audit.
totalChecks	integer	The number of checks in this audit.
inProgressChecks	integer	The number of checks in progress.
waitingForDataCollectionChecks	integer	The number of checks waiting for data collection.
compliantChecks	integer	The number of checks that found compliant resources.
nonCompliantChecks	integer	The number of checks that found noncompliant resources.
failedChecks	integer	The number of checks.
canceledChecks	integer	The number of checks that did not run because the audit was canceled.
scheduledAuditName	string length- max:128 min:1 pattern: [a-zA-Z0-9_-]+	The name of the scheduled audit (only if the audit was a scheduled audit).
auditDetails	map	Detailed information about each check performed during this audit.
checkRunStatus	string	The completion status of this check, one of IN_PROGRESS, WAITING_FOR_DATA_COLLECTION, CANCELED, COMPLETED_COMPLIANT, COMPLETED_NON_COMPLIANT, or FAILED. enum: IN_PROGRESS | WAITING_FOR_DATA_COLLECTION | CANCELED | COMPLETED_COMPLIANT | COMPLETED_NON_COMPLIANT | FAILED
checkCompliant	boolean	True if the check completed and found all resources compliant.
totalResourcesCount	long	The number of resources on which the check was performed.

Name	Type	Description
nonCompliantResourcesCount	long	The number of resources that the check found noncompliant.
errorCode	string	The code of any error encountered when performing this check during this audit. One of INSUFFICIENT_PERMISSIONS or AUDIT_CHECK_DISABLED.
message	string length- max:2048	The message associated with any error encountered when performing this check during this audit.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ResourceNotFoundException`

The specified resource does not exist.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

ListAuditTasks

Lists the Device Defender audits that have been performed during a given time period.

Synopsis

```
aws iot  list-audit-tasks \
    --start-time <value> \
    --end-time <value> \
    [--task-type <value>] \
    [--task-status <value>] \
    [--next-token <value>] \
    [--max-results <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "startTime": "timestamp",
  "endTime": "timestamp",
  "taskType": "string",
  "taskStatus": "string",
  "nextToken": "string",
  "maxResults": "integer"
}
```

`cli-input-json` Fields

Name	Type	Description
startTime	timestamp	The beginning of the time period. Audit information is retained for a limited time (180 days). Requesting a start time prior to what is retained results in an `InvalidRequestException`.
endTime	timestamp	The end of the time period.
taskType	string	A filter to limit the output to the specified type of audit: can be one of ON_DEMAND_AUDIT_TASK or SCHEDULED__AUDIT_TASK. enum: ON_DEMAND_AUDIT_TASK | SCHEDULED_AUDIT_TASK
taskStatus	string	A filter to limit the output to audits with the specified completion status: can be one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED. enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
nextToken	string	The token for the next set of results.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.

Output

```
{
  "tasks": [
    {
      "taskId": "string",
      "taskStatus": "string",
      "taskType": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
tasks	list member: AuditTaskMetadata java class: java.util.List	The audits that were performed during the specified time period.
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of this audit.
taskStatus	string	The status of this audit: one of IN_PROGRESS, COMPLETED, FAILED, or CANCELED. enum: IN_PROGRESS | COMPLETED | FAILED | CANCELED
taskType	string	The type of this audit: one of ON_DEMAND_AUDIT_TASK or SCHEDULED_AUDIT_TASK. enum: ON_DEMAND_AUDIT_TASK | SCHEDULED_AUDIT_TASK
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no additional results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

CancelAuditTask

Cancels an audit that is in progress. The audit can be either scheduled or on-demand. If the audit is not in progress, an `InvalidRequestException` occurs.

Synopsis

```
aws iot  cancel-audit-task \
    --task-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit you want to cancel. You can only cancel an audit that is IN_PROGRESS.

Output

None

Errors

`ResourceNotFoundException`

The specified resource does not exist.

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Check audit results

Use `ListAuditFindings` to see the results of an audit. You can filter the results by the type of check, a specific resource, or the time of the audit. You can use this information to mitigate any problems that were found.

You can define mitigation actions and apply them to the findings from your audit. For more information, see Mitigation actions (p. 877).

ListAuditFindings

Lists the findings (results) of a Device Defender audit or of the audits performed during a specified time period. (Findings are retained for 180 days.)

Synopsis

```
aws iot  list-audit-findings \
    [--task-id <value>] \
    [--check-name <value>] \
    [--resource-identifier <value>] \
    [--max-results <value>] \
    [--next-token <value>] \
    [--start-time <value>] \
    [--end-time <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "taskId": "string",
  "checkName": "string",
  "resourceIdentifier": {
    "deviceCertificateId": "string",
    "caCertificateId": "string",
    "cognitoIdentityPoolId": "string",
    "clientId": "string",
    "policyVersionIdentifier": {
      "policyName": "string",
      "policyVersionId": "string"
    },

    "roleAliasArn": "string",
    "account": "string"
  },
  "maxResults": "integer",
  "nextToken": "string",
  "startTime": "timestamp",
  "endTime": "timestamp"
}
```

`cli-input-json` Fields

Name	Type	Description
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	A filter to limit results to the audit with the specified ID. You must specify either the taskId or the startTime and endTime, but not both.
checkName	string	A filter to limit results to the findings for the specified audit check.
resourceIdentifier	ResourceIdentifier	Information that identifies the noncompliant resource.
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the CA certificate used to authorize the certificate.

Name	Type	Description
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.
roleAliasArn	string	The ARN of the role alias that has overly permissive actions. length- max:2048 min:1
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
maxResults	integer range- max:250 min:1	The maximum number of results to return at one time. The default is 25.
nextToken	string	The token for the next set of results.
startTime	timestamp	A filter to limit results to those found after the specified time. You must specify either the startTime and endTime or the taskId, but not both.
endTime	timestamp	A filter to limit results to those found before the specified time. You must specify either the startTime and endTime or the taskId, but not both.

Output

```
{
  "findings": [
    {
      "taskId": "string",
      "checkName": "string",
      "taskStartTime": "timestamp",
      "findingTime": "timestamp",
      "severity": "string",
      "nonCompliantResource": {
        "resourceType": "string",
        "resourceIdentifier": {
          "deviceCertificateId": "string",
          "caCertificateId": "string",
          "cognitoIdentityPoolId": "string",
          "clientId": "string",
          "policyVersionIdentifier": {
            "policyName": "string",
            "policyVersionId": "string"
          },
          "account": "string"
        },
        "additionalInfo": {
          "string": "string"
        }
      },
      "relatedResources": [
        {
          "resourceType": "string",
          "resourceIdentifier": {
            "deviceCertificateId": "string",
            "caCertificateId": "string",
            "cognitoIdentityPoolId": "string",
            "clientId": "string",

            "iamRoleArn": "string",

            "policyVersionIdentifier": {
              "policyName": "string",
              "policyVersionId": "string"
            },
            "account": "string"
          },

          "roleAliasArn": "string",

          "additionalInfo": {
            "string": "string"
          }
        }
      ],
      "reasonForNonCompliance": "string",
      "reasonForNonComplianceCode": "string"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Type	Description
findings	list member: AuditFinding	The findings (results) of the audit.
taskId	string length- max:40 min:1 pattern: [a-zA-Z0-9-]+	The ID of the audit that generated this result (finding).

Name	Type	Description
checkName	string	The audit check that generated this result.
taskStartTime	timestamp	The time the audit started.
findingTime	timestamp	The time the result (finding) was discovered.
severity	string	The severity of the result (finding). enum: CRITICAL | HIGH | MEDIUM | LOW
nonCompliantResource	NonCompliantResource	The resource that was found to be noncompliant with the audit check.
resourceType	string	The type of the noncompliant resource. enum: DEVICE_CERTIFICATE | CA_CERTIFICATE | IOT_POLICY | COGNITO_IDENTITY_POOL | CLIENT_ID | ACCOUNT_SETTINGS
resourceIdentifier	ResourceIdentifier	Information that identifies the noncompliant resource.
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the CA certificate used to authorize the certificate.
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.

Name	Type	Description
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
additionalInfo	map	Other information about the noncompliant resource.
relatedResources	list member: RelatedResource	The list of related resources.
resourceType	string	The type of resource. enum: DEVICE_CERTIFICATE | CA_CERTIFICATE | IOT_POLICY | COGNITO_IDENTITY_POOL | CLIENT_ID | ACCOUNT_SETTINGS
resourceIdentifier	ResourceIdentifier	Information that identifies the resource.
deviceCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the certificate attached to the resource.
caCertificateId	string length- max:64 min:64 pattern: (0x)?[a-fA-F0-9]+	The ID of the CA certificate used to authorize the certificate.
cognitoIdentityPoolId	string	The ID of the Amazon Cognito identity pool.
clientId	string	The client ID.
policyVersionIdentifier	PolicyVersionIdentifier	The version of the policy associated with the resource.
iamRoleArn	string length- max:2048 min:20	The ARN of the IAM role that has overly permissive actions.
policyName	string length- max:128 min:1 pattern: [w+=,.@-]+	The name of the policy.
policyVersionId	string pattern: [0-9]+	The ID of the version of the policy associated with the resource.

Name	Type	Description
roleAliasArn	string length- max:2048 min:1	The ARN of the role alias that has overly permissive actions.
account	string length- max:12 min:12 pattern: [0-9]+	The account with which the resource is associated.
additionalInfo	map	Other information about the resource.
reasonForNonCompliance	string	The reason the resource was noncompliant.
reasonForNonComplianceCode	string	A code that indicates the reason that the resource was noncompliant.
nextToken	string	A token that can be used to retrieve the next set of results, or `null` if there are no additional results.

Errors

`InvalidRequestException`

The contents of the request were invalid.

`ThrottlingException`

The rate exceeds the limit.

`InternalFailureException`

An unexpected error has occurred.

Audit finding suppressions

When you run an audit, it reports findings for all non-compliant resources. This means your audit reports include findings for resources where you're working toward mitigating issues and also for resources that are known to be non-compliant, such as test or broken devices. The audit continues to report findings for resources that remain non-compliant in successive audit runs, which may add unwanted information to your reports. Audit finding suppressions enable you to suppress or filter out findings for a defined period of time until the resource is fixed, or indefinitely for a resource associated with a test or broken device.

Note
Mitigation actions won't be available for suppressed audit findings. For more information about mitigation actions, see Mitigation actions (p. 877).

For information about audit finding suppression quotas, see AWS IoT Device Defender endpoints and quotas.

How audit finding suppressions work

When you create an audit finding suppression for a non-compliant resource, your audit reports and notifications behave differently.

Your audit reports will include a new section that lists all the suppressed findings associated with the report. Suppressed findings won't be considered when we evaluate whether an audit check is compliant or not. A suppressed resource count is also returned for each audit check when you use the describe-audit-task command in the command line interface (CLI).

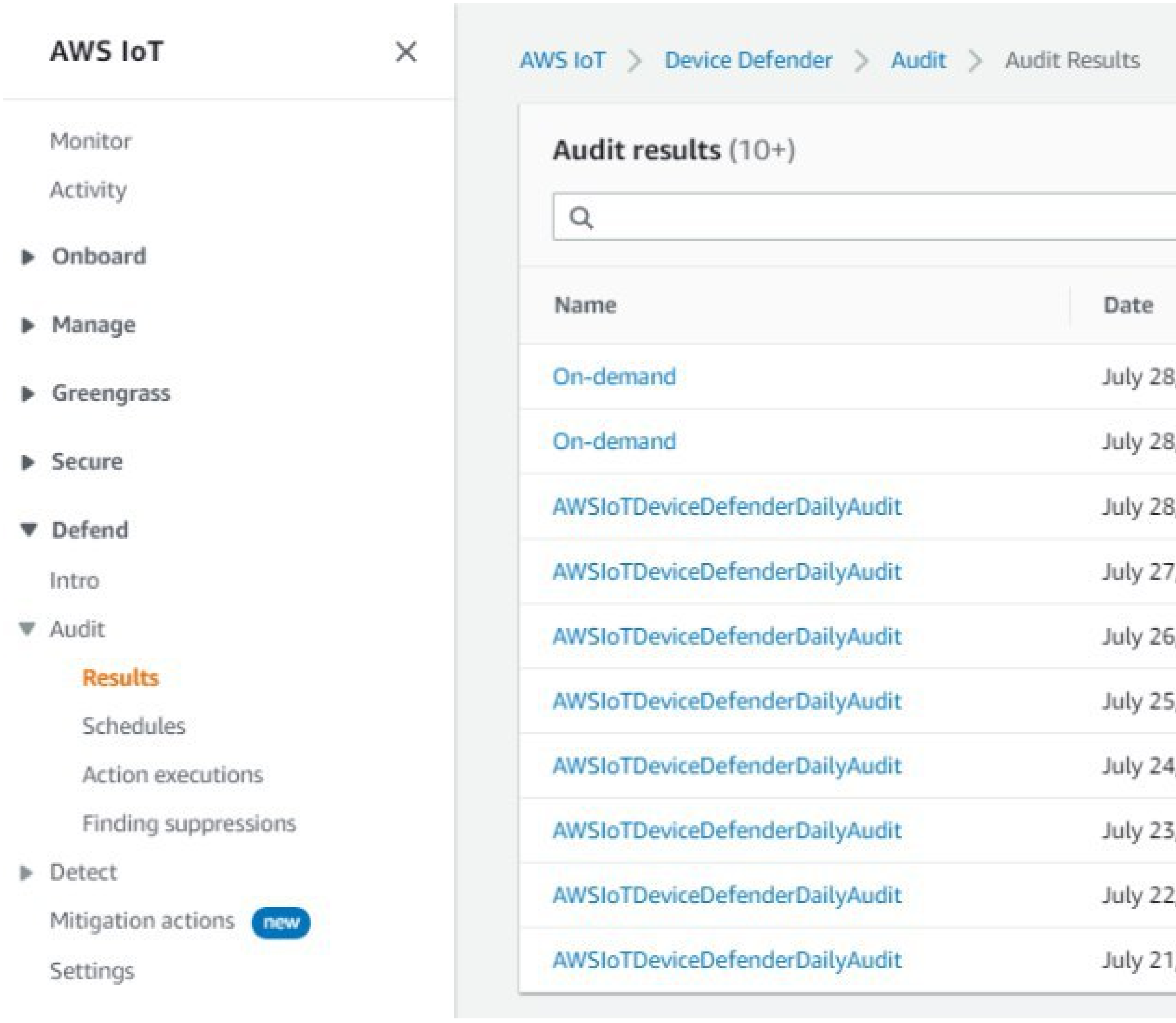
For audit notifications, suppressed findings aren't considered when we evaluate whether an audit check is compliant or not. A suppressed resource count is also included in each audit check notification AWS IoT Device Defender publishes to Amazon CloudWatch and Amazon Simple Notification Service (Amazon SNS).

How to use audit finding suppressions in the console

To suppress a finding from an audit report

The following procedure shows you how to create an audit finding suppression in the AWS IoT console.

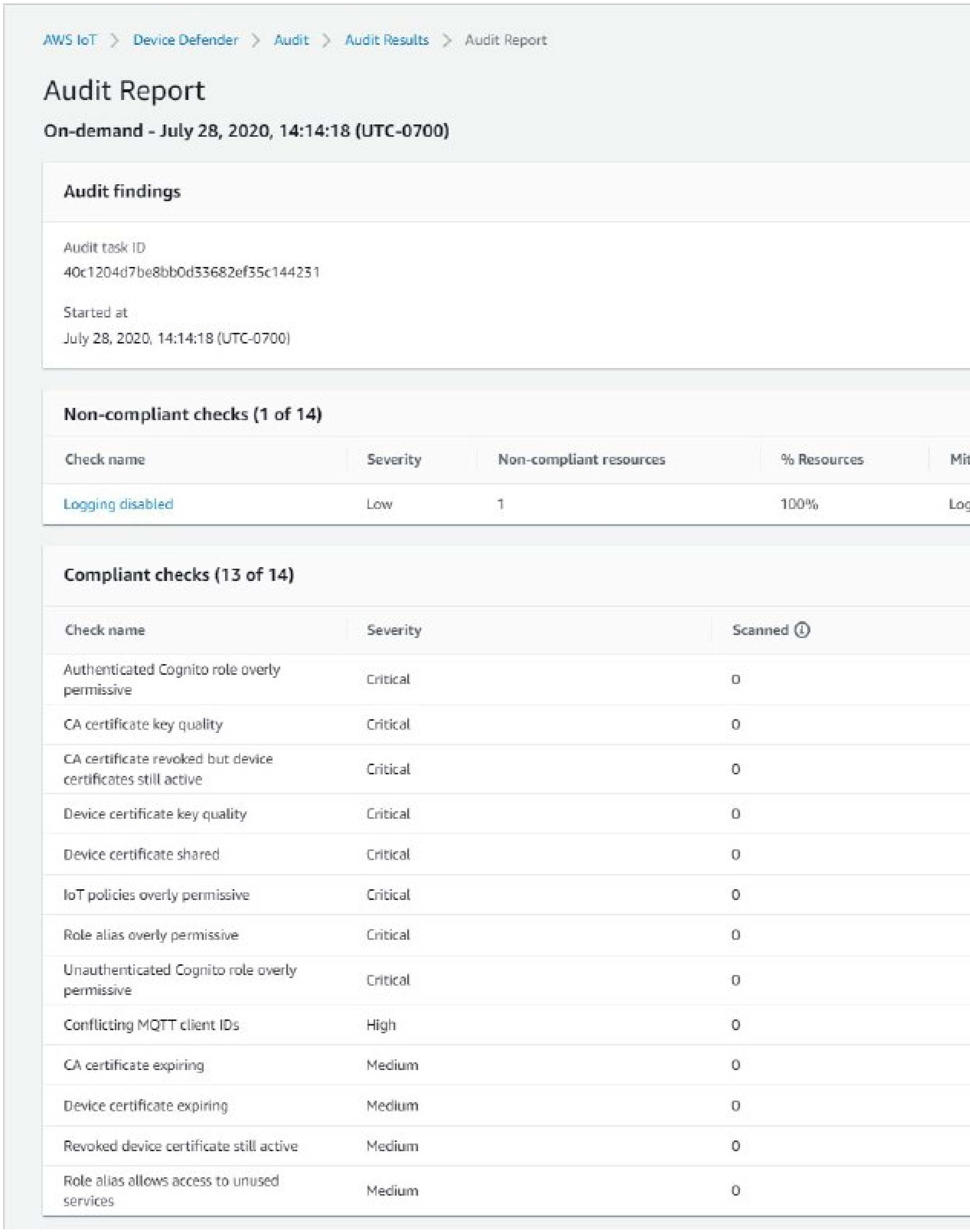
1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Results**.
2. Select an audit report you'd like to review.



Audit results (10+)

Name	Date
On-demand	July 28
On-demand	July 28
AWSIoTDeviceDefenderDailyAudit	July 28
AWSIoTDeviceDefenderDailyAudit	July 27
AWSIoTDeviceDefenderDailyAudit	July 26
AWSIoTDeviceDefenderDailyAudit	July 25
AWSIoTDeviceDefenderDailyAudit	July 24
AWSIoTDeviceDefenderDailyAudit	July 23
AWSIoTDeviceDefenderDailyAudit	July 22
AWSIoTDeviceDefenderDailyAudit	July 21

3. In the **Non-compliant checks** section, under **Check name**, choose the audit check that you're interested in.



AWS IoT > Device Defender > Audit > Audit Results > Audit Report

Audit Report

On-demand - July 28, 2020, 14:14:18 (UTC-0700)

Audit findings

Audit task ID
40c1204d7be8bb0d33682ef35c144231

Started at
July 28, 2020, 14:14:18 (UTC-0700)

Non-compliant checks (1 of 14)

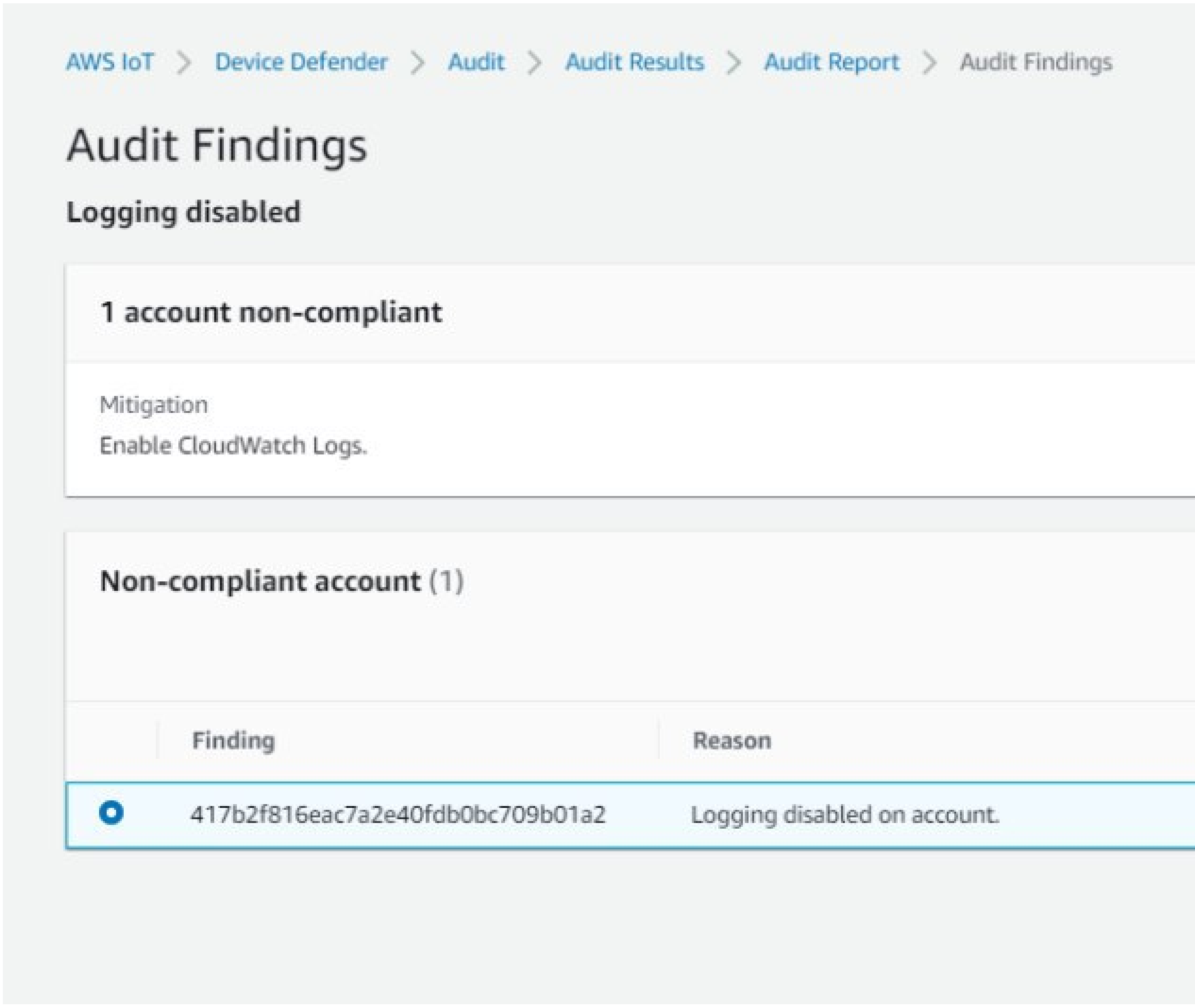
Check name	Severity	Non-compliant resources	% Resources	Mit
Logging disabled	Low	1	100%	Log

Compliant checks (13 of 14)

Check name	Severity	Scanned
Authenticated Cognito role overly permissive	Critical	0
CA certificate key quality	Critical	0
CA certificate revoked but device certificates still active	Critical	0
Device certificate key quality	Critical	0
Device certificate shared	Critical	0
IoT policies overly permissive	Critical	0
Role alias overly permissive	Critical	0
Unauthenticated Cognito role overly permissive	Critical	0
Conflicting MQTT client IDs	High	0
CA certificate expiring	Medium	0
Device certificate expiring	Medium	0
Revoked device certificate still active	Medium	0
Role alias allows access to unused services	Medium	0

4. On the audit check details screen, if there are findings you don't want to see, select the option button next to the finding. Next, choose **Actions**, and then choose the amount of time you'd like your audit finding suppression to persist.

   Note
   In the console, you can select *1 week*, *1 month*, *3 months*, *6 months*, or *Indefinitely* as expiration dates for your audit finding suppression. If you want to set a specific expiration date, you can do so only in the CLI or API. Audit finding suppressions can also be canceled anytime regardless of expiration date.


AWS IoT > Device Defender > Audit > Audit Results > Audit Report > Audit Findings

Audit Findings

Logging disabled

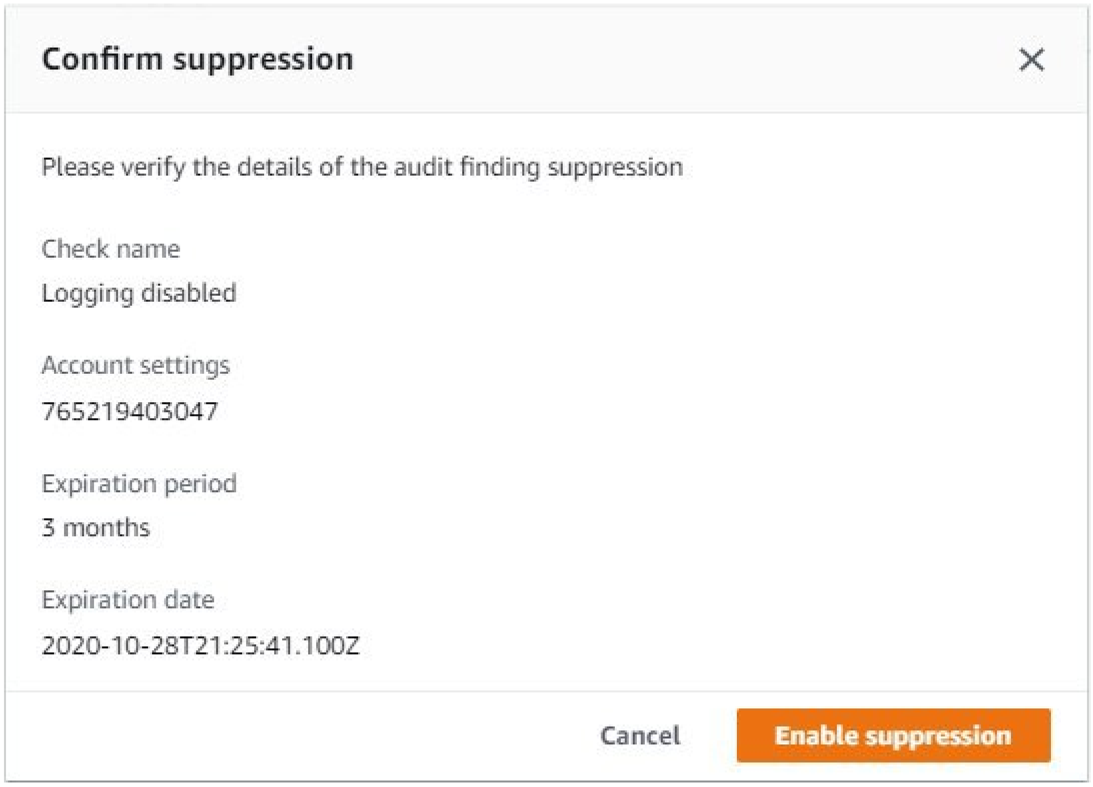
1 account non-compliant

Mitigation
Enable CloudWatch Logs.

Non-compliant account (1)

	Finding	Reason
●	417b2f816eac7a2e40fdb0bc709b01a2	Logging disabled on account.

5. Confirm the suppression details, and then choose **Enable suppression**.



Confirm suppression

Please verify the details of the audit finding suppression

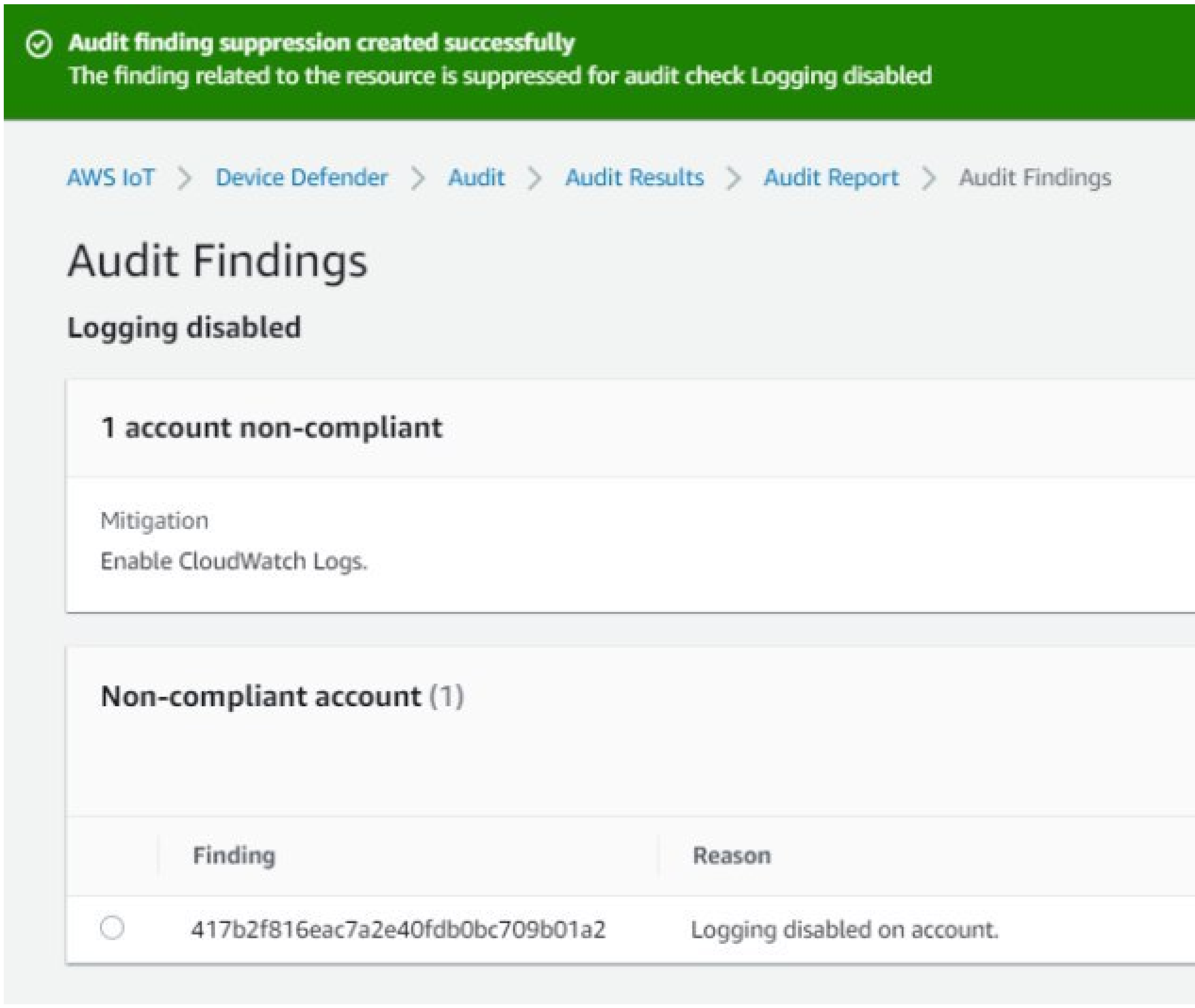
Check name
Logging disabled

Account settings
765219403047

Expiration period
3 months

Expiration date
2020-10-28T21:25:41.100Z

6. After you've created the audit finding suppression, a banner appears confirming your audit finding suppression was created.



Audit finding suppression created successfully
The finding related to the resource is suppressed for audit check Logging disabled

AWS IoT > Device Defender > Audit > Audit Results > Audit Report > Audit Findings

Audit Findings

Logging disabled

1 account non-compliant

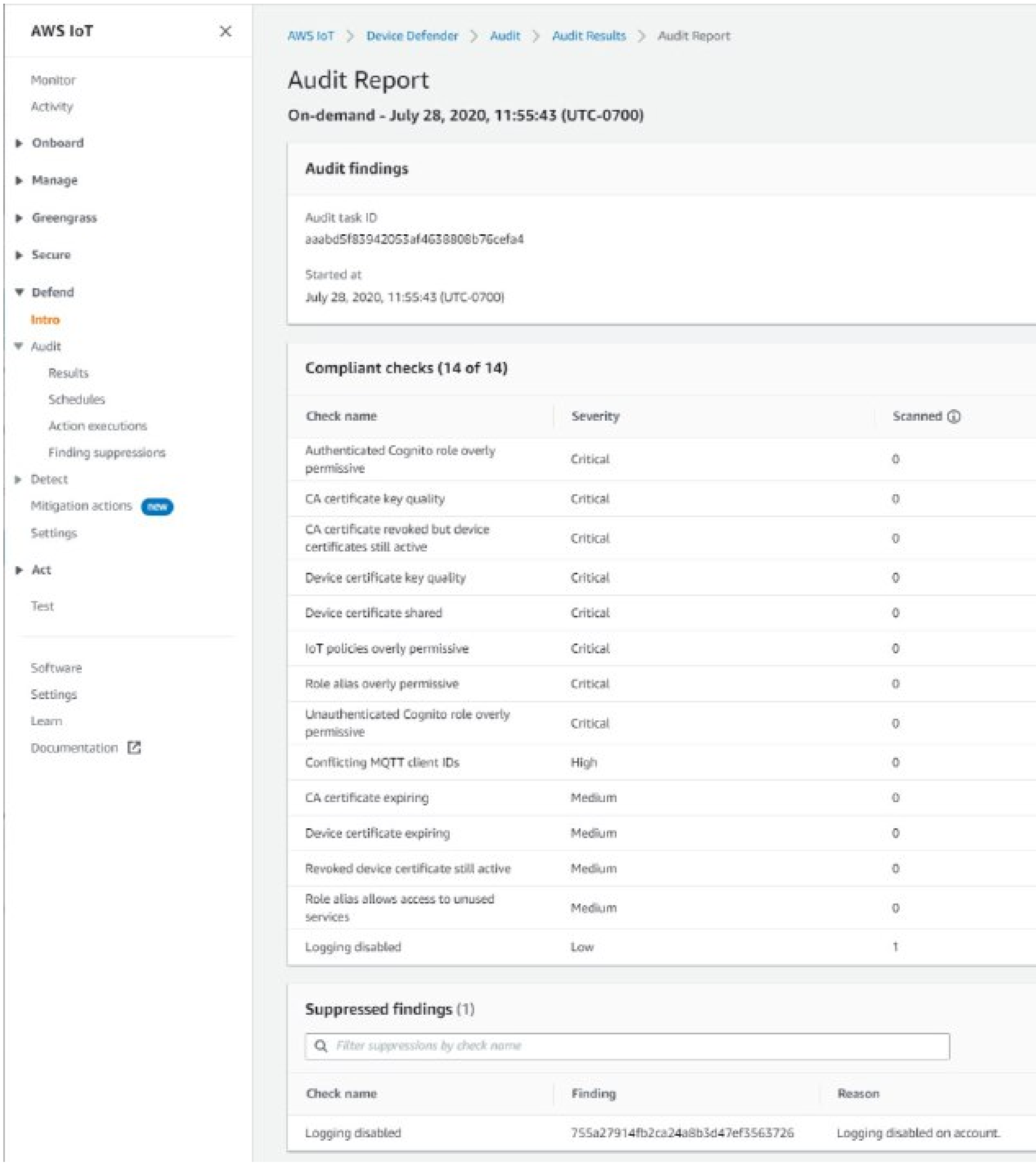
Mitigation
Enable CloudWatch Logs.

Non-compliant account (1)

	Finding	Reason
○	417b2f816eac7a2e40fdb0bc709b01a2	Logging disabled on account.

To view your suppressed findings in an audit report

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Results**.
2. Select an audit report you'd like to review.
3. In the **Suppressed findings** section, view which audit findings have been suppressed for your chosen audit report.



Audit Report

On-demand - July 28, 2020, 11:55:43 (UTC-0700)

Audit findings

Audit task ID
aaabd5f83942053af4638808b76cefa4

Started at
July 28, 2020, 11:55:43 (UTC-0700)

Compliant checks (14 of 14)

Check name	Severity	Scanned
Authenticated Cognito role overly permissive	Critical	0
CA certificate key quality	Critical	0
CA certificate revoked but device certificates still active	Critical	0
Device certificate key quality	Critical	0
Device certificate shared	Critical	0
IoT policies overly permissive	Critical	0
Role alias overly permissive	Critical	0
Unauthenticated Cognito role overly permissive	Critical	0
Conflicting MQTT client IDs	High	0
CA certificate expiring	Medium	0
Device certificate expiring	Medium	0
Revoked device certificate still active	Medium	0
Role alias allows access to unused services	Medium	0
Logging disabled	Low	1

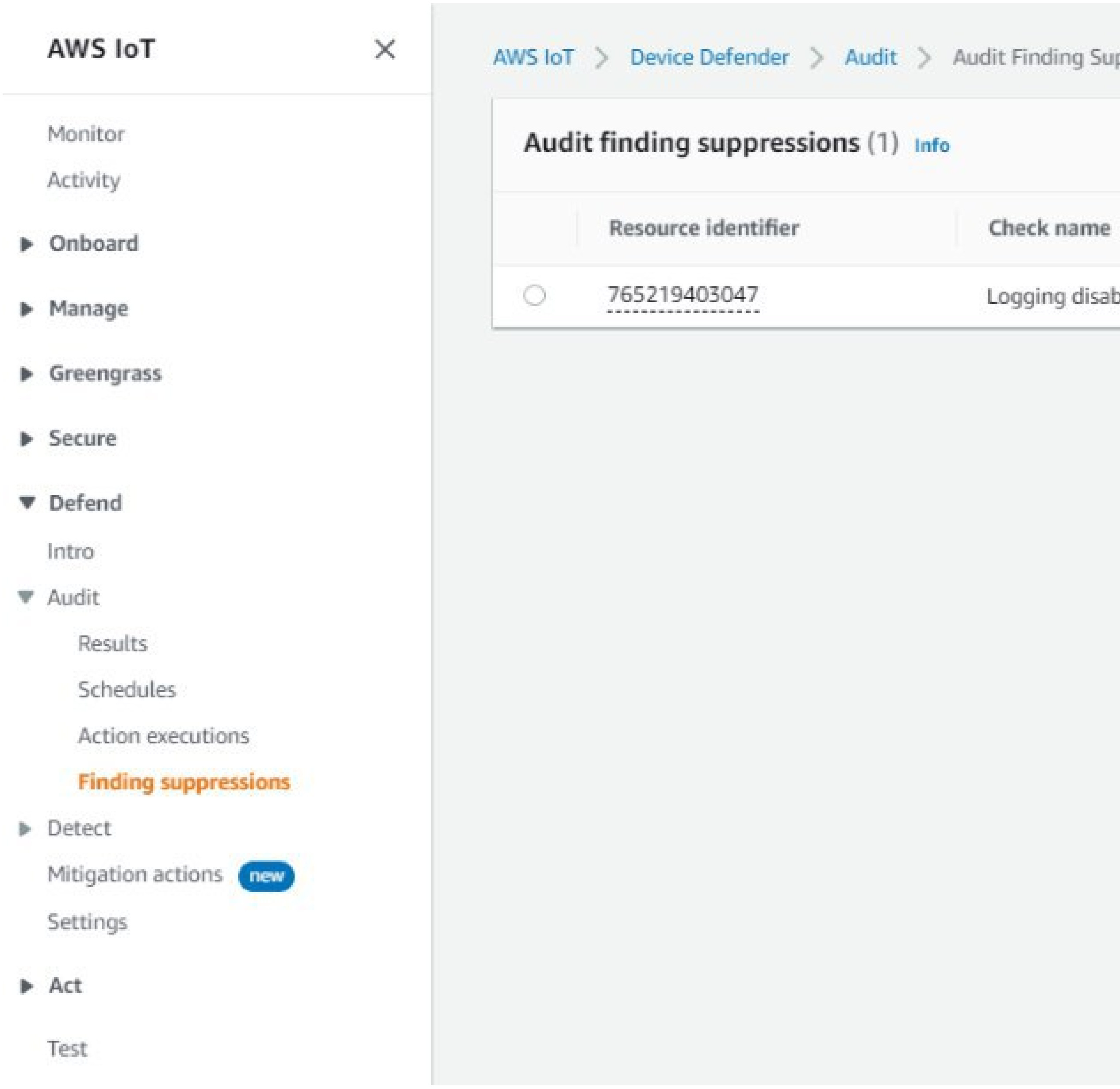
Suppressed findings (1)

Filter suppressions by check name

Check name	Finding	Reason
Logging disabled	755a27914fb2ca24a8b3d47ef3563726	Logging disabled on account.

To list your audit finding suppressions

- In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.

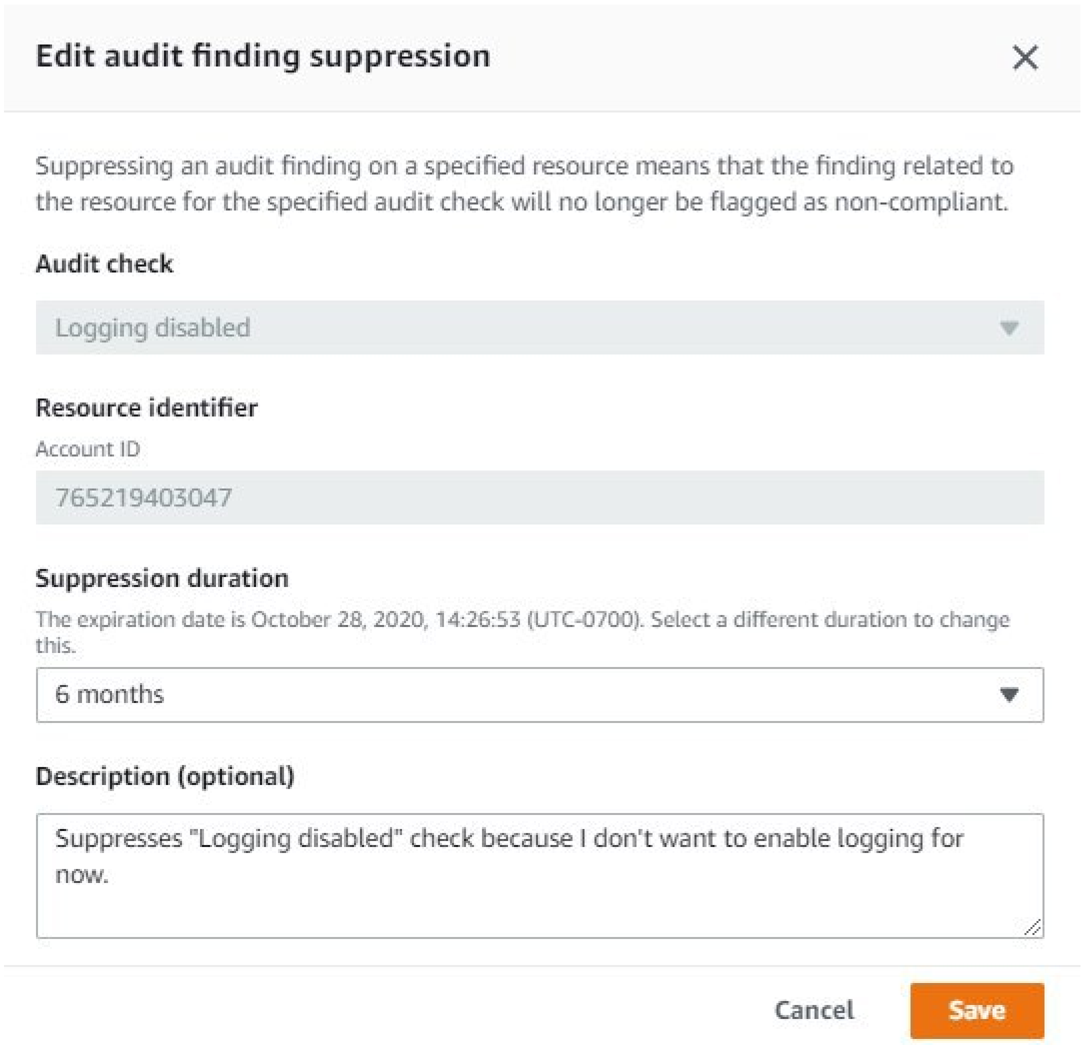


Audit finding suppressions (1)

	Resource identifier	Check name
○	765219403047	Logging disab

To edit your audit finding suppression

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.
2. Select the option button next to the audit finding suppression you'd like to edit. Next, choose **Actions**, **Edit**.
3. On the **Edit audit finding suppression** window, you can change the **Suppression duration** or **Description (optional)**.



Suppressing an audit finding on a specified resource means that the finding related to the resource for the specified audit check will no longer be flagged as non-compliant.

Audit check: Logging disabled

Resource identifier

Account ID: 765219403047

Suppression duration

The expiration date is October 28, 2020, 14:26:53 (UTC-0700). Select a different duration to change this.

6 months

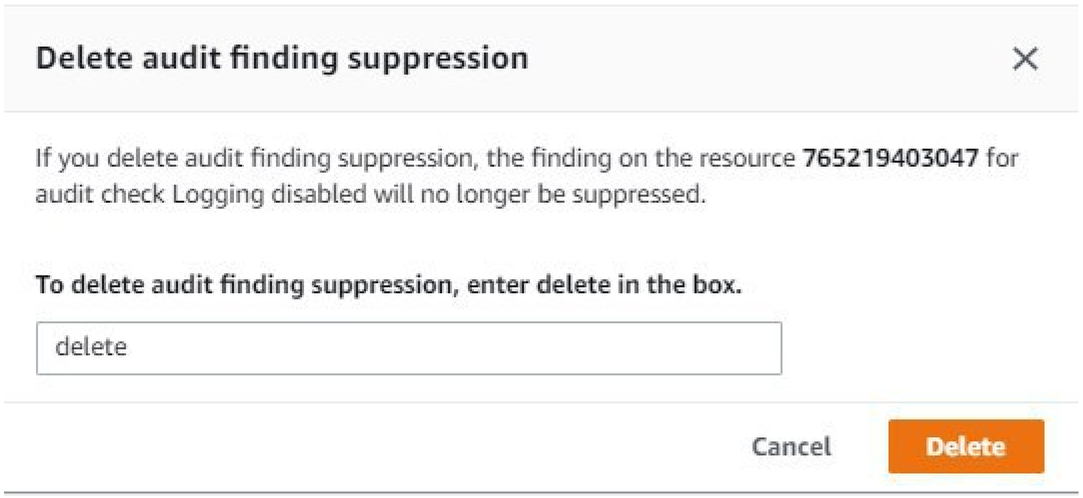
Description (optional)

Suppresses "Logging disabled" check because I don't want to enable logging for now.

4. After you've made your changes, choose **Save**. The **Finding suppressions** window opens.

To delete an audit finding suppression

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Audit**, **Finding suppressions**.
2. Select the option button next to the audit finding suppression you'd like to delete, and then choose **Actions**, **Delete**.
3. On the **Delete audit finding suppression** window, enter `delete` in the text box to confirm your deletion, and then choose **Delete**. The **Finding suppressions** window opens.



Delete audit finding suppression

If you delete audit finding suppression, the finding on the resource **765219403047** for audit check Logging disabled will no longer be suppressed.

To delete audit finding suppression, enter delete in the box.

delete

How to use audit finding suppressions in the CLI

You can use the following CLI commands to create and manage audit finding suppressions.

- create-audit-suppression
- describe-audit-suppression
- update-audit-suppression
- delete-audit-suppression
- list-audit-suppressions

The `resource-identifier` you input depends on the `check-name` you're suppressing findings for. The following table details which checks require which `resource-identifier` for creating and editing suppressions.

> **Note**
> The suppression commands do not indicate turning off an audit. Audits will still run on your AWS IoT devices. Suppressions are only applicable to the audit findings.

check-name	**resource-identifier**
AUTHENTICATE_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	cognitoIdentityPoolId
CA_CERT_APPROACHING_EXPIRATION_CHECK	caCertificateId
CA_CERTIFICATE_KEY_QUALITY_CHECK	caCertificateId
CONFLICTING_CLIENT_IDS_CHECK	clientId
DEVICE_CERT_APPROACHING_EXPIRATION_CHECK	deviceCertificateId
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK	deviceCertificateId
DEVICE_CERTIFICATE_SHARED_CHECK	deviceCertificateId
IOT_POLICY_OVERLY_PERMISSIVE_CHECK	policyVersionIdentifier
IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK	roleAliasArn

check-name	resource-identifier
IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK	roleAliasArn
LOGGING_DISABLED_CHECK	account
REVOKED_CA_CERT_CHECK	caCertificateId
REVOKED_DEVICE_CERT_CHECK	deviceCertificateId
UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	cognitoIdentityPoolId

To create and apply an audit finding suppression

The following procedure shows you how to create an audit finding suppression in the AWS CLI.

- Use the `create-audit-suppression` command to create an audit finding suppression. The following example creates an audit finding suppression for AWS account *123456789012* on the basis of the check **Logging disabled**.

```
aws iot create-audit-suppression \
    --check-name LOGGING_DISABLED_CHECK \
    --resource-identifier account=123456789012 \
    --client-request-token 28ac32c3-384c-487a-a368-c7bbd481f554 \
    --suppress-indefinitely \
    --description "Suppresses logging disabled check because I don't want to enable
 logging for now."
```

There is no output for this command.

Audit finding suppressions APIs

The following APIs can be used to create and manage audit finding suppressions.

- CreateAuditSuppression
- DescribeAuditSuppression
- UpdateAuditSuppression
- DeleteAuditSuppression
- ListAuditSuppressions

To filter *for* specific audit findings, you can use the ListAuditFindings API.

Detect

AWS IoT Device Defender Detect lets you identify unusual behavior that might indicate a compromised device by monitoring the behavior of your devices. Using a combination of cloud-side metrics (from AWS IoT) and device-side metrics (from agents that you install on your devices) you can detect:

- Changes in connection patterns.
- Devices that communicate to unauthorized or unrecognized endpoints.
- Changes in inbound and outbound device traffic patterns.

You create security profiles, which contain definitions of expected device behaviors, and assign them to a group of devices or to all the devices in your fleet. AWS IoT Device Defender Detect uses these security profiles to detect anomalies and send alarms through Amazon CloudWatch metrics and Amazon Simple Notification Service notifications.

AWS IoT Device Defender Detect can detect security issues frequently found in connected devices:

- Traffic from a device to a known malicious IP address or to an unauthorized endpoint that indicates a potential malicious command and control channel.
- Anomalous traffic, such as a spike in outbound traffic, that indicates a device is participating in a DDoS.
- Devices with remote management interfaces and ports that are remotely accessible.
- A spike in the rate of messages sent to your account (for example, from a rogue device that can result in excessive per-message charges).

Use cases:

Measure attack surface

> You can use AWS IoT Device Defender Detect to measure the attack surface of your devices. For example, you can identify devices with service ports that are often the target of attack campaigns (telnet service running on ports 23/2323, SSH service running on port 22, HTTP/S services running on ports 80/443/8080/8081). While these service ports might have legitimate reasons to be used on the devices, they are also usually part of the attack surface for adversaries and carry associated risks. After AWS IoT Device Defender Detect alarms you to the attack surface, you can minimize it (by eliminating unused network services) or run additional assessments to identify security weaknesses (for example, telnet configured with common, default, or weak passwords).

Detect device behavioral anomalies with possible security root causes

> You can use AWS IoT Device Defender Detect to alarm you to unexpected device behavioral metrics (the number of open ports, number of connections, an unexpected open port, connections to unexpected IP addresses) that might indicate a security breach. For example, a higher than expected number of TCP connections might indicate a device is being used for a DDoS attack. A process listening on a port other than the one you expect might indicate a backdoor installed on a device for remote control. You can use AWS IoT Device Defender Detect to probe the health of your device fleets and verify your security assumptions (for example, no device is listening on port 23 or 2323).
>
> You can enable machine learning (ML)-based threat detection to automatically identify potential threats.

Detect an incorrectly configured device

> A spike in the number or size of messages sent from a device to your account might indicate an incorrectly configured device. Such a device might increase your per-message charges. Similarly, a device with many authorization failures might require a reconfigured policy.

Monitoring the behavior of unregistered devices

AWS IoT Device Defender Detect makes it possible to identify unusual behaviors for devices that are not registered in the AWS IoT registry. You can define security profiles that are specific to one of the following target types:

- All devices
- All registered devices (things in the AWS IoT registry)
- All unregistered devices
- Devices in a thing group

A security profile defines a set of expected behaviors for devices in your account and specifies the actions to take when an anomaly is detected. Security profiles should be attached to the most specific targets to give you granular control over which devices are being evaluated against that profile.

Unregistered devices must provide a consistent MQTT client identifier or thing name (for devices that report device metrics) over the device lifetime so all violations and metrics are attributed to the same device.

> **Important**
> Messages reported by devices are rejected if the thing name contains control characters or if the thing name is longer than 128 bytes of UTF-8 encoded characters.

Concepts

metric

AWS IoT Device Defender Detect uses metrics to detect anomalous behavior of devices. AWS IoT Device Defender Detect compares the reported value of a metric with the expected value you provide. These metrics can be taken from two sources: cloud-side metrics and device-side metrics. There are 17 total metrics, 6 of which are supported by ML Detect. For a list of supported metrics for ML Detect, see Supported metrics (p. 840).

Abnormal behavior on the AWS IoT network is detected by using cloud-side metrics such as the number of authorization failures, or the number or size of messages a device sends or receives through AWS IoT.

AWS IoT Device Defender Detect can also collect, aggregate, and monitor metrics data generated by AWS IoT devices (for example, the ports a device is listening on, the number of bytes or packets sent, or the device's TCP connections).

You can use AWS IoT Device Defender Detect with cloud-side metrics alone. To use device-side metrics, you must first deploy the AWS IoT SDK on your AWS IoT connected devices or device gateways to collect the metrics and send them to AWS IoT. See Sending metrics from devices (p. 859).

Security Profile

A Security Profile defines anomalous behaviors for a group of devices (a thing group (p. 179)) or for all devices in your account, and specifies which actions to take when an anomaly is detected. You can use the AWS IoT console or API commands to create a Security Profile and associate it with a group of devices. AWS IoT Device Defender Detect starts recording security-related data and uses the behaviors defined in the Security Profile to detect anomalies in the behavior of the devices.

behavior

A behavior tells AWS IoT Device Defender Detect how to recognize when a device is doing something anomalous. Any device action that doesn’t match a behavior triggers an alert. A Rules Detect behavior consists of a metric and an absolute-value or statistical threshold with an operator (for example, less than or equal to, greater than or equal to), which describe the expected device behavior. An ML Detect behavior consists of a metric and an ML Detect configuration, which set an ML model to learn the normal behavior of devices.

ML model

An ML model is a machine learning model created to monitor each behavior a customer configures. The model trains on metric data patterns from targeted device groups and generates three anomaly confidence thresholds (high, medium, and low) for the metric-based behavior. It inferences anomalies based on ingested metric data at the device level. In the context of ML Detect, one ML model is created to evaluate one metric-based behavior. For more information, see ML Detect reference (p. 839)

confidence level

ML Detect supports three confidence levels: `High`, `Medium`, and `Low`. `High` confidence means low sensitivity in anomalous behavior evaluation and frequently a lower number of alarms. `Medium` confidence means medium sensitivity and `Low` confidence means high sensitivity and frequently a higher number of alarms.

dimension

You can define a dimension to adjust the scope of a behavior. For example, you can define a topic filter dimension that applies a behavior to MQTT topics that match a pattern. For information about defining a dimension for use in a Security Profile, see CreateDimension.

alarm

When an anomaly is detected, an alarm notification can be sent through a CloudWatch metric (see Using AWS IoT metrics (p. 318)) or an SNS notification. An alarm notification is also displayed in the AWS IoT console along with information about the alarm, and a history of alarms for the device. An alarm is also sent when a monitored device stops exhibiting anomalous behavior or when it had been causing an alarm but stops reporting for an extended period.

alarm suppression

Manage Detect alarm SNS notifications by setting behavior notification to `on` or `suppressed`. Suppressing alarms doesn't stop Detect from performing device behavior evaluations; Detect continues to flag anomalous behaviors as violation alarms. However, suppressed alarms wouldn't be forwarded for SNS notification. They can only be accessed through the AWS IoT console or API.

Behaviors

A Security Profile contains a set of behaviors. Each behavior contains a metric that specifies the normal behavior for a group of devices or for all devices in your account. Behaviors fall into two categories: Rules Detect behaviors and ML Detect behaviors. With Rules Detect behaviors, you define how your devices should behave whereas ML Detect uses historical device data to define how your devices should behave.

See Cloud-side metrics (p. 861), Device-side metrics (p. 846) and CreateSecurityProfile for more information.

The following describes some of the fields that are used in the definition of a `behavior`:

Common to Rules Detect and ML Detect

`name`

The name for the behavior.

`metric`

The name of the metric used (that is, what is measured by the behavior).

`consecutiveDatapointsToAlarm`

If a device is in violation of the behavior for the specified number of consecutive data points, an alarm occurs. If not specified, the default is 1. This appears as **Datapoints required to trigger alarm** for ML Detect in the console.

`consecutiveDatapointsToClear`

If an alarm has occurred and the offending device is no longer in violation of the behavior for the specified number of consecutive data points, the alarm is cleared. If not specified, the default is 1. This appears as **Datapoints required to clear alarm** for ML Detect in the console.

`threshold type`

A Security Profile can be one of two threshold types: ML Detect or Rules Detect. ML Security Profiles automatically detect device-level operational and security anomalies across your fleet by learning from past data. Rule-based Security Profiles require that you manually set static alarms for your device behaviors.

`alarm suppressions`

Manage Detect alarm SNS notifications by setting behavior notification to `on` or `suppressed`. Suppressing alarms doesn't stop Detect from performing device behavior evaluations; Detect continues to flag anomalous behaviors as violation alarms. However, suppressed alarms wouldn't be forwarded for SNS notification. They can only be accessed through the AWS IoT console or API.

Rules Detect

`dimension`

You can define a dimension to adjust the scope of a behavior. For example, you can define a topic filter dimension that applies a behavior to MQTT topics that match a pattern. To define a dimension for use in a Security Profile, see CreateDimension. Applies to Rules Detect only.

`criteria`

The criteria that determine if a device is behaving normally in regard to the `metric`.

`comparisonOperator`

The operator that relates the thing measured (`metric`) to the criteria (`value` or `statisticalThreshold`).

Possible values are: "less-than", "less-than-equals", "greater-than", "greater-than-equals", "in-cidr-set", "not-in-cidr-set", "in-port-set", and "not-in-port-set". Not all operators are valid for every metric. Operators for CIDR sets and ports are only for use with metrics involving such entities.

`value`

The value to be compared with the `metric`. Depending on the type of metric, this should contain a `count` (a value), `cidrs` (a list of CIDRs), or `ports` (a list of ports).

`statisticalThreshold`

The statistical threshold by which a behavior violation is determined. This field contains a `statistic` field that has the following possible values: "p0", "p0.1", "p0.01", "p1", "p10", "p50", "p90", "p99", "p99.9", "p99.99", or "p100".

This `statistic` indicates a percentile. It resolves to a value by which compliance with the behavior is determined. Metrics are collected one or more times over the specified duration (`durationSeconds`) from all reporting devices associated with this Security Profile, and percentiles are calculated based on that data. After that, measurements are collected for a device and accumulated over the same duration. If the resulting value for the device falls above or below (`comparisonOperator`) the value associated with the percentile specified, then the device is considered to be in compliance with the behavior. Otherwise, the device is in violation of the behavior.

A percentile indicates the percentage of all the measurements considered that fall below the associated value. For example, if the value associated with "p90" (the 90th percentile) is 123, then 90% of all measurements were below 123.

`durationSeconds`

Use this to specify the period of time over which the behavior is evaluated, for those criteria that have a time dimension (for example, `NUM_MESSAGES_SENT`). For a `statisticalThreshhold` metric comparison, this is the time period during which measurements are collected for all devices to determine the `statisticalThreshold` values, and then for each device to determine how its behavior ranks in comparison.

ML Detect

`ML Detect confidence`

ML Detect supports three confidence levels: `High`, `Medium` and `Low`. `High` confidence means low sensitivity in anomalous behavior evaluation and frequently a lower number of alarms. `Medium` confidence means medium sensitivity and `Low` confidence means high sensitivity and frequently a higher number of alarms.

ML Detect reference

ML Detect is in preview release for AWS IoT Device Defender and is subject to change.

With machine learning (ML) Detect, you can create Security Profiles that uses machine learning to learn expected device behaviors by automatically creating models based on historical device data, and assign these profiles to a group of devices or all the devices in your fleet. AWS IoT Device Defender then identifies anomalies and triggers alarms using the ML models.

For steps on how to get started with ML Detect, see ML Detect guide (p. 735).

This chapter contains the following sections:

How ML Detect works

With Rules Detect, you can create static alarms to identify operational and security anomalies across seven cloud-side metrics (p. 861) and ten device-side metrics (p. 846). ML Detect takes this a step further by automatically learning device behaviors with machine learning models using data across six cloud-side metrics (p. 840) from a trailing 14 day period. It then retrains the models each day to refresh the device behaviors based on the latest trailing 14 days after initial models are built. ML Detect monitors and identifies anomalous datapoints for these metrics with the ML models and triggers an alarm if an anomaly is detected. The key benefits are that it automatically detects operational and security anomalies across fleet devices and it dynamically updates expected device behaviors based on new data trends to reduce false positive rates.

While ML Detect is building its initial model, it requires 14 days and a minimum of 25,000 datapoints per metric to generate the model. Afterwards, it updates the model every day as long as the minimum 25,000 metric datapoints per model are met. If the minimum metric datapoint requirement is not met, ML Detect will attempt to build the model on the next day. It will retry daily for 30 days before discontinuing the model.

Minimum requirements

For training and creating the initial ML model, ML Detect has two minimum requirements.

Minimum training period

It takes 14 days for the initial models to be built. After that, the model refreshes every day with a 14-day trailing window of metric data.

Minimum datapoints

The minimum required datapoints to build an ML model is 25,000 datapoints per metric for the last 14 days. For ongoing training and refreshing of the model, ML Detect requires the minimum datapoints met from monitored devices. It's roughly the equivalent of the setups below:

- 60 devices connecting and having activity on AWS IoT at 45-minute intervals.
- 40 devices at 30-minute intervals.
- 15 devices at 10-minute intervals.
- 7 to 8 devices at 5-minute intervals.

Devices attached to a Security Profile target

In order for data collection to progress, you must have things in the target that the SecurityProfile is attached to.

After the initial model is created, ML models refresh on a daily basis and require the same datapoint minimums.

Supported metrics

You can use the following cloud-side metrics with ML Detect:

- Authorization failures (aws:authorization-failures) (p. 864)
- Connection attempts (aws:num-connection-attempts) (p. 865)
- Disconnects (aws:num-disconnects) (p. 866)
- Message size (aws:message-byte-size) (p. 861)
- Messages sent (aws:num-messages-sent) (p. 862)
- Messages received (num-messages-received) (p. 863)

Service quotas

For information about ML Detect service quotas and limits, see AWS IoT Device Defender endpoints and quotas.

ML Detect CLI commands

You can use the following CLI commands to create and manage ML Detect.

- create-security-profile
- attach-security-profile
- list-security-profiles
- describe-security-profile
- update-security-profile

- delete-security-profile
- get-behavior-model-training-summaries
- list-active-violations
- list-violation-events

ML Detect APIs

The following APIs can be used to create and manage ML Detect security profiles.

- CreateSecurityProfile
- AttachSecurityProfile
- ListSecurityProfiles
- DescribeSecurityProfile
- UpdateSecurityProfile
- DeleteSecurityProfile
- GetBehaviorModelTrainingSummaries
- ListActiveViolations
- ListViolationEvents

Custom metrics

With AWS IoT Device Defender custom metrics, you can define and monitor metrics that are unique to your fleet or use case, such as number of devices connected to Wi-Fi gateways, charge levels for batteries, or number of power cycles for smart plugs. Custom metric behaviors are defined in Security Profiles, which specify expected behaviors for a group of devices (a thing group) or for all devices. You can monitor behaviors by setting up alarms, which you can use to detect and respond to issues that are specific to the devices.

This chapter contains the following sections:

How to use custom metrics in the console

Tutorials

AWS IoT Device Defender Agent SDK (Python)

To get started, download the AWS IoT Device Defender Agent SDK (Python) sample agent. The agent gathers the metrics and publishes reports. Once your device-side metrics are publishing, you can view the metrics being collected and determine thresholds for setting up alarms. Instructions for setting up the device agent are available on the AWS IoT Device Defender Agent SDK (Python) Readme. For more information, see AWS IoT Device Defender Agent SDK (Python).

Create a custom metric and add it to a Security Profile

The following procedure shows you how to create a custom metric in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. On the **Custom metrics** page, choose **Create**.
3. On the **Create custom metric** page, do the following.

    1. Under **Name**, enter a name for your custom metric. You can't modify this name after you create the custom metric.
    2. Under **Display name (optional)**, you can enter a friendly name for your custom metric. It doesn't have to be unique and it can be modified after creation.
    3. Under **Type**, choose the type of metric you'd like to monitor. Metric types include **string-list**, **ip-address-list**, **number-list**, and **number**. The type can't be modified after creation.
    4. Under **Tags**, you can select tags to be associated with the resource.

    When you're done, choose **Confirm**.
4. After you've created your custom metric, the **Custom metrics** page appears, where you can see your newly created custom metric.
5. Next, you need to add your custom metric to a Security Profile. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Security profiles**.
6. Choose the Security Profile you'd like to add your custom metric to.
7. Choose **Actions**, **Edit**.
8. Choose **Additional Metrics to retain**, and then choose your custom metric. Choose **Next** on the following screens until you reach the **Confirm** page. Choose **Save** and **Continue**. After your custom metric has been successfully added, the Security Profile details page appears.

    Note
    Percentile statistics are not available for metrics when any of the metric values are negative numbers.

View custom metric details

The following procedure shows you how to view a custom metric's details in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the **Metric name** of the custom metric you'd like to view the details of.

Update a custom metric

The following procedure shows you how to update a custom metric in the console.

1. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the option button next to the custom metric you'd like to update. Then, for **Actions**, choose **Edit**.
3. On the **Update custom metric** page, you can edit the display name and remove or add tags.
4. After you're done, choose **Update**. The **Custom metrics** page.

Delete a custom metric

The following procedure shows you how to delete a custom metric in the console.

1. First, remove your custom metric from any Security Profile it's referenced in. You can view which Security Profiles contain your custom metric on your custom metric details page. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
2. Choose the custom metric you'd like to remove. Remove the custom metric from any Security Profile listed under **Security Profiles** on the custom metric details page.
3. In the AWS IoT console, in the navigation pane, expand **Defend**, and then choose **Detect**, **Metrics**.
4. Choose the option button next to the custom metric you'd like to delete. Then, for **Actions**, choose **Delete**.
5. On the **Are you sure you want to delete custom metric?** message, choose **Delete custom metric**.

   Warning
   After you've deleted a custom metric, you lose all data associated with the metric. This action can't be undone.

How to use custom metrics from the CLI

Tutorials

AWS IoT Device Defender Agent SDK (Python)

To get started, download the AWS IoT Device Defender Agent SDK (Python) sample agent. The agent gathers the metrics and publishes reports. After your device-side metrics are publishing, you can view the metrics being collected and determine thresholds for setting up alarms. Instructions for setting up the device agent are available on the AWS IoT Device Defender Agent SDK (Python) Readme. For more information, see AWS IoT Device Defender Agent SDK (Python).

Create a custom metric and add it to a Security Profile

The following procedure shows you how to create a custom metric and add it to a Security Profile from the CLI.

1. Use the `create-custom-metric` command to create your custom metric. The following example creates a custom metric that measures battery percentage.

```
aws iot create-custom-metric \
    --metric-name "batteryPercentage" \
    --metric-type "number" \
    --display-name "Remaining battery percentage." \
    --region us-east-1
    --client-request-token "02ccb92b-33e8-4dfa-a0c1-35b181ed26b0" \
```

   Output:

```
{
    "metricName": "batteryPercentage",
    "metricArn": "arn:aws:iot:us-east-1:1234564789012:custommetric/batteryPercentage"
}
```

2. After you've created your custom metric, you can either add the custom metric to an existing profile using `update-security-profile` or create a new security profile to add the custom metric to using `create-security-profile`. Here, we create a new security profile called *`batteryUsage`* to add our new *`batteryPercentage`* custom metric to. We also add a Rules Detect metric called *`cellularBandwidth`*.

```
aws iot create-security-profile \
    --security-profile-name batteryUsage \
    --security-profile-description "Shows how much battery is left in percentile."  \
    --behaviors "[{\"name\":\"great-than-75\",\"metric\":\"batteryPercentage\",
\"criteria\":{\"comparisonOperator\":\"greater-than\",\"value\":{\"number
\":75},\"consecutiveDatapointsToAlarm\":5,\"consecutiveDatapointsToClear
\":1}},{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size\",
\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]" \
    --region us-east-1
```

Output:

```
{
    "securityProfileArn": "arn:aws:iot:us-east-1:1234564789012:securityprofile/
batteryUsage",
    "securityProfileName": "batteryUsage"
}
```

Note
Percentile statistics are not available for metrics when any of the metric values are negative numbers.

View custom metric details

The following procedure shows you how to view the details for a custom metric from the CLI.

- Use the `list-custom-metrics` command to view all of your custom metrics.

```
aws iot list-custom-metrics \
    --region us-east-1
```

The output of this command looks like the following.

```
{
    "metricNames": [
        "batteryPercentage"
    ]
}
```

Update a custom metric

The following procedure shows you how to update a custom metric from the CLI.

- Use the `update-custom-metric` command to update a custom metric. The following example updates the `display-name`.

```
aws iot update-custom-metric \
    --metric-name batteryPercentage \
    --display-name 'remaining battery percentage on device' \
    --region us-east-1
```

The output of this command looks like the following.

```
{
    "metricName": "batteryPercentage",
    "metricArn": "arn:aws:iot:us-east-1:1234564789012:custommetric/batteryPercentage",
    "metricType": "number",
    "displayName": "remaining battery percentage on device",
    "creationDate": "2020-11-17T23:01:35.110000-08:00",
    "lastModifiedDate": "2020-11-17T23:02:12.879000-08:00"
}
```

Delete a custom metric

The following procedure shows you how to delete a custom metric from the CLI.

1. To delete a custom metric, first remove it from any Security Profiles that it's attached to. Use the `list-security-profiles` command to view Security Profiles with a certain custom metric.
2. To remove a custom metric from a Security Profile, use the `update-security-profiles` command. Enter all information that you want to keep, but exclude the custom metric.

```
aws iot update-security-profile \
  --security-profile-name batteryUsage \
  --behaviors "[{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]"
```

The output of this command looks like the following.

```
{
  "behaviors": [{\"name\":\"cellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}],
  "securityProfileName": "batteryUsage",
  "lastModifiedDate": 2020-11-17T23:02:12.879000-09:00,
  "securityProfileDescription": "Shows how much battery is left in percentile.",
  "version": 2,
  "securityProfileArn": "arn:aws:iot:us-east-1:1234564789012:securityprofile/
batteryUsage",
  "creationDate": 2020-11-17T23:02:12.879000-09:00
}
```

3. After the custom metric is detached, use the `delete-custom-metric` command to delete the custom metric.

```
aws iot delete-custom-metric  \
  --metric-name batteryPercentage \
  --region us-east-1
```

The output of this command looks like the following

```
HTTP 200
```

Custom metrics CLI commands

You can use the following CLI commands to create and manage custom metrics.

- create-custom-metric
- describe-custom-metric
- list-custom-metrics
- update-custom-metric
- delete-custom-metric
- list-security-profiles

Custom metrics APIs

The following APIs can be used to create and manage custom metrics.

- CreateCustomMetric
- DescribeCustomMetric
- ListCustomMetrics
- UpdateCustomMetric
- DeleteCustomMetric
- ListSecurityProfiles

Device-side metrics

When creating a Security Profile, you can specify your IoT device's expected behavior by configuring behaviors and thresholds for metrics generated by IoT devices. The following are device-side metrics, which are metrics from agents that you install on your devices.

Bytes out (aws:all-bytes-out)

The number of outbound bytes from a device during a given time period.

Use this metric to specify the maximum or minimum amount of outbound traffic that a device should send, measured in bytes, in a given period of time.

Can be used with ML Detect: No

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-bytes-out",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 4096
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 5,
    "consecutiveDatapointsToClear": 4
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-bytes-out",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 900,
    "consecutiveDatapointsToAlarm": 5,
    "consecutiveDatapointsToClear": 4
  }
}
```

Bytes in (aws:all-bytes-in)

The number of inbound bytes to a device during a given time period.

Use this metric to specify the maximum or minimum amount of inbound traffic that a device should receive, measured in bytes, in a given period of time.

Can be used with ML Detect: No

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-bytes-in",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 4096
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 3
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-bytes-in",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 3
  }
}
```

Listening TCP port count (aws:num-listening-tcp-ports)

The number of TCP ports the device is listening on.

Use this metric to specify the maximum number of authorization failures allowed for each device in a given period of time. An authorization failure occurs when a request from a device to AWS IoT is denied (for example, if a device attempts to publish to a topic for which it does not have sufficient permissions).

Can be used with ML Detect: No

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: failures

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Listening UDP port count (aws:num-listening-udp-ports)

The number of UDP ports the device is listening on.

Use this metric to specify the maximum number of authorization failures allowed for each device in a given period of time. An authorization failure occurs when a request from a device to AWS IoT is denied (for example, if a device attempts to publish to a topic for which it does not have sufficient permissions).

Can be used with ML Detect: No

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: failures

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Packets out (aws:all-packets-out)

The number of outbound packets from a device during a given time period.

Use this metric to specify the maximum or minimum amount of total outbound traffic that a device should send in a given period of time.

Can be used with ML Detect: No

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: packets

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-packets-out",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 100
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 3
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "TCP outbound traffic",
  "metric": "aws:all-packets-out",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 3
  }
}
```

Packets in (aws:all-packets-in)

The number of inbound packets to a device during a given time period.

Use this metric to specify the maximum or minimum amount of total inbound traffic that a device should receive in a given period of time.

Can be used with ML Detect: No

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: packets

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800 or 3600 seconds.

Example

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-packets-in",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 100
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Example

Example using a `statisticalThreshold`

```
{
  "name": "TCP inbound traffic",
  "metric": "aws:all-packets-in",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Destination IPs (aws:destination-ip-addresses)

A set of IP destinations.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT.

Can be used with ML Detect: No

Operators: in-cidr-set | not-in-cidr-set

Values: a list of CIDRs

Units: n/a

Example

```
{
  "name": "Denied source IPs",
  "metric": "aws:source-ip-address",
  "criteria": {
    "comparisonOperator": "not-in-cidr-set",
    "value": {
      "cidrs": [ "12.8.0.0/16", "15.102.16.0/24" ]
    }
  }
}
```

Listening TCP ports (aws:listening-tcp-ports)

The TCP ports that the device is listening on.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) TCP ports on which each device must or must not listen.

Can be used with ML Detect: No

Operators: in-port-set | not-in-port-set

Values: a list of ports

Units: n/a

Example

```
{
  "name": "Listening TCP Ports",
  "metric": "aws:listening-tcp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 443, 80 ]
    }
  }
}
```

Listening UDP ports (aws:listening-udp-ports)

The UDP ports that the device is listening on.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) UDP ports on which each device must or must not listen.

Can be used with ML Detect: No

Operators: in-port-set | not-in-port-set

Values: a list of ports

Units: n/a

Example

```
{
  "name": "Listening UDP Ports",
  "metric": "aws:listening-udp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 1025, 2000 ]
    }
  }
}
```

Established TCP connections count (aws:num-established-tcp-connections)

The number of TCP connections for a device.

Use this metric to specify the maximum or minimum number of active TCP connections that each device should have. (All TCP states)

Can be used with ML Detect: No

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: connections

Example

```
{
  "name": "TCP Connection Count",
  "metric": "aws:num-established-tcp-connections",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 3
    },
    "consecutiveDatapointsToAlarm": 3,
    "consecutiveDatapointsToClear": 3
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "TCP Connection Count",
  "metric": "aws:num-established-tcp-connections",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 900,
    "consecutiveDatapointsToAlarm": 3,
    "consecutiveDatapointsToClear": 3
  }
}
```

Device metrics document specification

Overall structure

Long name	Short name	Required	Type	Constraints	Notes
header	hed	Y	Object		Complete block required for well-formed report.
metrics	met	Y	Object		Complete block required for well-formed report.

Header block

Long name	Short name	Required	Type	Constraints	Notes
report_id	rid	Y	Integer		Monotonically increasing value. Epoch timestamp recommended.
version	v	Y	String	Major.Minor	Minor increments with addition of field. Major increments if metrics removed.

Metrics block:

TCP connections

Long name	Short name	Parent element	Required	Type	Constraints	Notes
tcp_connections	tc	metrics	N	Object		
established_connections	ec	tcp_connections	N	List<Connection>		Established TCP state
connections	cs	established_connections	N	List<Object>		
remote_addr	rad	connections	Y	Number	ip:port	IP can be IPv6 or IPv4
local_port	lp	connections	N	Number	>= 0	
local_interface	li	connections	N	String		Interface name

Long name	Short name	Parent element	Required	Type	Constraints	Notes
total	t	established_connections	N	Number	>= 0	Number of established connections

Listening TCP ports

Long name	Short name	Parent element	Required	Type	Constraints	Notes
listening_tcp_ports	tp	metrics	N	Object		
ports	pts	listening_tcp_ports	N	List<Port>	> 0	
port	pt	ports	N	Number	> 0	ports should be numbers greater than 0
interface	if	ports	N	String		Interface name
total	t	listening_tcp_ports	N	Number	>= 0	

Listening UDP ports

Long name	Short name	Parent element	Required	Type	Constraints	Notes
listening_udp_ports	up	metrics	N	Object		
ports	pts	listening_udp_ports	N	List<Port>	> 0	
port	pt	ports	N	Number	> 0	Ports should be numbers greater than 0
interface	if	ports	N	String		Interface name
total	t	listening_udp_ports	N	Number	>= 0	

Network statistics

Long name	Short name	Parent element	Required	Type	Constraints	Notes
network_stats	ns	metrics	N	Object		
bytes_in	bi	network_stats	N	Number	Delta Metric, >= 0	
bytes_out	bo	network_stats	N	Number	Delta Metric, >= 0	

Long name	Short name	Parent element	Required	Type	Constraints	Notes
packets_in	pi	network_stats	N	Number	Delta Metric, >= 0	
packets_out	po	network_stats	N	Number	Delta Metric, >= 0	

Example

The following JSON structure uses long names.

```
{
    "header": {
        "report_id": 1530304554,
        "version": "1.0"
    },
    "metrics": {
        "listening_tcp_ports": {
            "ports": [
                {
                    "interface": "eth0",
                    "port": 24800
                },
                {
                    "interface": "eth0",
                    "port": 22
                },
                {
                    "interface": "eth0",
                    "port": 53
                }
            ],
            "total": 3
        },
        "listening_udp_ports": {
            "ports": [
                {
                    "interface": "eth0",
                    "port": 5353
                },
                {
                    "interface": "eth0",
                    "port": 67
                }
            ],
            "total": 2
        },
        "network_stats": {
            "bytes_in": 29358693495,
            "bytes_out": 26485035,
            "packets_in": 10013573555,
            "packets_out": 11382615
        },
        "tcp_connections": {
            "established_connections": {
                "connections": [
                    {
                        "local_interface": "eth0",
                        "local_port": 80,
                        "remote_addr": "192.168.0.1:8000"
                    },
                    {
                        "local_interface": "eth0",
                        "local_port": 80,
                        "remote_addr": "192.168.0.1:8000"
                    }
                ],
                "total": 2
            }
         ],
         "total":3
      },
      "listening_udp_ports":{
         "ports":[
            {
               "interface":"eth0",
               "port":5353
            },
            {
               "interface":"eth0",
               "port":67
            }
         ],
         "total":2
      },
      "network_stats":{
         "bytes_in":29358693495,
         "bytes_out":26485035,
         "packets_in":10013573555,
         "packets_out":11382615
      },
      "tcp_connections":{
         "established_connections":{
            "connections":[
               {
                  "local_interface":"eth0",
                  "local_port":80,
                  "remote_addr":"192.168.0.1:8000"
               },
               {
                  "local_interface":"eth0",
                  "local_port":80,
                  "remote_addr":"192.168.0.1:8000"
               }
            ],
            "total":2
         }
      }
   },
   "custom_metrics":{
      "MyMetricOfType_Number":[
         {
            "number":1.0
         }
      ],
      "MyMetricOfType_NumberList":[
         {
            "number_list":[
               1.0,
               2.0,
               3.0
            ]
         }
      ],
      "MyMetricOfType_StringList":[
         {
            "string_list":[
                "value_1",
                "value_2"
             ]
          }
       ],
       "MyMetricOfType_IpList":[
          {
             "ip_list":[
                "172.0.0.0",
                "172.0.0.10"
             ]
          }
       ]
    }
}
```

Example Example JSON structure using short names

```
{
    "hed": {
        "rid": 1530305228,
        "v": "1.0"
    },
    "met": {
        "tp": {
            "pts": [
                {
                    "if": "eth0",
                    "pt": 24800
                },
                {
                    "if": "eth0",
                    "pt": 22
                },
                {
                    "if": "eth0",
                    "pt": 53
                }
            ],
            "t": 3
        },
        {
          "if": "eth0",
          "pt": 22
        },
        {
          "if": "eth0",
          "pt": 53
        }
      ],
      "t": 3
    },
    "up": {
      "pts": [
        {
          "if": "eth0",
          "pt": 5353
        },
        {
          "if": "eth0",
          "pt": 67
        }
      ],
      "t": 2
    },
    "ns": {
      "bi": 29359307173,
      "bo": 26490711,
      "pi": 10014614051,
      "po": 11387620
    },
    "tc": {
      "ec": {
        "cs": [
          {
            "li": "eth0",
            "lp": 80,
            "rad": "192.168.0.1:8000"
          },
          {
            "li": "eth0",
            "lp": 80,
            "rad": "192.168.0.1:8000"
          }
        ],
        "t": 2
      }
    }
  },
  "cmet": {
    "MyMetricOfType_Number": [
      {
        "number": 1
      }
    ],
    "MyMetricOfType_NumberList": [
      {
        "number_list": [
          1,
          2,
          3
        ]
      }
    ],
    "MyMetricOfType_StringList": [
      {
        "string_list": [
          "value_1",
          "value_2"
        ]
      }
    ],
    "MyMetricOfType_IpList": [
      {
        "ip_list": [
          "172.0.0.0",
          "172.0.0.10"
        ]
      }
    ]
  }
}
```

Sending metrics from devices

AWS IoT Device Defender Detect can collect, aggregate, and monitor metrics data generated by AWS IoT devices to identify devices that exhibit abnormal behavior. This section shows you how to send metrics from a device to AWS IoT Device Defender.

You must securely deploy the AWS IoT SDK version two on your AWS IoT connected devices or device gateways to collect device-side metrics. AWS IoT Device Defender provides sample agents to use as examples when you create your own. If you can't provide device metrics, you can still get limited functionality based on cloud-side metrics. See the full list of SDKs here.

There are two methods for configuring your device to publish metrics, through AWS IoT Device Client and AWS IoT Device Defender sample agent. Generally you should use AWS IoT Device Client because it provides a single agent that covers the features present in both AWS IoT Device Defender and AWS IoT Device Management. These features include jobs, secure tunneling, AWS IoT Device Defender metrics publishing and more. If you define custom metrics for your device to monitor, then you should use AWS IoT Device Defender sample agent in Python to send data.

Using the AWS IoT Device Client to publish metrics

To install AWS IoT Device Client, you can download it from Github. After you've installed the AWS IoT Device Client on the device for which you want to collect device-side data, you must configure it to send device-side metrics to AWS IoT Device Defender. Verify that the AWS IoT Device Client configuration file has the following parameters set in the `device-defender` section:

```
"device-defender":    {
      "enabled":    true,
      "interval-in-seconds": 300
  }
```

Warning
At a minimum, you should set the time interval to 300 seconds. If you set the time interval to anything less than 300 seconds, your metric data may be throttled.

After you've updated your configuration, you can create security profiles and behaviors in the AWS IoT Device Defender console to monitor the metrics that your devices publish to the cloud. You can find published metrics in the AWS IoT Core console by choosing Defend, Detect, and then Metrics.

Using the AWS IoT Device Defender sample agent to publish metrics

You can use the AWS IoT Device Defender sample agent to monitor device-side metrics and custom metrics from AWS IoT devices.

A sample device metric reporting agent is currently available in C at https://github.com/aws-samples/aws-iot-device-defender-agent-c. There is also a sample device metric reporting agent available in Python on GitHub at https://github.com/aws-samples/aws-iot-device-defender-agent-sdk-python. Custom metrics are only supported by the Python sample agent. Specifically see the greengrass_defender_agent.py file for a sample agent used with AWS IoT Greengrass devices.

To use the sample agents or create your own custom agent, you must install the AWS IoT Device SDK. To find resources for your development language, see AWS IoT Core Resources.

- All agents must create a connection to AWS IoT and publish metrics to one of these reserved AWS IoT Device Defender MQTT topics:

  ```
  $aws/things/THING_NAME/defender/metrics/json
  ```

  or

  ```
  $aws/things/THING_NAME/defender/metrics/cbor
  ```

  AWS IoT Device Defender uses one of these topics to reply with the receipt status of your metrics reports:

```
$aws/things/THING_NAME/defender/metrics/json/accepted
```

```
$aws/things/THING_NAME/defender/metrics/cbor/accepted
```

```
$aws/things/THING_NAME/defender/metrics/json/rejected
```

```
$aws/things/THING_NAME/defender/metrics/cbor/rejected
```

- To report metrics, a device must be registered as a thing in AWS IoT.
- A device should, generally, send a metric report once every five minutes. Devices are throttled so they can't make more than one metric report every five minutes.
- Devices must report current metrics. Historical metrics reporting isn't supported.
- You can optionally use Jobs (p. 537) to change how often Device Defender sends metrics. An example is included with the AWS IoT Device Defender Agent C samples. For more information, see the README.md.

Cloud-side metrics

When creating a Security Profile, you can specify your IoT device's expected behavior by configuring behaviors and thresholds for metrics generated by IoT devices. The following are cloud-side metrics, which are metrics from AWS IoT.

Message size (aws:message-byte-size)

The number of bytes in a message. Use this metric to specify the maximum or minimum size (in bytes) of each message transmitted from a device to AWS IoT.

Can be used with ML Detect: Yes

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: bytes

Example

```
{
  "name": "Max Message Size",
  "metric": "aws:message-byte-size",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 1024
    },
    "consecutiveDatapointsToAlarm": 3,
    "consecutiveDatapointsToClear": 3
  }

}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Large Message Size",
  "metric": "aws:message-byte-size",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p90"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 3,
    "consecutiveDatapointsToClear": 3
  }

}
```

An alarm occurs for a device if, during three consecutive five-minute periods, it transmits messages where the cumulative size is more than that measured for 90 percent of all other devices reporting for this Security Profile behavior.

Messages sent (aws:num-messages-sent)

The number of messages sent by a device during a given time period.

Use this metric to specify the maximum or minimum number of messages that can be sent between AWS IoT and each device in a given period of time.

Can be used with ML Detect: Yes

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: messages

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{

  "name": "Out bound message count",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 50
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 2

}
```

Example Example using a `statisticalThreshold`

```
{

  "name": "Out bound message rate",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p99"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 2
  }

}
```

Messages received (num-messages-received)

The number of messages received by a device during a given time period.

Use this metric to specify the maximum or minimum number of messages that can be received between AWS IoT and each device in a given period of time.

Can be used with ML Detect: Yes

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: messages

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{

  "name": "Out bound message count",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 50
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 2

}
```

Example Example using a `statisticalThreshold`

```
{

  "name": "Out bound message rate",
  "metric": "aws:num-messages-sent",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p99"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 2
  }
}
```

Authorization failures (aws:authorization-failures)

Use this metric to specify the maximum number of authorization failures allowed for each device in a given period of time. An authorization failure occurs when a request from a device to AWS IoT is denied (for example, if a device attempts to publish to a topic for which it does not have sufficient permissions).

Can be used with ML Detect: Yes

Unit: failures

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Authorization Failures",
  "metric": "aws:num-authorization-failures",
  "criteria": {
    "comparisonOperator": "less-than",
    "statisticalThreshold": {
      "statistic": "p50"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 2,
    "consecutiveDatapointsToClear": 1
  }
}
```

Source IP (aws:source-ip-address)

The IP address from which a device has connected to AWS IoT.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT.

Can be used with ML Detect: No

Operators: in-cidr-set | not-in-cidr-set

Values: a list of CIDRs

Units: n/a

Example

```
{
  "name": "Denied source IPs",
  "metric": "aws:source-ip-address",
  "criteria": {
    "comparisonOperator": "not-in-cidr-set",
    "value": {
      "cidrs": [ "12.8.0.0/16", "15.102.16.0/24" ]
    }
  }
}
```

Connection attempts (aws:num-connection-attempts)

The number of times a device attempts to make a connection in a given time period.

Use this metric to specify the maximum or minimum number of connection attempts for each device. Successful and unsuccessful attempts are counted.

Can be used with ML Detect: Yes

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: connection attempts

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Connection Attempts",
  "metric": "aws:num-connection-attempts",
  "criteria": {
    "comparisonOperator": "greater-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 600,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 2
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Connection Attempts",
  "metric": "aws:num-connection-attempts",
  "criteria": {
    "comparisonOperator": "greater-than",
    "statisticalThreshold": {
      "statistic": "p10"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 2
  }
}
```

Disconnects (aws:num-disconnects)

The number of times a device disconnects from AWS IoT during a given time period.

Use this metric to specify the maximum or minimum number of times a device disconnected from AWS IoT during a given time period.

Can be used with ML Detect: Yes

Operators: less-than | less-than-equals | greater-than | greater-than-equals

Value: a non-negative integer

Units: disconnects

Duration: a non-negative integer. Valid values are 300, 600, 900, 1800, or 3600 seconds.

Example

```
{
  "name": "Disconnections",
  "metric": "aws:num-disconnects",
  "criteria": {
    "comparisonOperator": "greater-than",
    "value": {
      "count": 5
    },
    "durationSeconds": 600,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 2
  }
}
```

Example Example using a `statisticalThreshold`

```
{
  "name": "Disconnections",
  "metric": "aws:num-disconnects",
  "criteria": {
    "comparisonOperator": "greater-than",
    "statisticalThreshold": {
      "statistic": "p10"
    },
    "durationSeconds": 300,
    "consecutiveDatapointsToAlarm": 1,
    "consecutiveDatapointsToClear": 2
  }
}
```

Scoping metrics in security profiles using dimensions

Dimensions are attributes that you can define to get more precise data about metrics and behaviors in your security profile. You define the scope by providing a value or pattern that is used as a filter. For example, you can define a topic filter dimension that applies a metric only to MQTT topics that match a particular value, such as "data/bulb/+/activity". For information about defining a dimension that you can use in your security profile, see CreateDimension.

Dimension values support MQTT wildcards. MQTT wildcards help you subscribe to multiple topics simultaneously. There are two different kinds of wildcards: single-level (`+`) and multi-level (`#`). For example, the dimension value `Data/bulb/+/activity` creates a subscription that matches all topics that exist on the same level as the `+`. Dimension values also support the MQTT client ID substitution variable ${iot:ClientId}.

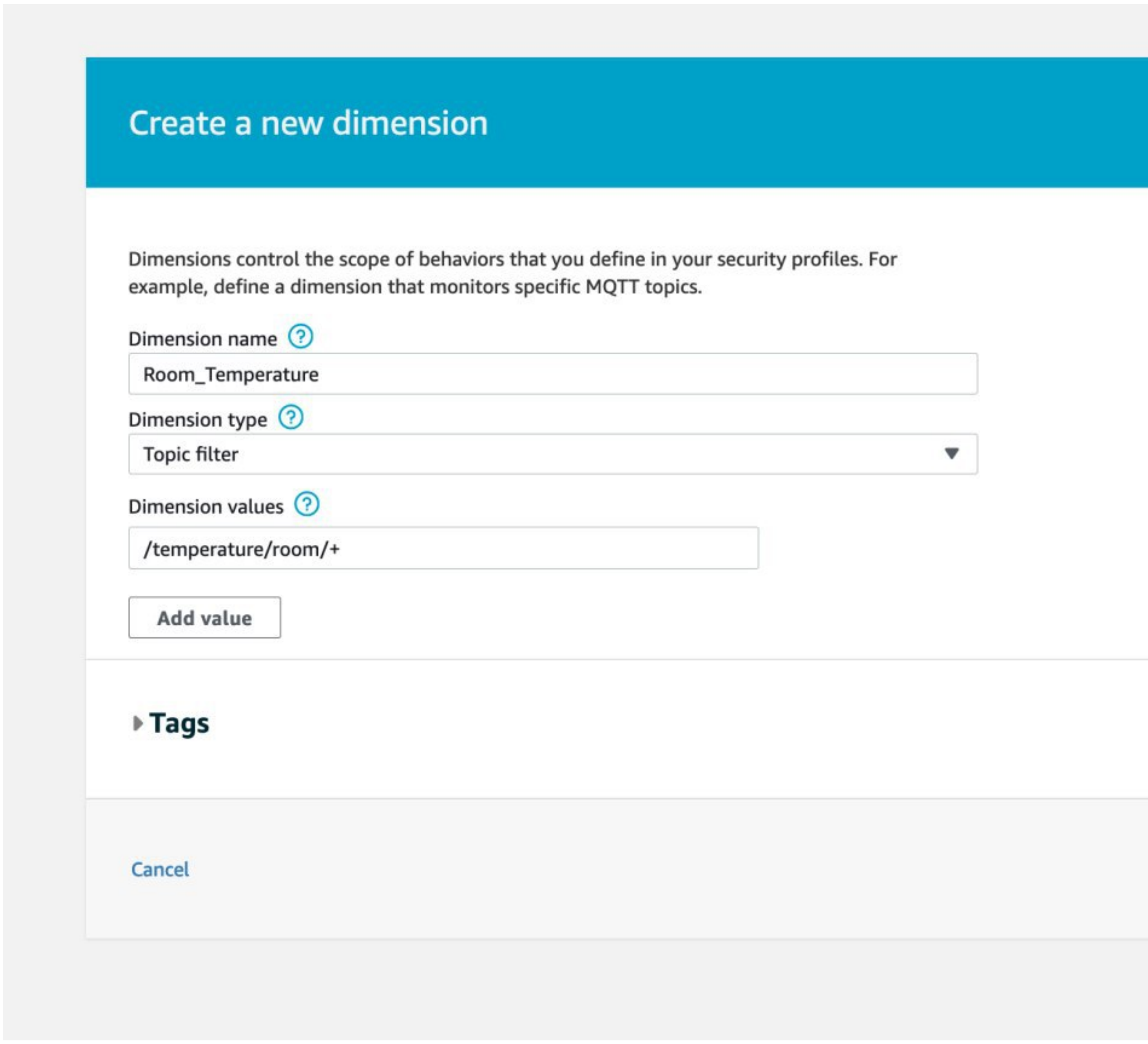
Dimensions of type TOPIC_FILTER are compatible with the following set of cloud-side metrics:

- Number of messages sent
- Number of messages received
- Message byte size
- Source IP address
- Number of authorization failures

How to use dimensions in the console

To create and apply a dimension to a security profile behavior

1. In the AWS IoT console, in the navigation pane, expand **Defend**, expand **Detect**, and then choose **Security profiles**.
2. On the **Security profiles** page, choose **Create** to add a new security profile, or **Edit** to apply a dimension to an existing security profile.
3. On the **Expected behaviors** page, select one of the five cloud-side metrics dimensions supports under **Metric**. The **Dimension** and **Dimension operator** boxes display. For information about supported cloud-side metrics, see Scoping metrics in security profiles using dimensions (p. 867).
4. For **Dimension**, choose **Add dimension**.
5. On the **Create a new dimension** page, enter details for your new dimension. **Dimensions values** supports MQTT wildcards # and + and the MQTT client ID substitution variable ${iot:ClientId}.



Create a new dimension

Dimensions control the scope of behaviors that you define in your security profiles. For example, define a dimension that monitors specific MQTT topics.

Dimension name: Room_Temperature

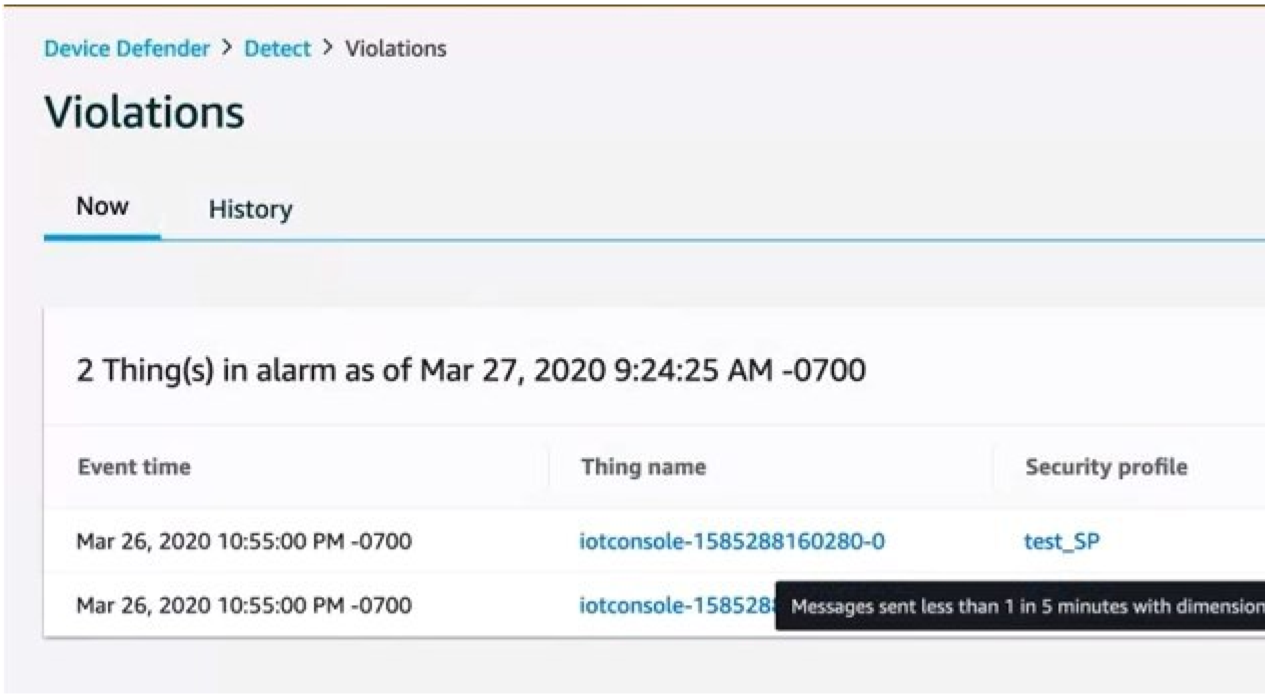
Dimension type: Topic filter

Dimension values: /temperature/room/+

6. Choose **Save**.
7. You can optionally add dimensions to metrics under **Additional Metrics to retain**.
8. To finish creating the behavior, type the information in the other required fields, and then choose **Next**.
9. Complete the remaining steps to finish creating a security profile.

To view your violations

1. In the AWS IoT console, in the navigation pane, expand **Defend**, expand **Detect**, and then choose **Violations**.



Device Defender > Detect > Violations

Violations

Now History

2 Thing(s) in alarm as of Mar 27, 2020 9:24:25 AM -0700

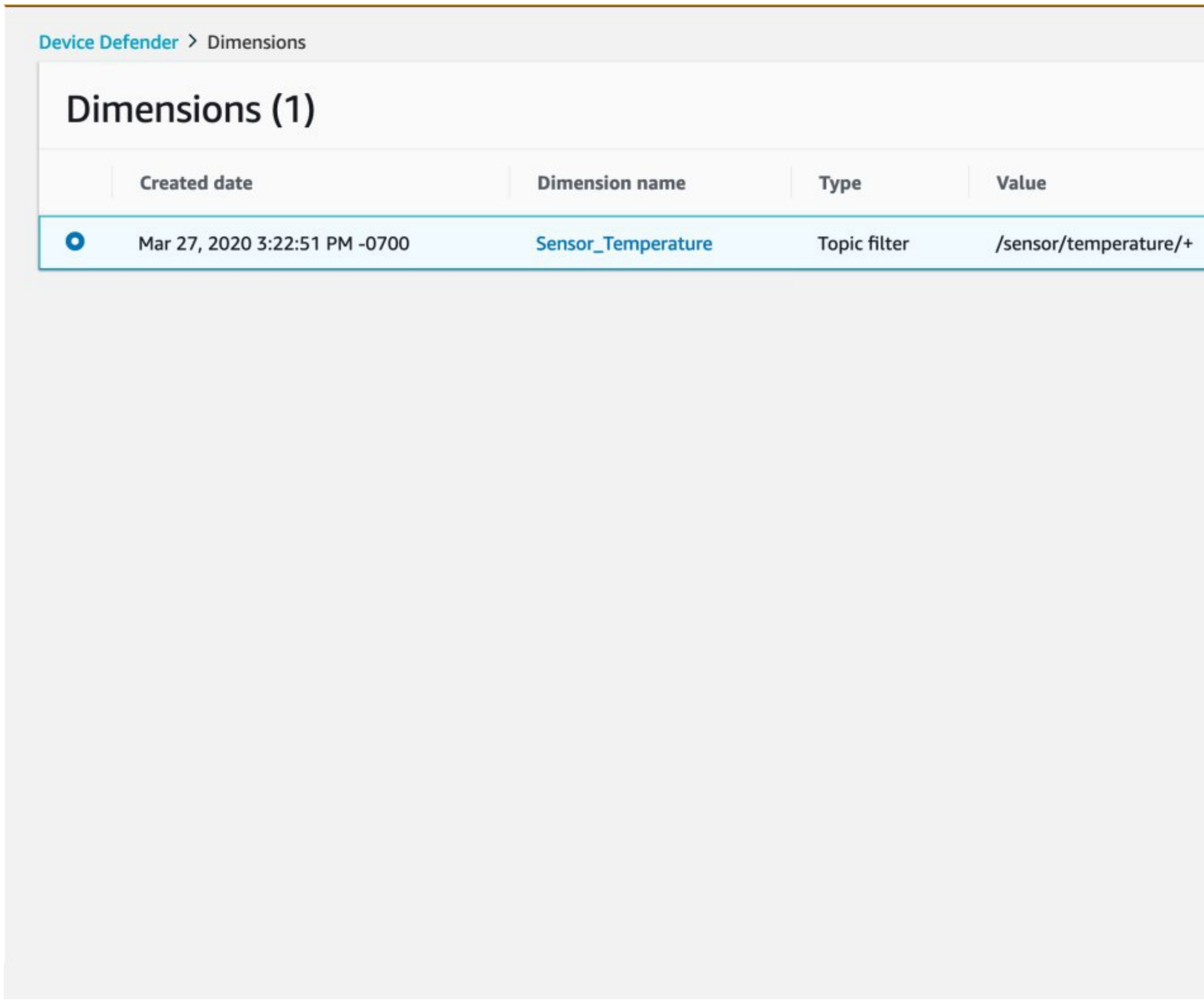
Event time	Thing name	Security profile
Mar 26, 2020 10:55:00 PM -0700	iotconsole-1585288160280-0	test_SP
Mar 26, 2020 10:55:00 PM -0700	iotconsole-158528	

Messages sent less than 1 in 5 minutes with dimension

2. In the **Behavior** column, pause over the behavior you want to see the violation information for.

To view and update your dimensions

1. In the AWS IoT console, in the navigation pane, expand **Defend**, expand **Detect**, and then choose **Dimensions**.
2. Select the dimension that you want to edit.
3. Choose **Actions**, and then choose **Edit**.

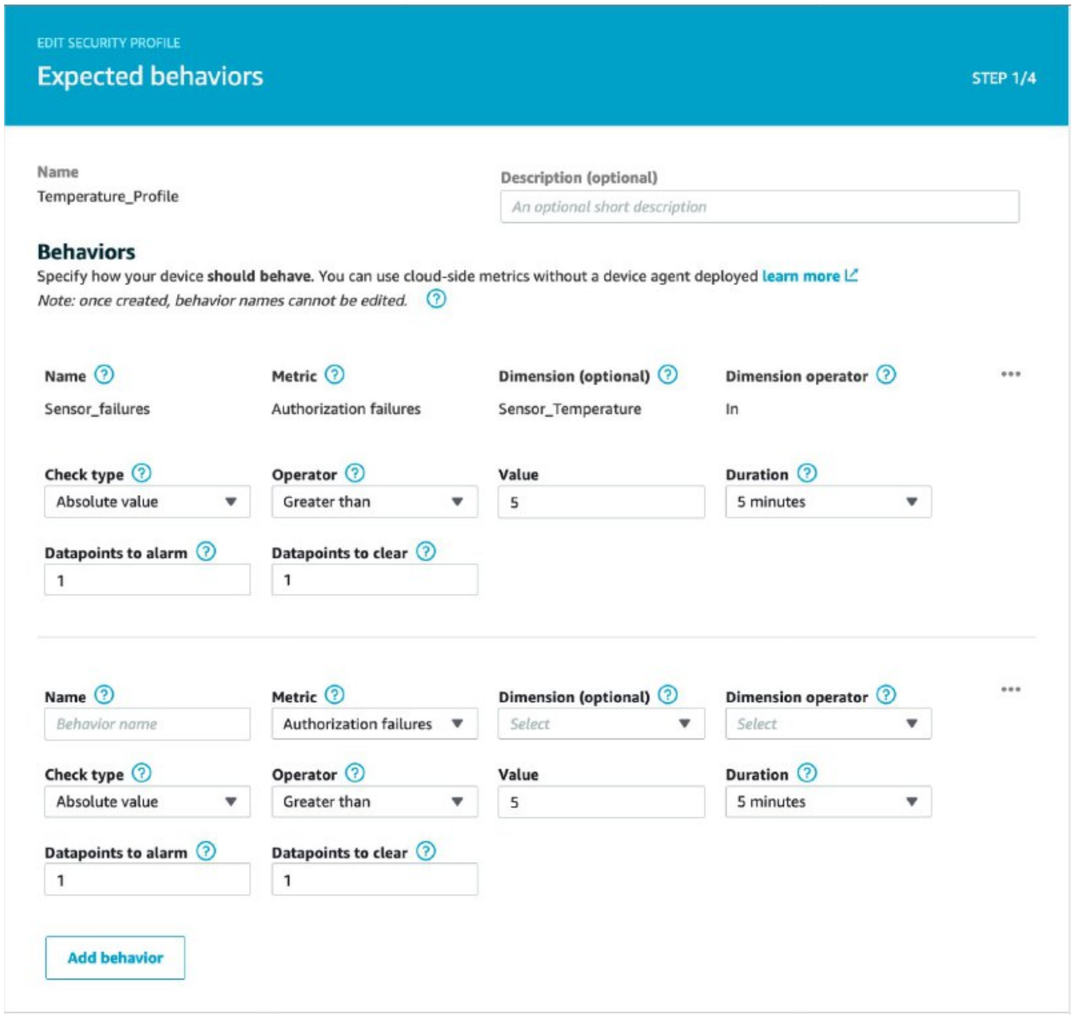


Dimensions (1)

	Created date	Dimension name	Type	Value
●	Mar 27, 2020 3:22:51 PM -0700	Sensor_Temperature	Topic filter	/sensor/temperature/+

To delete a dimension

1. In the AWS IoT console, in the navigation pane, expand **Defend**, expand **Detect**, and then choose **Dimensions**.
2. Select the dimension that you want to delete.
3. Confirm that the dimension isn't attached to a security profile by checking the **Used in** column. If the dimension is attached to a security profile, open the **Security profiles** page on the left, and edit the security profiles that the dimension is attached to. When you delete the dimension, you also delete the behavior. If you want to keep the behavior, choose the ellipsis, then choose **Copy**. Then you can proceed with deleting the behavior. If you want to delete another dimension, follow the steps in this section.


EDIT SECURITY PROFILE
Expected behaviors
STEP 1/4

Name
Temperature_Profile

Description (optional)
An optional short description

Behaviors
Specify how your device **should behave.** You can use cloud-side metrics without a device agent deployed learn more
Note: once created, behavior names cannot be edited.

Name	Metric	Dimension (optional)	Dimension operator
Sensor_failures	Authorization failures	Sensor_Temperature	In

Check type	Operator	Value	Duration
Absolute value	Greater than	5	5 minutes

Datapoints to alarm	Datapoints to clear
1	1

Name	Metric	Dimension (optional)	Dimension operator
Behavior name	Authorization failures	Select	Select

Check type	Operator	Value	Duration
Absolute value	Greater than	5	5 minutes

Datapoints to alarm	Datapoints to clear
1	1

4. Choose **Actions**, and then choose **Delete**.

How to use dimensions on the AWS CLI

To create and apply a dimension to a security profile behavior

1. First create the dimension before attaching it to a security profile. Use the CreateDimension command to create a dimension:

```
aws iot create-dimension \
  --name TopicFilterForAuthMessages \
  --type TOPIC_FILTER \
  --string-values device/+/auth
```

The output of this command looks like the following:

```
{
    "arn": "arn:aws:iot:us-west-2:123456789012:dimension/TopicFilterForAuthMessages",
    "name": "TopicFilterForAuthMessages"
}
```

2. Either add the dimension to an existing security profile by using UpdateSecurityProfile, or add the dimension to a new security profile by using CreateSecurityProfile. In the following example, we create a new security profile that checks if messages to `TopicFilterForAuthMessages` are under 128 bytes, and retains the number of messages sent to non-auth topics.

```
aws iot create-security-profile \
  --security-profile-name ProfileForConnectedDevice \
  --security-profile-description "Check to see if messages to
 TopicFilterForAuthMessages are under 128 bytes and retains the number of messages sent
 to non-auth topics." \
  --behaviors "[{\"name\":\"CellularBandwidth\",\"metric\":\"aws:message-byte-size
\",\"criteria\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}},{\"name
\":\"Authorization\",\"metric\":\"aws:num-authorization-failures\",\"criteria\":
{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":10},\"durationSeconds\":300,
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]" \
  --additional-metrics-to-retain-v2 "[{\"metric\": \"aws:num-authorization-failures\",
\"metricDimension\": {\"dimensionName\": \"TopicFilterForAuthMessages\",\"operator\":
 \"NOT_IN\"}}]"
```

The output of this command looks like the following:

```
{
    "securityProfileArn": "arn:aws:iot:us-west-2:1234564789012:securityprofile/
ProfileForConnectedDevice",
    "securityProfileName": "ProfileForConnectedDevice"
}
```

To save time, you can also load a parameter from a file instead of typing it as a command line parameter value. For more information, see Loading AWS CLI Parameters from a File. The following shows the `behavior` parameter in expanded JSON format:

```
[
  {
    "criteria": {
      "comparisonOperator": "less-than",
      "consecutiveDatapointsToAlarm": 1,
      "consecutiveDatapointsToClear": 1,
      "value": {
        "count": 128
      }
    },
    "metric": "aws:message-byte-size",
    "metricDimension": {
      "dimensionName:": "TopicFilterForAuthMessages"
    },
    "name": "CellularBandwidth"
  }
]
```

To view security profiles with a dimension

- Use the ListSecurityProfiles command to view security profiles with a certain dimension:

```
aws iot list-security-profiles \
  --dimension-name TopicFilterForAuthMessages
```

The output of this command looks like the following:

```
{
    "securityProfileIdentifiers": [
        {
            "name": "ProfileForConnectedDevice",
            "arn": "arn:aws:iot:us-west-2:1234564789012:securityprofile/
ProfileForConnectedDevice"
        }
    ]
}
```

To update your dimension

- Use the UpdateDimension command to update a dimension:

```
aws iot update-dimension \
  --name TopicFilterForAuthMessages \
  --string-values device/${iot:ClientId}/auth
```

The output of this command looks like the following:

```
{
    "name": "TopicFilterForAuthMessages",
    "lastModifiedDate": 1585866222.317,
    "stringValues": [
        "device/${iot:ClientId}/auth"
    ],
    "creationDate": 1585854500.474,
    "type": "TOPIC_FILTER",
    "arn": "arn:aws:iot:us-west-2:1234564789012:dimension/TopicFilterForAuthMessages"
}
```

To delete a dimension

1. To delete a dimension, first detach it from any security profiles that it's attached to. Use the ListSecurityProfiles command to view security profiles with a certain dimension.
2. To remove a dimension from a security profile, use the UpdateSecurityProfile command. Enter all information that you want to keep, but exclude the dimension:

```
aws iot update-security-profile \
  --security-profile-name ProfileForConnectedDevice \
  --security-profile-description "Check to see if authorization fails 10 times in 5
 minutes or if cellular bandwidth exceeds 128" \
  --behaviors "[{\"name\":\"metric\":\"aws:message-byte-size\",\"criteria
\":{\"comparisonOperator\":\"less-than\",\"value\":{\"count\":128},
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}},{\"name
\":\"Authorization\",\"metric\":\"aws:num-authorization-failures\",\"criteria\":
{\comparisonOperator\":\"less-than\",\"value\"{\"count\":10},\"durationSeconds\":300,
\"consecutiveDatapointsToAlarm\":1,\"consecutiveDatapointsToClear\":1}}]"
```

The output of this command looks like the following:

```
{
  "behaviors": [
    {
      "metric": "aws:message-byte-size",
      "name": "CellularBandwidth",
      "criteria": {
        "consecutiveDatapointsToClear": 1,
        "comparisonOperator": "less-than",
        "consecutiveDatapointsToAlarm": 1,
        "value": {
          "count": 128
        }
      }
    },
    {
      "metric": "aws:num-authorization-failures",
      "name": "Authorization",
      "criteria": {
        "durationSeconds": 300,
        "comparisonOperator": "less-than",
        "consecutiveDatapointsToClear": 1,
        "consecutiveDatapointsToAlarm": 1,
        "value": {
          "count": 10
        }
      }
    }
  ],
  "securityProfileName": "ProfileForConnectedDevice",
  "lastModifiedDate": 1585936349.12,
  "securityProfileDescription": "Check to see if authorization fails 10 times in 5
 minutes or if cellular bandwidth exceeds 128",
  "version": 2,
  "securityProfileArn": "arn:aws:iot:us-west-2:123456789012:securityprofile/Preo/
ProfileForConnectedDevice",
  "creationDate": 1585846909.127
}
```

3. After the dimension is detached, use the DeleteDimension command to delete the dimension:

```
aws iot delete-dimension \
  --name TopicFilterForAuthMessages
```

Permissions

This section contains information about how to set up the IAM roles and policies required to manage AWS IoT Device Defender Detect. For more information, see the IAM User Guide.

Give AWS IoT Device Defender detect permission to publish alarms to an SNS topic

If you use the `alertTargets` parameter in CreateSecurityProfile, you must specify an IAM role with two policies: a permissions policy and a trust policy. The permissions policy grants permission to AWS IoT Device Defender to publish notifications to your SNS topic. The trust policy grants AWS IoT Device Defender permission to assume the required role.

Permission policy

```
{
  "Version":"2012-10-17",
  "Statement":[
      {
        "Effect":"Allow",
        "Action":[
          "sns:Publish"
        ],
        "Resource":[
          "arn:aws:sns:region:account-id:your-topic-name"
        ]
    }
  ]
}
```

Trust policy

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

Pass role policy

You also need an IAM permissions policy attached to the IAM user that allows the user to pass roles. See Granting a User Permissions to Pass a Role to an AWS Service.

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

Detect commands

You can use the AWS IoT Device Defender Detect commands in this section to identify unusual behavior that might indicate a compromised device.

- AttachSecurityProfile
- CreateDimension
- CreateSecurityProfile
- DeleteDimension
- DeleteSecurityProfile
- CreateDimension
- DescribeDimension

- DetachSecurityProfile
- GetBehaviorModelTrainingSummaries
- ListActiveViolations
- ListDimensions
- CreateDimension
- ListSecurityProfilesForTarget
- ListTargetsForSecurityProfile
- ListViolationEvents
- UpdateDimension
- UpdateSecurityProfile
- ValidateSecurityProfileBehaviors

How to use AWS IoT Device Defender detect

1. You can use AWS IoT Device Defender Detect with just cloud-side metrics, but if you plan to use device-reported metrics, you must first deploy the AWS IoT SDK on your AWS IoT connected devices or device gateways. For more information, see Sending metrics from devices (p. 859).
2. Consider viewing the metrics that your devices generate before you define behaviors and create alarms. AWS IoT can collect metrics from your devices so you can first identify usual or unusual behavior for a group of devices, or for all devices in your account. Use CreateSecurityProfile, but specify only those `additionalMetricsToRetain` that you're interested in. Don't specify `behaviors` at this point.

   Use the AWS IoT console to look at your device metrics to see what constitutes typical behavior for your devices.
3. Create a set of behaviors for your security profile. Behaviors contain metrics that specify normal behavior for a group of devices or for all devices in your account. For more information and examples, see Cloud-side metrics (p. 861) and Device-side metrics (p. 846). After you create a set of behaviors, you can validate them with ValidateSecurityProfileBehaviors.
4. Use the CreateSecurityProfile action to create a security profile that includes your behaviors. You can use the `alertTargets` parameter to have alarms sent to a target (an SNS topic) when a device violates a behavior. (If you send alarms using SNS, be aware that these count against your AWS account's SNS topic quota. It's possible that a large burst of violations can exceed your SNS topic quota. You can also use CloudWatch metrics to check for violations. For more information, see

   The metrics reported by AWS IoT provide information that you can analyze in different ways. The following use cases are based on a scenario where you have ten things that connect to the internet once a day. Each day:

   - Ten things connect to AWS IoT at roughly the same time.
   - Each thing subscribes to a topic filter, and then waits for an hour before disconnecting. During this period, things communicate with one another and learn more about the state of the world.
   - Each thing publishes some perception it has based on its newly found data using `UpdateThingShadow`.
   - Each thing disconnects from AWS IoT.

   To help you get started, these topics explore some of the questions that you might have.

   - How can I be notified if my things do not connect successfully each day? (p. 319)

- How can I be notified if my things are not publishing data each day? (p. 319)
- How can I be notified if my thing's shadow updates are being rejected each day? (p. 320)
- How can I create a CloudWatch alarm for Jobs? (p. 320)

(p. 318).

5. Use the AttachSecurityProfile action to attach the security profile to a group of devices (a thing group), all registered things in your account, all unregistered things, or all devices. AWS IoT Device Defender Detect starts checking for abnormal behavior and, if any behavior violations are detected, sends alarms. You might want to attach a security profile to all unregistered things if, for example, you expect to interact with mobile devices that are not in your account's thing registry. You can define different sets of behaviors for different groups of devices to meet your needs.

   To attach a security profile to a group of devices, you must specify the ARN of the thing group that contains them. A thing group ARN has the following format.

   ```
   arn:aws:iot:region:account-id:thinggroup/thing-group-name
   ```

   To attach a security profile to all of the registered things in an AWS account (ignoring unregistered things), you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/registered-things
   ```

   To attach a security profile to all unregistered things, you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/unregistered-things
   ```

   To attach a security profile to all devices, you must specify an ARN with the following format.

   ```
   arn:aws:iot:region:account-id:all/things
   ```
6. You can also keep track of violations with the ListActiveViolations action, which lets you to see which violations were detected for a given security profile or target device.

   Use the ListViolationEvents action to see which violations were detected during a specified time period. You can filter these results by security profile or device.
7. If your devices violate the defined behaviors too often, or not often enough, you should fine-tune the behavior definitions.
8. To review the security profiles that you set up and the devices that are being monitored, use the ListSecurityProfiles, ListSecurityProfilesForTarget, and ListTargetsForSecurityProfile actions.

   Use the DescribeSecurityProfile action to get more details about a security profile.
9. To update a security profile, use the UpdateSecurityProfile action. Use the DetachSecurityProfile action to detach a security profile from an account or target thing group. Use the DeleteSecurityProfile action to delete a security profile entirely.

Mitigation actions

You can use AWS IoT Device Defender to take actions to mitigate issues that were found in an Audit finding or Detect alarm.

Note
Mitigation actions won't be performed on suppressed audit findings. For more information about audit finding suppressions, see Audit finding suppressions (p. 823).

Audit mitigation actions

AWS IoT Device Defender provides predefined actions for the different audit checks. You configure those actions for your AWS account and then apply them to a set of findings. Those findings can be:

- All findings from an audit. This option is available in both the AWS IoT console and by using the AWS CLI.
- A list of individual findings. This option is only available by using the AWS CLI.
- A filtered set of findings from an audit.

The following table lists the types of audit checks and the supported mitigation actions for each:

Audit check to mitigation action mapping

Audit check	Supported mitigation actions
REVOKED_CA_CERT_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
DEVICE_CERTIFICATE_SHARED_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS
AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS
IOT_POLICY_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS, REPLACE_DEFAULT_POLICY_VERSION
CA_CERT_APPROACHING_EXPIRATION_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
CONFLICTING_CLIENT_IDS_CHECK	PUBLISH_FINDING_TO_SNS
DEVICE_CERT_APPROACHING_EXPIRATION_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
REVOKED_DEVICE_CERT_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
LOGGING_DISABLED_CHECK	PUBLISH_FINDING_TO_SNS, ENABLE_IOT_LOGGING
DEVICE_CERTIFICATE_KEY_QUALITY_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_DEVICE_CERTIFICATE, ADD_THINGS_TO_THING_GROUP
CA_CERTIFICATE_KEY_QUALITY_CHECK	PUBLISH_FINDING_TO_SNS, UPDATE_CA_CERTIFICATE
IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK	PUBLISH_FINDING_TO_SNS

Audit check	Supported mitigation actions
IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK	PUBLISH_FINDING_TO_SNS

All audit checks support publishing the audit findings to Amazon SNS so you can take custom actions in response to the notification. Each type of audit check can support additional mitigation actions:

REVOKED_CA_CERT_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

DEVICE_CERTIFICATE_SHARED_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK

- No additional supported actions.

AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK

- No additional supported actions.

IOT_POLICY_OVERLY_PERMISSIVE_CHECK

- Add a blank AWS IoT policy version to restrict permissions.

CA_CERT_APPROACHING_EXPIRATION_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

CONFLICTING_CLIENT_IDS_CHECK

- No additional supported actions.

DEVICE_CERT_APPROACHING_EXPIRATION_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

DEVICE_CERTIFICATE_KEY_QUALITY_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

CA_CERTIFICATE_KEY_QUALITY_CHECK

- Change the state of the certificate to mark it as inactive in AWS IoT.

REVOKED_DEVICE_CERT_CHECK

- Change the state of the device certificate to mark it as inactive in AWS IoT.
- Add the devices that use that certificate to a thing group.

LOGGING_DISABLED_CHECK

- Enable logging.

AWS IoT Device Defender supports the following types of mitigation actions on Audit findings:

Action type	Notes
ADD_THINGS_TO_THING_GROUP	You specify the group to which you want to add the devices. You also specify whether membership in one or more dynamic groups should be overridden if that would exceed the maximum number of groups to which the thing can belong.

Action type	Notes
ENABLE_IOT_LOGGING	You specify the logging level and the role with permissions for logging. You cannot specify a logging level of `DISABLED`.
PUBLISH_FINDING_TO_SNS	You specify the topic to which the finding should be published.
REPLACE_DEFAULT_POLICY_VERSION	You specify the template name. Replaces the policy version with a default or blank policy. Only a value of `BLANK_POLICY` is currently supported.
UPDATE_CA_CERTIFICATE	You specify the new state for the CA certificate. Only a value of `DEACTIVATE` is currently supported.
UPDATE_DEVICE_CERTIFICATE	You specify the new state for the device certificate. Only a value of `DEACTIVATE` is currently supported.

By configuring standard actions when issues are found during an audit, you can respond to those issues consistently. Using these defined mitigation actions also helps you resolve the issues more quickly and with less chance of human error.

Important
Applying mitigation actions that change certificates, add things to a new thing group, or replace the policy can have an impact on your devices and applications. For example, devices might be unable to connect. Consider the implications of the mitigation actions before you apply them. You might need to take other actions to correct the problems before your devices and applications can function normally. For example, you might need to provide updated device certificates. Mitigation actions can help you quickly limit your risk, but you must still take corrective actions to address the underlying issues.

Some actions, such as reactivating a device certificate, can only be performed manually. AWS IoT Device Defender does not provide a mechanism to automatically roll back mitigation actions that have been applied.

Detect mitigation actions

AWS IoT Device Defender supports the following types of mitigation actions on Detect alarms:

Action type	Notes
ADD_THINGS_TO_THING_GROUP	You specify the group to which you want to add the devices. You also specify whether membership in one or more dynamic groups should be overridden if that would exceed the maximum number of groups to which the thing can belong.

How to define and manage mitigation actions

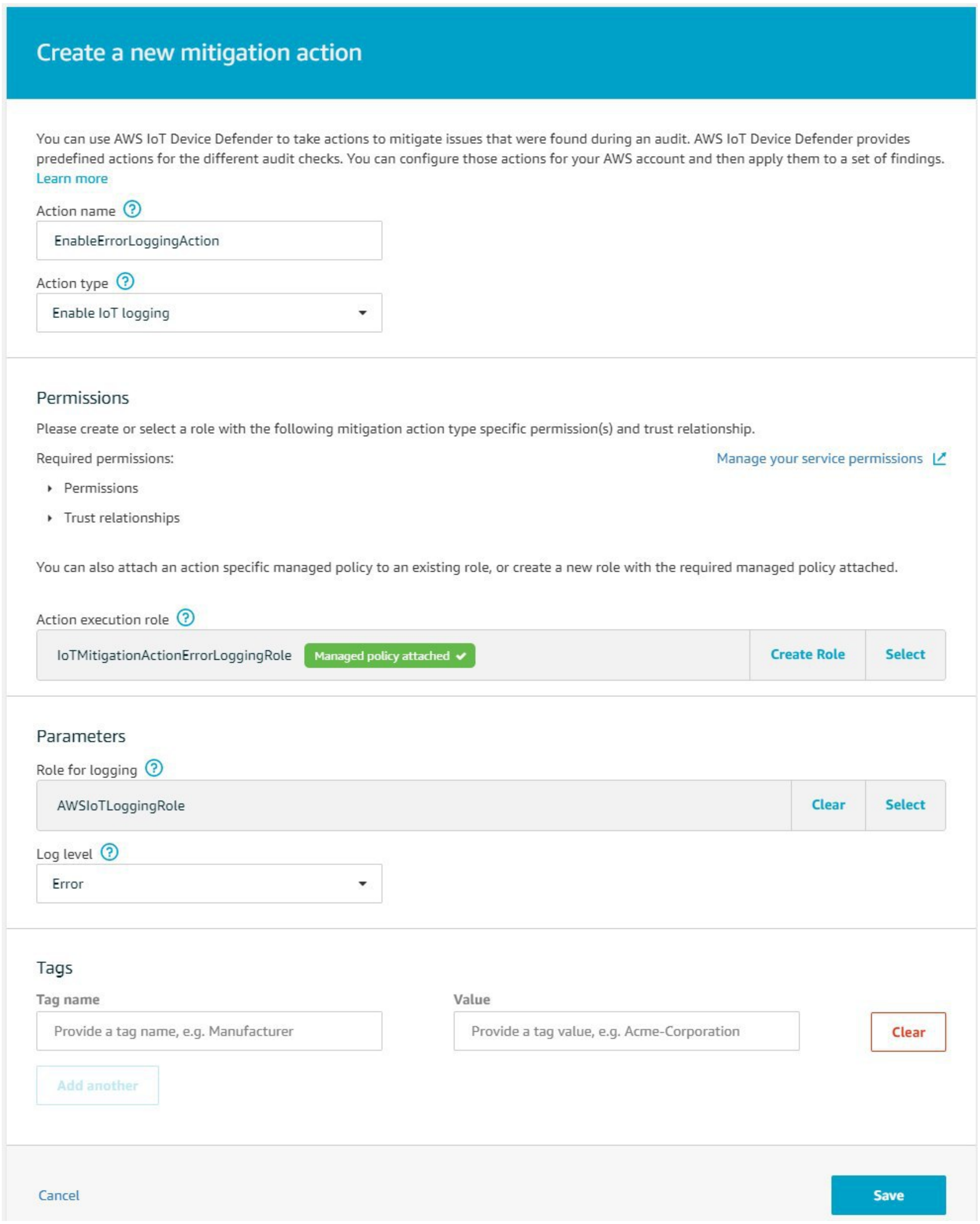
You can use the AWS IoT console or the AWS CLI to define and manage mitigation actions for your AWS account.

Create mitigation actions

Each mitigation action that you define is a combination of a predefined action type and parameters specific to your account.

To use the AWS IoT console to create mitigation actions

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Mitigation Actions**.
3. On the **Mitigation Actions** page, choose **Create**.



Create a new mitigation action

You can use AWS IoT Device Defender to take actions to mitigate issues that were found during an audit. AWS IoT Device Defender provides predefined actions for the different audit checks. You can configure those actions for your AWS account and then apply them to a set of findings. Learn more

Action name: EnableErrorLoggingAction

Action type: Enable IoT logging

Permissions

Please create or select a role with the following mitigation action type specific permission(s) and trust relationship.

Required permissions:

- Permissions
- Trust relationships

Manage your service permissions

You can also attach an action specific managed policy to an existing role, or create a new role with the required managed policy attached.

Action execution role: IoTMitigationActionErrorLoggingRole (Managed policy attached ✓)

Parameters

Role for logging: AWSIoTLoggingRole

Log level: Error

Tags

Tag name: Provide a tag name, e.g. Manufacturer

Value: Provide a tag value, e.g. Acme-Corporation

4. On the **Create a Mitigation Action** page, in **Action name**, enter a unique name for your mitigation action.
5. In **Action type**, specify the type of action that you want to define.
6. Each action type requests a different set of parameters. Enter the parameters for the action. For example, if you choose the **Add things to thing group** action type, choose the destination group and select or clear **Override dynamic groups**.
7. In **Action execution role**, choose the role under whose permissions the action is applied.
8. Choose **Save** to save your mitigation action to your AWS account.

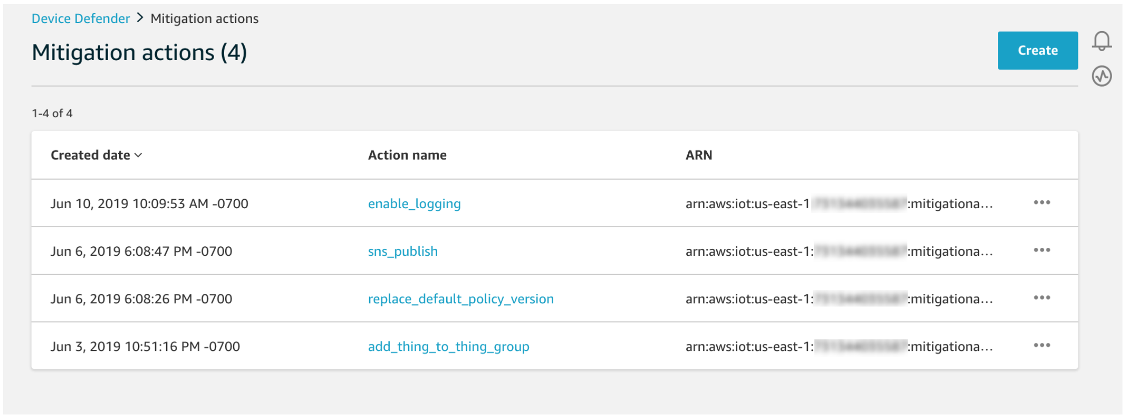
To use the AWS CLI to create mitigation actions

- Use the CreateMitigationAction command to create your mitigation action. The unique name that you give the action is used when you apply that action to audit findings. Choose a meaningful name.

To use the AWS IoT console to view and modify mitigation actions

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Mitigation Actions**.

   The **Mitigation Actions** page displays a list of all of the mitigation actions that are defined for your AWS account.

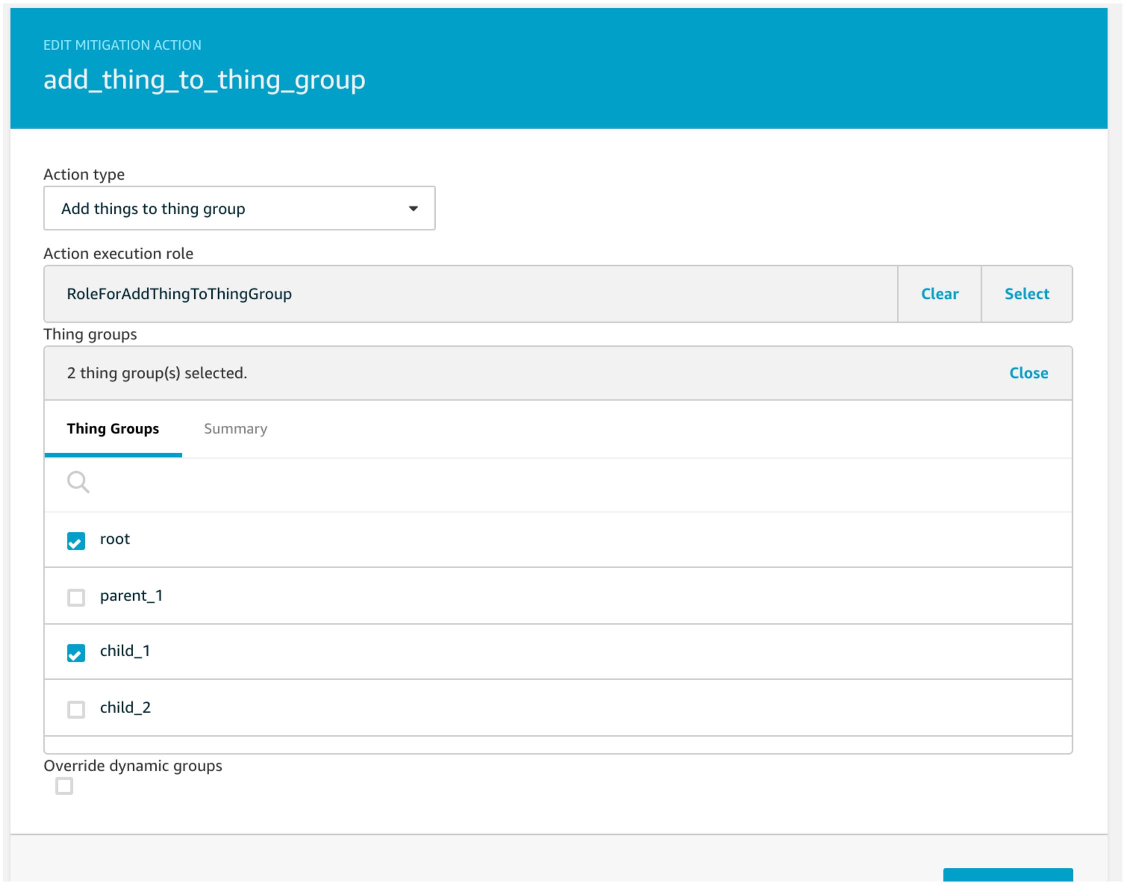


Mitigation actions (4)

1-4 of 4

Created date	Action name	ARN
Jun 10, 2019 10:09:53 AM -0700	enable_logging	arn:aws:iot:us-east-1[redacted]:mitigationa...
Jun 6, 2019 6:08:47 PM -0700	sns_publish	arn:aws:iot:us-east-1:[redacted]:mitigationa...
Jun 6, 2019 6:08:26 PM -0700	replace_default_policy_version	arn:aws:iot:us-east-1:[redacted]:mitigationa...
Jun 3, 2019 10:51:16 PM -0700	add_thing_to_thing_group	arn:aws:iot:us-east-1:[redacted]:mitigationa...

3. Choose the action name link for the mitigation action that you want to change.
4. Make your changes to the mitigation action. Because the name of the mitigation action is used to identify it, you cannot change the name.


EDIT MITIGATION ACTION

add_thing_to_thing_group

Action type: Add things to thing group

Action execution role: RoleForAddThingToThingGroup

Thing groups: 2 thing group(s) selected.

5. Choose **Save** to save the changes to the mitigation action to your AWS account.

To use the AWS CLI to list a mitigation action

- Use the ListMitigationAction command to list your mitigation actions. If you want to change or delete a mitigation action, make a note of the name.

To use the AWS CLI to update a mitigation action

- Use the UpdateMitigationAction command to change your mitigation action.

To use the AWS IoT console to delete a mitigation action

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Mitigation Actions**.

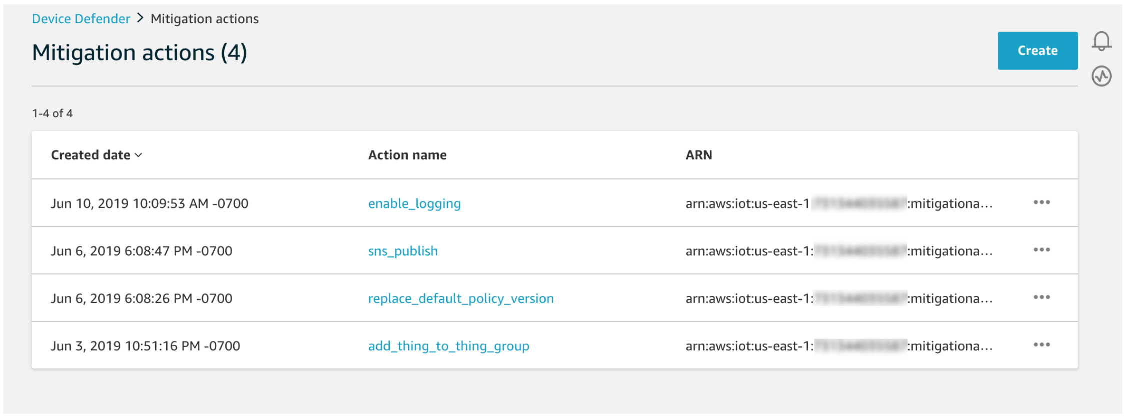
   The **Mitigation Actions** page displays all of the mitigation actions that are defined for your AWS account.
3. Choose the ellipsis (**...**) for the mitigation action that you want to delete, and then choose **Delete**.

To use the AWS CLI to delete mitigation actions

- Use the UpdateMitigationAction command to change your mitigation action.

To use the AWS IoT console to view mitigation action details

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Mitigation Actions**.



Device Defender > Mitigation actions

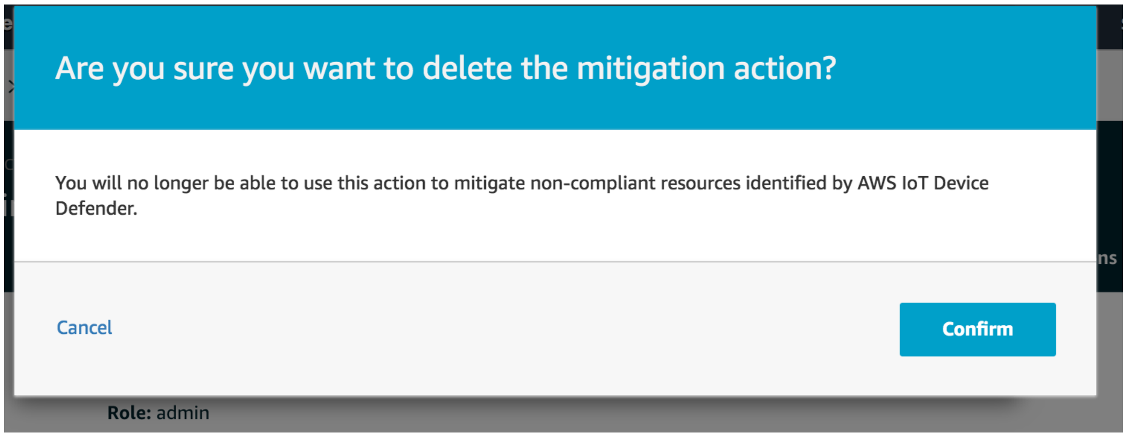
Mitigation actions (4)

1-4 of 4

Created date	Action name	ARN
Jun 10, 2019 10:09:53 AM -0700	enable_logging	arn:aws:iot:us-east-1[redacted]:mitigationa...
Jun 6, 2019 6:08:47 PM -0700	sns_publish	arn:aws:iot:us-east-1:[redacted]:mitigationa...
Jun 6, 2019 6:08:26 PM -0700	replace_default_policy_version	arn:aws:iot:us-east-1:[redacted]:mitigationa...
Jun 3, 2019 10:51:16 PM -0700	add_thing_to_thing_group	arn:aws:iot:us-east-1:[redacted]:mitigationa...

The **Mitigation Actions** page displays all of the mitigation actions that are defined for your AWS account.

3. Choose the action name link for the mitigation action that you want to change.
4. In the **Are you sure you want to delete the mitigation action** window, choose **Confirm**.



Are you sure you want to delete the mitigation action?

You will no longer be able to use this action to mitigate non-compliant resources identified by AWS IoT Device Defender.

Role: admin

To use the AWS CLI to view mitigation action details

- Use the DescribeMitigationAction command to view details for your mitigation action.

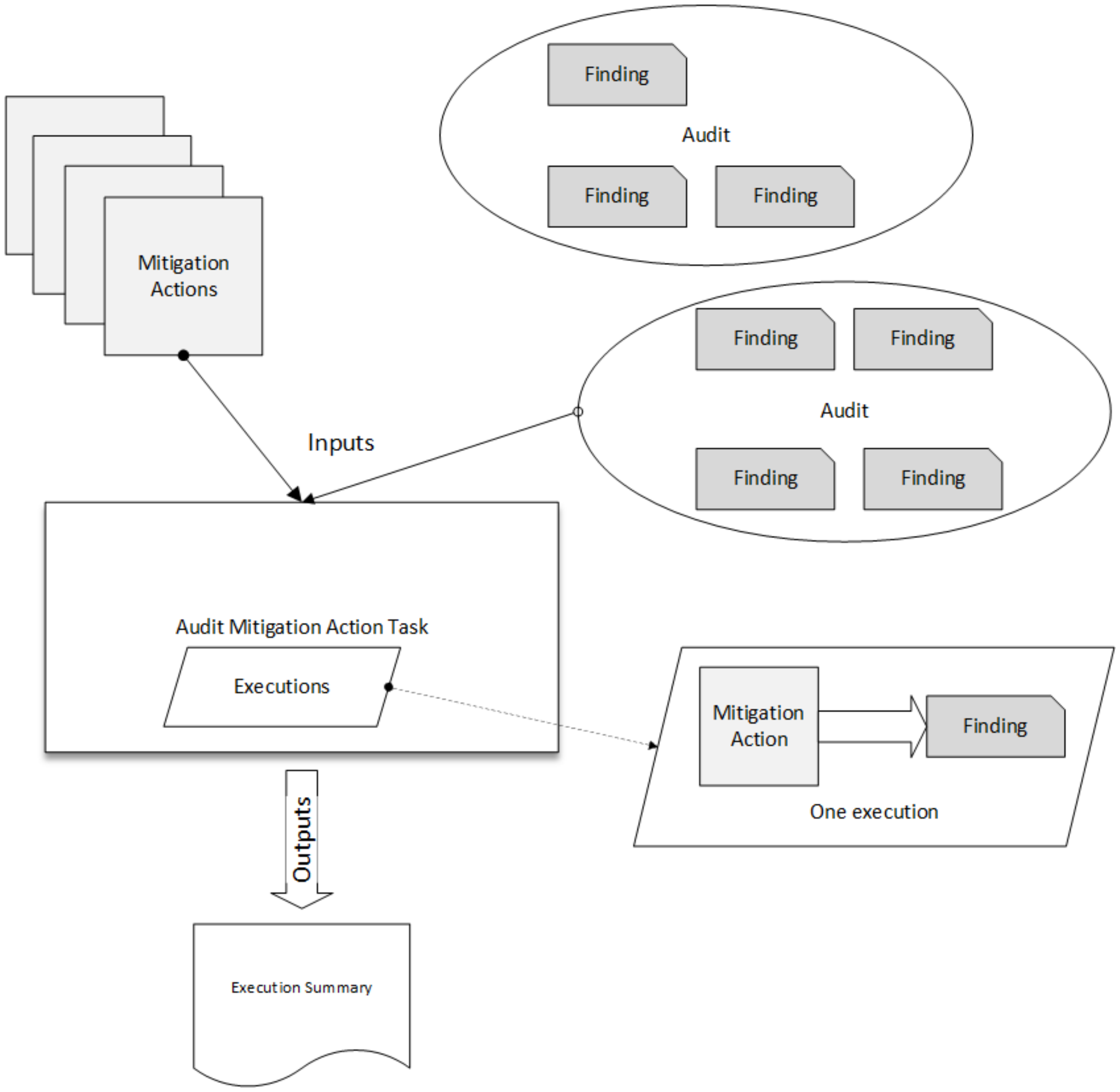
Apply mitigation actions

After you have defined a set of mitigation actions, you can apply those actions to the findings from an audit. When you apply actions, you start an audit mitigation actions task. This task might take some time to complete, depending on the set of findings and the actions that you apply to them. For example, if you have a large pool of devices whose certificates have expired, it might take some time to deactivate all of those certificates or to move those devices to a quarantine group. Other actions, such as enabling logging, can be completed quickly.

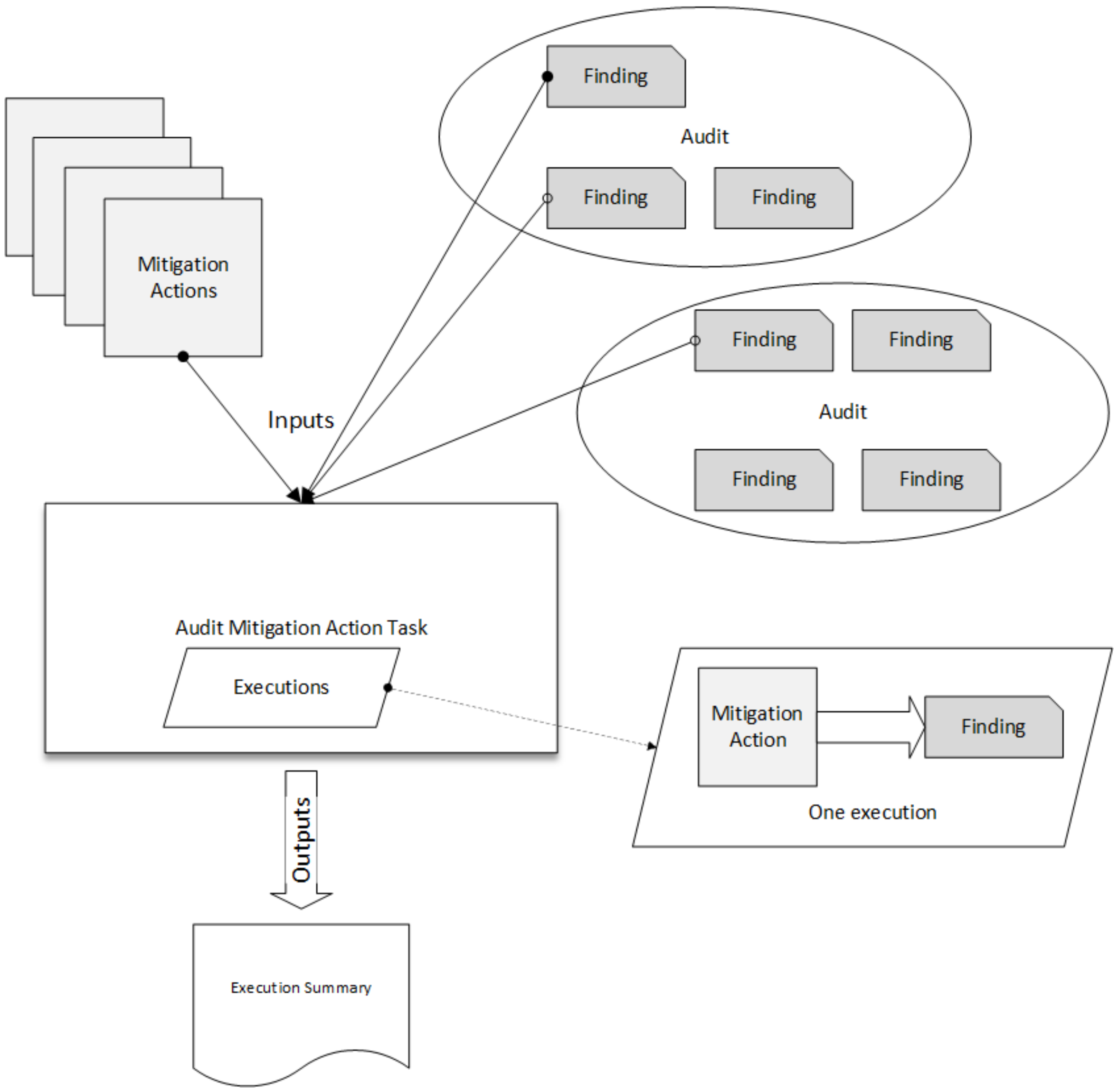
You can view the list of action executions and cancel an execution that has not yet been completed. Actions already performed as part of the canceled action execution are not rolled back. If you are applying multiple actions to a set of findings and one of those actions failed, the subsequent actions are skipped for that finding (but are still applied to other findings). The task status for the finding is FAILED. The `taskStatus` is set to failed if one or more of the actions failed when applied to the findings. Actions are applied in the order in which they are specified.

Each action execution applies a set of actions to a target. That target can be a list of findings or it can be all findings from an audit.

The following diagram shows how you can define an audit mitigation task that takes all findings from one audit and applies a set of actions to those findings. A single execution applies one action to one finding. The audit mitigation actions task outputs an execution summary.



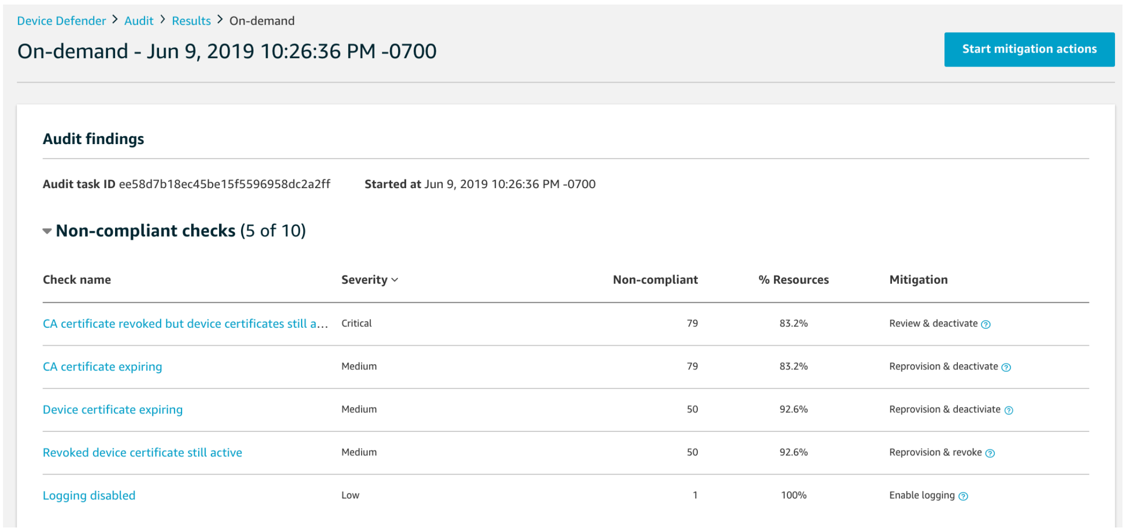
The following diagram shows how you can define an audit mitigation task that takes a list of individual findings from one or more audits and applies a set of actions to those findings. A single execution applies one action to one finding. The audit mitigation actions task outputs an execution summary.



You can use the AWS IoT console or the AWS CLI to apply mitigation actions.

To use the AWS IoT console to apply mitigation actions by starting an action execution

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, choose **Audit**, and then choose **Results**.



Device Defender > Audit > Results > On-demand

On-demand - Jun 9, 2019 10:26:36 PM -0700

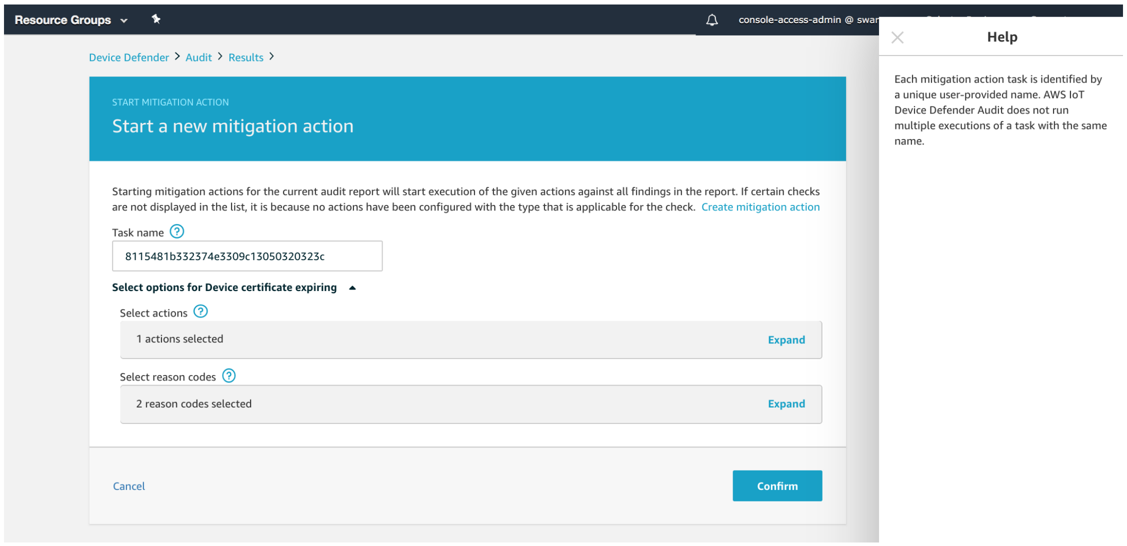
Audit findings

Audit task ID ee58d7b18ec45be15f5596958dc2a2ff **Started at** Jun 9, 2019 10:26:36 PM -0700

Non-compliant checks (5 of 10)

Check name	Severity	Non-compliant	% Resources	Mitigation
CA certificate revoked but device certificates still a...	Critical	79	83.2%	Review & deactivate
CA certificate expiring	Medium	79	83.2%	Reprovision & deactivate
Device certificate expiring	Medium	50	92.6%	Reprovision & deactivate
Revoked device certificate still active	Medium	50	92.6%	Reprovision & revoke
Logging disabled	Low	1	100%	Enable logging

3. Choose the name for the audit to which you want to apply actions.



Device Defender > Audit > Results >

START MITIGATION ACTION

Start a new mitigation action

Starting mitigation actions for the current audit report will start execution of the given actions against all findings in the report. If certain checks are not displayed in the list, it is because no actions have been configured with the type that is applicable for the check. Create mitigation action

Task name: 8115481b332374e3309c13050320323c

Select options for Device certificate expiring

Select actions: 1 actions selected

Select reason codes: 2 reason codes selected

Help

Each mitigation action task is identified by a unique user-provided name. AWS IoT Device Defender Audit does not run multiple executions of a task with the same name.

4. Choose **Start Mitigation Actions**. This button is not available if all of your checks are compliant.

5. In **Are you sure that you want to start mitigation action task**, the task name defaults to the audit ID, but you can change it to something more meaningful.

6. For each type of check that had one or more noncompliant findings in the audit, you can choose one or more actions to apply. Only actions that are valid for the check type are displayed.

   Note

   If you have not configured actions for your AWS account, the list of actions is empty. You can choose the **click here** link to create one or more mitigation actions.

7. When you have specified all of the actions that you want to apply, choose **Confirm**.

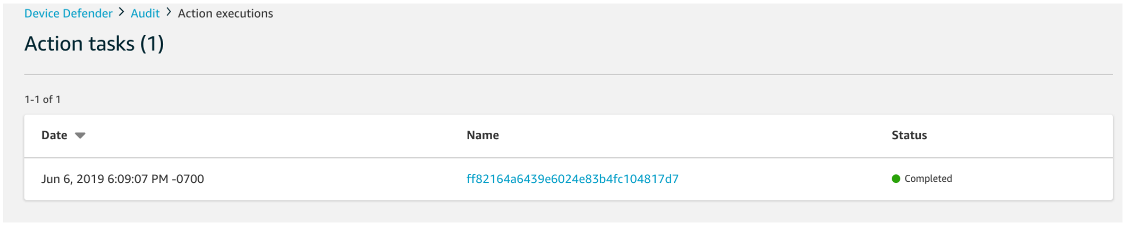
To use the AWS CLI to apply mitigation actions by starting an audit mitigation actions execution

1. If you want to apply actions to all findings for the audit, use the ListAuditTasks command to find the task ID.

2. If you want to apply actions to selected findings only, use the ListAuditFindings command to get the finding IDs.
3. Use the ListMitigationActions command and make note of the names of the mitigation actions that you want to apply.
4. Use the StartAuditMitigationActionsTask command to apply actions to the target. Make note of the task ID. You can use the ID to check the state of the action execution, review the details, or cancel it.

To use the AWS IoT console to view your action executions

1. Open the AWS IoT console.
2. In the left navigation pane, choose **Defend**, and then choose **Action Executions**.



Device Defender > Audit > Action executions

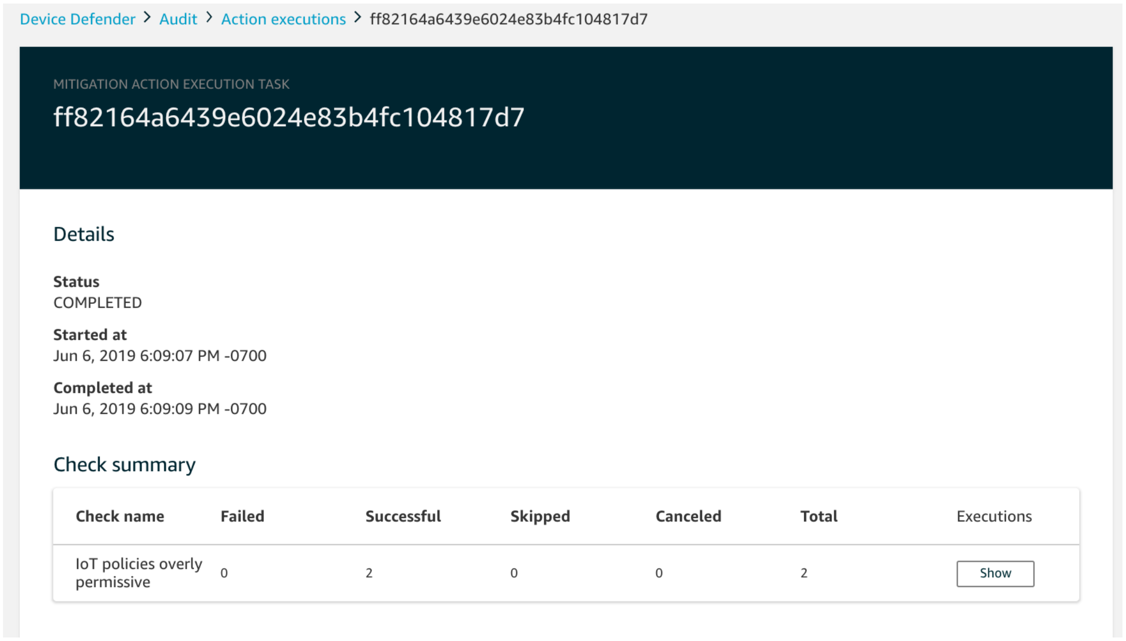
Action tasks (1)

1-1 of 1

Date	Name	Status
Jun 6, 2019 6:09:07 PM -0700	ff82164a6439e6024e83b4fc104817d7	● Completed

A list of action tasks shows when each was started and the current status.

3. Choose the **Name** link to see details for the task. The details include all of the actions that are applied by the task, their target, and their status.



Device Defender > Audit > Action executions > ff82164a6439e6024e83b4fc104817d7

MITIGATION ACTION EXECUTION TASK
ff82164a6439e6024e83b4fc104817d7

Details

Status
COMPLETED

Started at
Jun 6, 2019 6:09:07 PM -0700

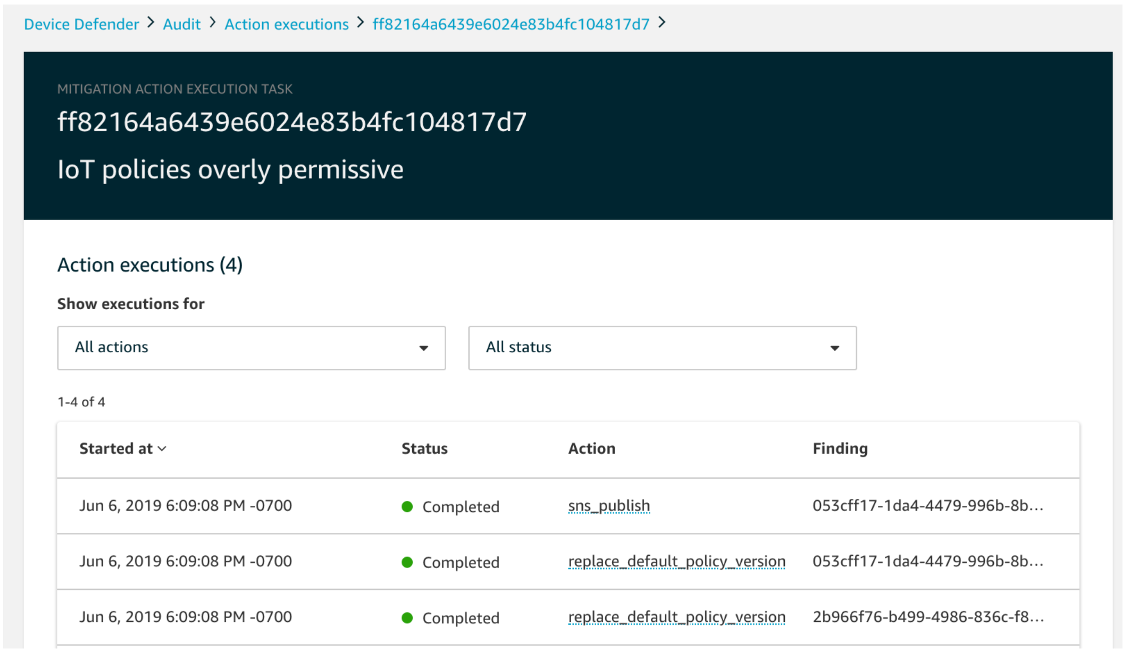
Completed at
Jun 6, 2019 6:09:09 PM -0700

Check summary

Check name	Failed	Successful	Skipped	Canceled	Total	Executions
IoT policies overly permissive	0	2	0	0	2	Show

You can use the **Show executions for** filters to focus on types of actions or action states.

4. To see details for the task, in **Executions**, choose **Show**.



Device Defender > Audit > Action executions > ff82164a6439e6024e83b4fc104817d7 >

MITIGATION ACTION EXECUTION TASK

ff82164a6439e6024e83b4fc104817d7

IoT policies overly permissive

Action executions (4)

Show executions for

All actions | All status

1-4 of 4

Started at	Status	Action	Finding
Jun 6, 2019 6:09:08 PM -0700	● Completed	sns_publish	053cff17-1da4-4479-996b-8b...
Jun 6, 2019 6:09:08 PM -0700	● Completed	replace_default_policy_version	053cff17-1da4-4479-996b-8b...
Jun 6, 2019 6:09:08 PM -0700	● Completed	replace_default_policy_version	2b966f76-b499-4986-836c-f8...

To use the AWS CLI to list your started tasks

1. Use ListAuditMitigationActionsTasks to view your audit mitigation actions tasks. You can provide filters to narrow the results. If you want to view details of the task, make note of the task ID.
2. Use ListAuditMitigationActionsExecutions to view execution details for a particular audit mitigation actions task.
3. Use DescribeAuditMitigationActionsTask to view details about the task, such as the parameters specified when it was started.

To use the AWS CLI to cancel a running audit mitigation actions task

1. Use the ListAuditMitigationActionsTasks command to find the task ID for the task whose execution you want to cancel. You can provide filters to narrow the results.
2. Use the ListDetectMitigationActionsExecutions command, using the task ID, to cancel your audit mitigation actions task. You cannot cancel tasks that have been completed. When you cancel a task, remaining actions are not applied, but mitigation actions that were already applied are not rolled back.

Permissions

For each mitigation action that you define, you must provide the role used to apply that action.

Permissions for mitigation actions

Action type	Permissions policy template	

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

Action type	Permissions policy template	

	`{`
	`"Effect":"Allow",`
	`"Action":[`
	`"iot:UpdateCertificate"`
	`],`
	`"Resource":[`
	`"*"`
	`]`
	`}`
	`]`
	`}`	

UPDATE_CA_CERTIFICATE	`{`
	`"Version":"2012-10-17",`
	`"Statement":[`
	`{`
	`"Effect":"Allow",`
	`"Action":[`
	`"iot:UpdateCACertificate"`
	`],`
	`"Resource":[`
	`"*"`
	`]`
	`}`
	`]`
	`}`	

ADD_THINGS_TO_THING_GROUP	`{`
	`"Version":"2012-10-17",`
	`"Statement":[`
	`{`
	`"Effect":"Allow",`
	`"Action":[`
	`"iot:ListPrincipalThings",`
	`"iot:AddThingToThingGroup"`
	`],`
	`"Resource":[`
	`"*"`
	`]`
	`}`
	`]`
	`}`	

Action type	Permissions policy template	
REPLACE_DEFAULT_POLICY_VERSION	`{ "Version":"2012-10-17", "Statement":[ { "Effect":"Allow", "Action":[ "iot:CreatePolicyVersion" ], "Resource":[ "*" ] } ] }`	
ENABLE_IOT_LOGGING	`{ "Version":"2012-10-17", "Statement":[ { "Effect":"Allow", "Action":[ "iot:SetV2LoggingOptions" ], "Resource":[ "*" ] }, { "Effect":"Allow", "Action":[ "iam:PassRole" ], "Resource":[ "<IAM role ARN used for setting up logging>" ] } ] }`	

Action type	Permissions policy template	
PUBLISH_FINDING_TO_SNS	`{ "Version":"2012-10-17", "Statement":[ { "Effect":"Allow", "Action":[ "sns:Publish" ], "Resource":[ "<The SNS topic to which the finding is published>" ] } ] }`	

For all mitigation action types, use the following trust policy template:

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

Mitigation action commands

You can use these mitigation action commands to define a set of actions for your AWS account that you can later apply to one or more sets of audit findings. There are three command categories:

- Those used to define and manage actions.
- Those used to start and manage the application of those actions to Audit findings.
- Those used to start and manage the application of those actions to Detect alarms.

Mitigation action commands

Define and manage actions	Start and manage Audit execution	Start and manage Detect execution
CreateMitigationAction	CancelAuditMitigationActionsTask	CancelDetectMitigationActionsTask
DeleteMitigationAction	DescribeAuditMitigationActionsTask	DescribeDetectMitigationActionsTask

Define and manage actions	Start and manage Audit execution	Start and manage Detect execution
DescribeMitigationAction	ListAuditMitigationActionsTasks	ListDetectMitigationActionsTasks
ListMitigationActions	StartAuditMitigationActionsTask	StartDetectMitigationActionsTask
UpdateMitigationAction	ListAuditMitigationActionsExecutions	ListDetectMitigationActionsExecutions

Device agent integration with AWS IoT Greengrass

AWS IoT Device Defender can be used with AWS IoT Greengrass. Device agent integration follows the standard AWS IoT Greengrass Lambda function deployment model, allowing you to add AWS IoT Device Defender security to your AWS IoT Greengrass core devices. Follow these steps to integrate a device agent.

Prerequisites:

- Set up your AWS IoT Greengrass environment.
- Configure and run your AWS IoT Greengrass core.
- Ensure you can successfully deploy and run a Lambda function on your AWS IoT Greengrass core.

In general, the process described here follows the Create and Package a Lambda Function section in the *AWS IoT Greengrass Developer Guide*.

To create a Lambda package

1. Clone the AWS IoT Device Defender Python samples repository.

```
git clone https://github.com/aws-samples/aws-iot-device-defender-agent-sdk-python.git
```

2. Create and activate a virtual environment (optional, but recommended).

```
pip install virtualenv
virtualenv metrics_lambda_environment
source metrics_lambda_environment/bin/activate
```

3. Install the AWS IoT Device Defender sample agent in the virtual environment. Install from PyPi.

```
pip install AWSIoTDeviceDefenderAgentSDK
```

4. Install the downloaded source.

```
cd aws-iot-device-defender-agent-sdk-python
#This must be run from the same directory as setup.py
pip install .
```

5. Create an empty directory to assemble your Lambda function. This is your Lambda directory.

```
mkdir metrics_lambda
cd metrics_lambda
```

6. Complete steps 1-4 in Create and Package a Lambda Function.
7. Unzip the AWS IoT Greengrass Python SDK into your Lambda directory.

```
unzip ../aws_greengrass_core_sdk/sdk/python_sdk_1_1_0.zip
cp -R ../aws_greengrass_core_sdk/examples/HelloWorld/greengrass_common .
cp -R ../aws_greengrass_core_sdk/examples/HelloWorld/greengrasssdk .
cp -R ../aws_greengrass_core_sdk/examples/HelloWorld/greengrass_ipc_python_sdk .
```

8. Copy the `AWSIoTDeviceDefenderAgentSDK` module to the root level of your Lambda directory.

```
cp -R ../aws-iot-device-defender-agent-sdk-python/AWSIoTDeviceDefenderAgentSDK .
```

9. Copy the AWS IoT Greengrass agent to the root level of your Lambda directory.

```
cp ../aws-iot-device-defender-agent-sdk-python/samples/greengrass/
greengrass_core_metrics_agent/greengrass_defender_agent.py .
```

10. Customize the AWS IoT Greengrass agent to include the name of your AWS IoT Greengrass core device and the desired metrics sample rate:

    - Replace `GREENGRASS_CORENAME` with the name of your AWS IoT Greengrass core.
    - Set the `SAMPLE_RATE_SECONDS` to your desired metrics reporting interval. The shortest reporting interval supported by AWS IoT Device Defender is 5 minutes (300 seconds).

11. Copy the dependencies from your virtual environment (or your system) into the root level of your Lambda directory.

```
cp -R ../metrics_lambda_environment/lib/python2.7/site-packages/psutil .
cp -R ../metrics_lambda_environment/lib/python2.7/site-packages/cbor .
```

12. Create your Lambda function zip file. Perform this command in the root level of your Lambda directory.

```
rm *.zip
zip -r greengrass_defender_metrics_lambda.zip *
```

To configure and deploy your AWS IoT Greengrass Lambda function

1. Upload your lambda zip file.
2. Select the Python 2.7 runtime, and in the Handler field, enter `greengrass_defender_agent.function_handler`.
3. Configure your Lambda function as a long-lived Lambda function.
4. Configure a subscription from your Lambda function to the AWS IoT cloud. For AWS IoT Device Defender, a subscription from the AWS Cloud to your Lambda function is not required.
5. Create a local resource to allow your Lambda function to collect metrics from your AWS IoT Greengrass core host:

    - Follow the instructions in Access Local Resources with Lambda Functions. Use the following parameters:
        - Resource Name: `Core Proc`
        - Type: `Volume`
        - Source Path: `/proc`
        - Destination Path: `/host_proc`
        - Group owner file access permission: Automatically add OS group permissions of the Linux group that owns the resource
        - Associate the resource with your metrics Lambda function.

6. Deploy your Lambda function to your AWS IoT Greengrass group.

To review your AWS IoT Device Defender device metrics using the AWS IoT console

1. Temporarily modify the publish topic in your AWS IoT Greengrass Lambda function to "metrics/test".
2. Deploy the Lambda function.
3. To see the metrics your AWS IoT Greengrass core is emitting, on the **Test** page of the AWS IoT console, add a subscription to the temporary topic ("metrics/test") .

Security best practices for device agents

Least Privilege

The agent process should be granted only the minimum permissions required to perform its duties.

Basic mechanisms

- Agent should be run as non-root user.
- Agent should run as a dedicated user, in its own group.
- User/groups should be granted read-only permissions on the resources required to gather and transmit metrics.
- Example: read-only on /proc /sys for the sample agent.
- For an example of how to set up a process to run with reduced permissions, see the setup instructions that are included with the Python sample agent.

There are a number of well-known Linux mechanisms that can help you further restrict or isolate your agent process:

Advanced mechanisms

- CGroups
- SELinux
- Chroot
- Linux Namespaces

Operational Resiliency

An agent process must be resilient to unexpected operational errors and exceptions and must not crash or exit permanently. The code needs to gracefully handle exceptions and, as a precaution, it must be configured to automatically restart in case of unexpected termination (for example, due to system restarts or uncaught exceptions).

Least Dependencies

An agent must use the least possible number of dependencies (that is, third-party libraries) in its implementation. If use of a library is justified due to the complexity of a task (for example, transport layer security), use only well-maintained dependencies and establish a mechanism to keep them up to date. If the added dependencies contain functionality not used by the agent and active by default (for example, opening ports, domain sockets), disable them in your code or by means of the library's configuration files.

Process Isolation

An agent process must only contain functionality required for performing device metric gathering and transmission. It must not piggyback on other system processes as a container or implement functionality for other out of scope use cases. In addition, the agent process must refrain from creating inbound communication channels such as domain socket and network service ports that would allow local or remote processes to interfere with its operation and impact its integrity and isolation.

Stealthiness

An agent process must not be named with keywords such as security, monitoring, or audit indicating its purpose and security value. Generic code names or random and unique-per-device process names are preferred. The same principle must be followed in naming the directory in which the agent's binaries reside and any names and values of process arguments.

Least Information Shared

Any agent artifacts deployed to devices must not contain sensitive information such as privileged credentials, debugging and dead code, or inline comments or documentation files that reveal details about server-side processing of agent-gathered metrics or other details about backend systems.

Transport Layer Security

To establish TLS secure channels for data transmission, an agent process must enforce all client-side validations, such as certificate chain and domain name validation, at the application level, if not enabled by default. Furthermore, an agent must use a root certificate store that contains trusted authorities and does not contain certificates belonging to compromised certificate issuers.

Secure Deployment

Any agent deployment mechanism, such as code push or sync and repositories containing its binaries, source code and any configuration files (including trusted root certificates), must be access-controlled to prevent unauthorized code injection or tampering. If the deployment mechanism relies on network communication, then use cryptographic methods to protect the integrity of deployment artifacts in transit.

Further Reading

- Security in AWS IoT (p. 199)
- Understanding the AWS IoT Security Model
- Redhat: A Bite of Python
- 10 common security gotchas in Python and how to avoid them
- What Is Least Privilege & Why Do You Need It?
- OWASP Embedded Security Top 10
- OWASP IoT Project

Device Advisor

Device Advisor is in preview and is subject to change.

Device Advisor is a cloud-based, fully managed test capability for validating IoT devices during device software development. Device Advisor provides pre-built tests that you can use to validate IoT devices for reliable and secure connectivity with AWS IoT Core, before deploying devices to production. By using Device Advisor, you can confirm that your devices can reliably connect to AWS IoT Core and follow security best practices. You can also download signed qualification reports to submit to the AWS Partner Network to get your device qualified for the AWS Partner Device Catalog without the need to send your device in and wait for it to be tested.

This chapter contains the following sections:

Any device that has been built to connect to AWS IoT Core can take advantage of Device Advisor. You can access Device Advisor from the AWS IoT console, or by using the AWS CLI/SDK. When you're ready to test your device, register it with AWS IoT Core and configure the device software with the Device Advisor endpoint. Then choose the prebuilt tests, configure them, run the tests on your device, and get the test results along with detailed logs or a qualification report.

Device Advisor is a test end point in the AWS cloud. You can test your devices by configuring them to connect to the test endpoint provided by the Device Advisor. After a device is configured to connect to the test endpoint, you can visit the Device Advisor’s console or use the AWS SDK to choose the tests you want to run on your devices. Device Advisor then manages the full lifecycle of a test, including the provisioning of resources, scheduling of the test process, managing the state machine, recording the device behavior, logging the results, and providing the final results in form of a test report.

Setting up

Device Advisor is in preview and is subject to change.

Before you use Device Advisor for the first time, complete the following tasks.

Prerequisites

- Create an IAM role to be used as your device role (p. 899)
- Create a custom managed policy for your Device Advisor user account (p. 900)
- Create an IAM user to use to run Device Advisor tests (p. 900)
- Create an AWS IoT thing and certificate (p. 901)

- Configure your test device (p. 901)

Create an IAM role to be used as your device role

This section shows you how to create an AWS account and add permissions to an IAM user that you can use to run Device Advisor tests on your devices.

1. Go to the AWS IAM console and log in to the account you use for Device Advisor testing.
2. First, create a policy that restricts the role permissions to the device thing policies. On the left navigation pane, chose **Policies**.
3. Choose **Create policy**.
4. Under **Create policy**, do the following.

    1. Choose **IoT** for **Service**.
    2. Under **Action**, check **All IoT actions (iot:*)**.
    3. Under **Resources**, you can select all resources. However, for best security practices, restrict **client**, **topic**, and **topicfilter**. If you choose to restrict these resources, follow the steps below.
        a. Choose **Specify client resource ARN for the Connect action**.
            i. Choose **Add ARN**.
            ii. Specify the **region**, **accountId**, and **clientId** in the visual ARN editor or manually specify the Amazon Resource Names (ARN) of the IoT topics you want to use to run test cases. The **clientId** is the MQTT **clientId** your device uses to interact with Device Advisor.
            iii. Choose **Add**.
        b. Choose **Specify topic resource ARN for the Receive and 1 more action**.
            i. Choose **Add ARN**.
            ii. Specify the **region**, **accountId**, and **topic name** in the visual ARN editor or manually specify the ARNs of the IoT topics you want to use to run test cases. The **topic name** is the MQTT **topic** your device use to publish messages to.
            iii. Choose **Add**.
        c. Choose **Specify topicfilter resource ARN for the Subscribe action**.
            i. Choose **Add ARN**.
            ii. Specify the **region**, **accountId**, and **topic name** in the visual ARN editor or manually specify the ARNs of the IoT topics you want to use to run test cases. The **topic name** is the MQTT **topic** your device uses to subscribe to.
            iii. Choose **Add**.
5. Choose **Review policy**.
6. Under **Review policy**, enter a **Name**.
7. Choose **Create policy**.
8. On the left navigation pane, Choose **Roles**.
9. Choose **Create Role**.
10. Under **Or select a service to view its use cases**, select **IoT**.
11. Under **Select your use case**, select **IoT**.
12. Choose **Next: Permissions**.
13. Under **Set permissions boundary**, Choose **Use a permissions boundary to control the maximum role permissions**, and then choose the policy you just created.
14. Choose **Next: Tags**.
15. Choose **Next: Review**.
16. Enter a **Role name** and a **Role description**.

17. Choose **Create role**.
18. Navigate to the role you created.
19. Choose **Attach policies** in the **Permissions** tab, and then choose the policy you created in **Step 4**.
20. Choose **Attach policy**.
21. Choose **Trust relationships** tab and choose **Edit trust relationship**.
22. Enter this policy:

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

23. Choose **Update Trust Policy**.

Create a custom managed policy for your Device Advisor user account

1. Navigate to the IAM console at https://console.aws.amazon.com/iam/ and log in to your account.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create Policy**, then choose the **JSON** tab.
4. Add the necessary permissions to use Device Advisor. The policy document can be found under Security best practices.
5. Choose **Review Policy**.
6. Enter a **Name** and **Description**.
7. Choose **Create Policy**.

Create an IAM user to use to run Device Advisor tests

Note
We recommend that you create an IAM user to use when you run Device Advisor tests. Although we don't recommend it, you can also use an IAM Admin user.

1. Navigate to the IAM console at https://console.aws.amazon.com/iam/ and log in to your account.
2. In the left navigation pane, Choose **Users**.
3. Choose **Add User**.
4. Enter a **User name**.
5. Select **Programmatic access**.
6. Choose **Next: Permissions**.
7. Choose **Attach existing policies directly**.
8. Enter the name of the custom-managed policy you created in the search box and then select the check box to the left of **Policy name**.

9. Choose **Next: Tags**.
10. Choose **Next: Review**.
11. Choose **Create user**.
12. Choose **Close**.

Create an AWS IoT thing and certificate

1. Go to the AWS IoT Core console. Log in to the account you use for Device Advisor testing, and in the left navigation pane, choose **Manage**.
2. If you have any existing things, choose **Create** to create a new thing. Otherwise, on the **You don't have any things yet** page, choose **Register a thing**.
3. On the **Creating AWS IoT things** page, choose **Create a single thing**.
4. On the **Add your device to the thing registry** page, enter a **Name** for your thing. Choose **Next**.
5. On the **Add a certificate for your thing** page, choose **Create certificate**. Notifications appears confirming that your thing and a certificate for your thing are created.
6. Copy the certificate you created in the previous step to your device. The correct location of the certificate depends on references to the certificate in your device's software or firmware.

Device Advisor requires access to your AWS resources (things, certificates, endpoint) on your behalf. Your IAM user must have the necessary permissions. Device Advisor will also publish logs to Amazon CloudWatch if you attach the necessary permissions policy to your IAM user.

Configure your test device

Configure your test device

Device Advisor uses the server name indication (SNI) TLS extension to apply TLS configurations. Devices must use this extension when connecting and pass a server name that is identical to the Device Advisor test endpoint.

Device Advisor allows the TLS connection when test is in "Running" state and denies the TLS connection before and after each test run. Device connect retry mechanism is recommended to have full-automated testing experience with Device Advisor. If you run a test suite with more than one test case say - TLS connect, MQTT connect and MQTT publish then we recommend that you have a mechanism built for your device to try connecting to our test end point every 5 seconds for one to two minutes. This will enable you to run multiple test cases in sequence in an automated manner.

- You must configure the firmware or software on the device that you'll use for testing to connect to the Device Advisor test endpoint for your account. The command to get the test endpoint is:

```
aws iot describe-endpoint --endpoint-type iot:DeviceAdvisor --region us-east-1
```

Getting started with Device Advisor in the console

Device Advisor is in preview and is subject to change.

This tutorial helps you quickly get started with Device Advisor in the console. Device Advisor offers features such as required tests and signed qualification reports to qualify and list devices in the AWS Partner Device Catalog as per the AWS IoT Core qualification program.

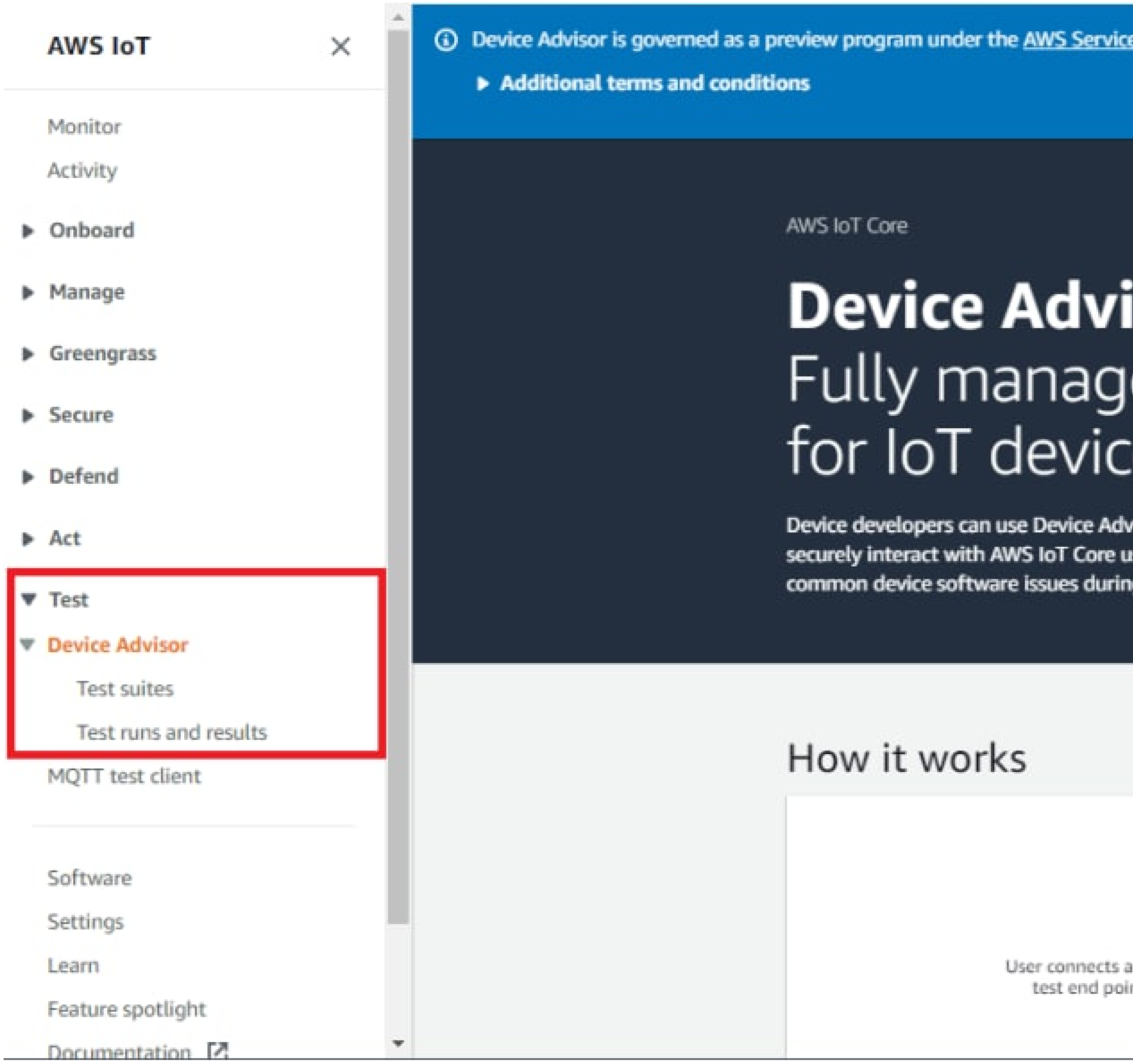
For more detailed instructions about using Device Advisor, see Device Advisor workflow (p. 913) and Device Advisor detailed console workflow (p. 917).

Note
Only the us-east-1 Region is supported for preview.

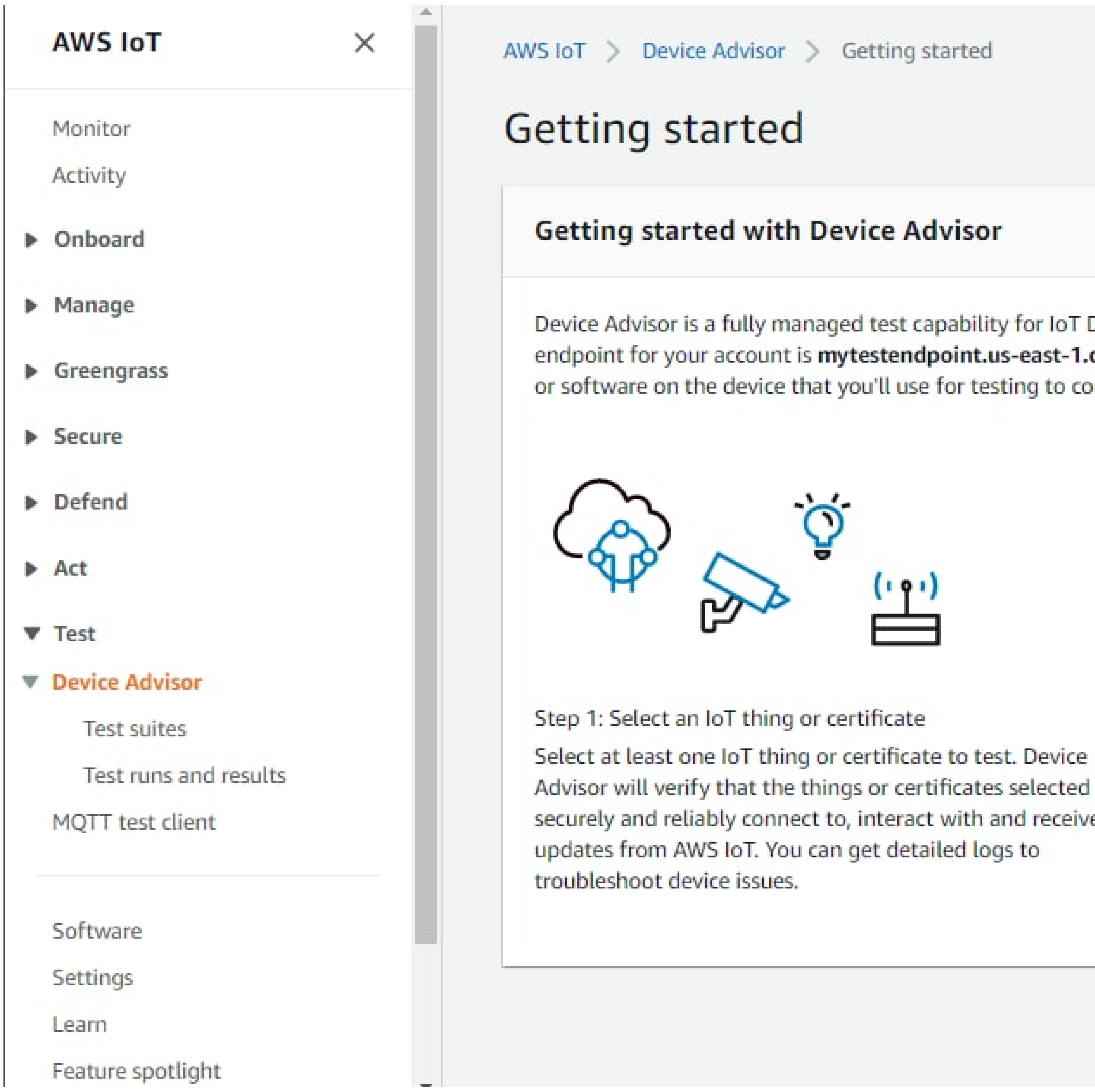
To complete this tutorial, you need to complete the steps outlined in Setting up (p. 898).

Getting started

1. In the AWS IoT console, in the navigation pane, expand **Test**, **Device Advisor** and then choose **Start walkthrough**.



2. The **Getting started with Device Advisor** gives an overview of the steps required to create a test suite and run tests against your device. You can also find the Device Advisor test endpoint for your account. You must configure the firmware or software on the device that you'll use for testing to connect to this test endpoint.



Getting started

Getting started with Device Advisor

Device Advisor is a fully managed test capability for IoT D
endpoint for your account is **mytestendpoint.us-east-1.c**
or software on the device that you'll use for testing to co

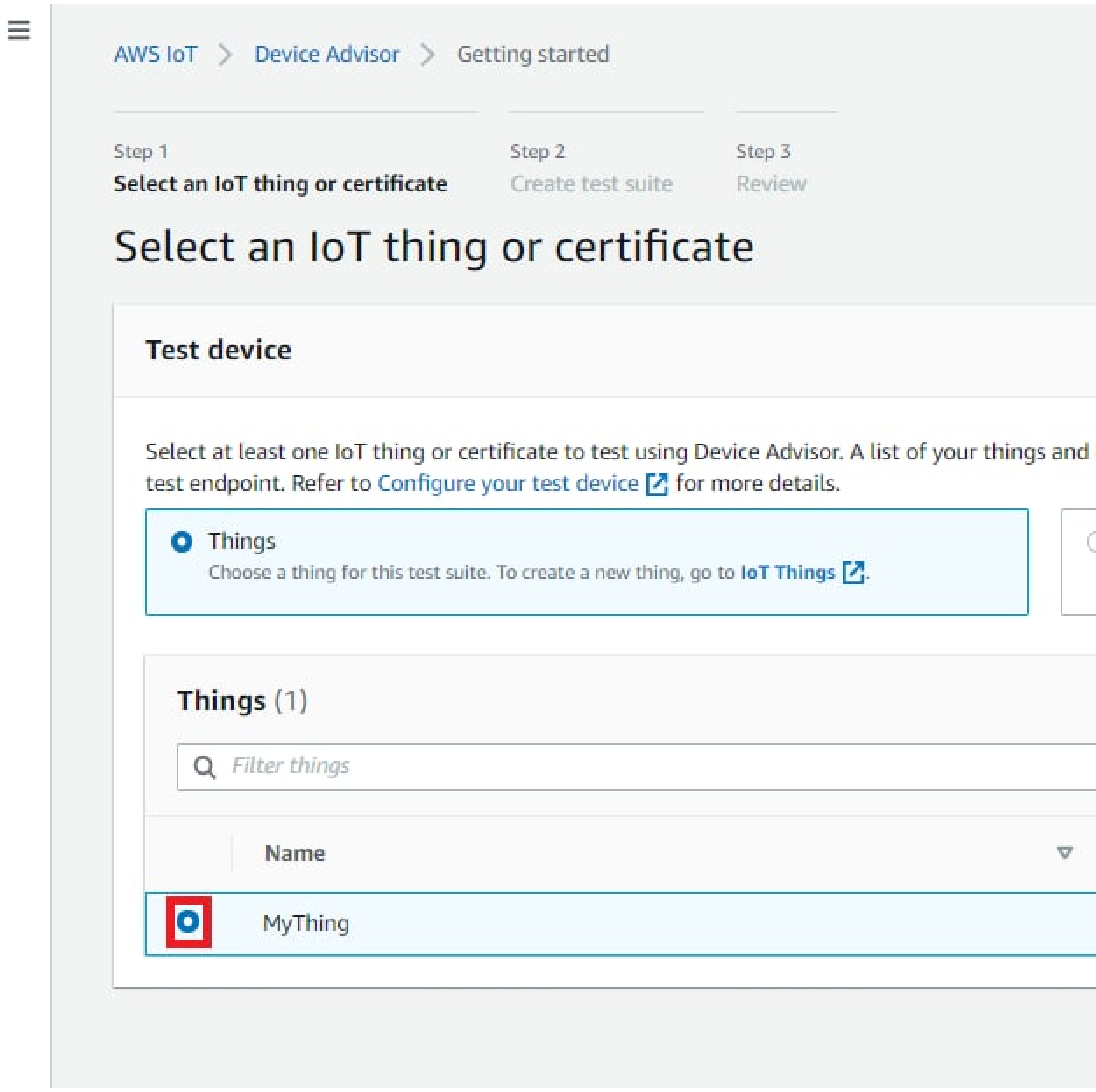
Step 1: Select an IoT thing or certificate

Select at least one IoT thing or certificate to test. Device Advisor will verify that the things or certificates selected securely and reliably connect to, interact with and receive updates from AWS IoT. You can get detailed logs to troubleshoot device issues.

Once you've reviewed the screen, choose **Next**.

3. In **Step 1**, you'll need to select an AWS IoT thing or certificate to test using Device Advisor. If you don't have any existing things or certificates, see Setting up (p. 898).

   If you do see the AWS IoT thing or certificate that you've configured with your Device Advisor test endpoint, select that device from the list and choose **Next**.



AWS IoT > Device Advisor > Getting started

Step 1
Select an IoT thing or certificate

Step 2
Create test suite

Step 3
Review

Select an IoT thing or certificate

Test device

Select at least one IoT thing or certificate to test using Device Advisor. A list of your things and
test endpoint. Refer to Configure your test device for more details.

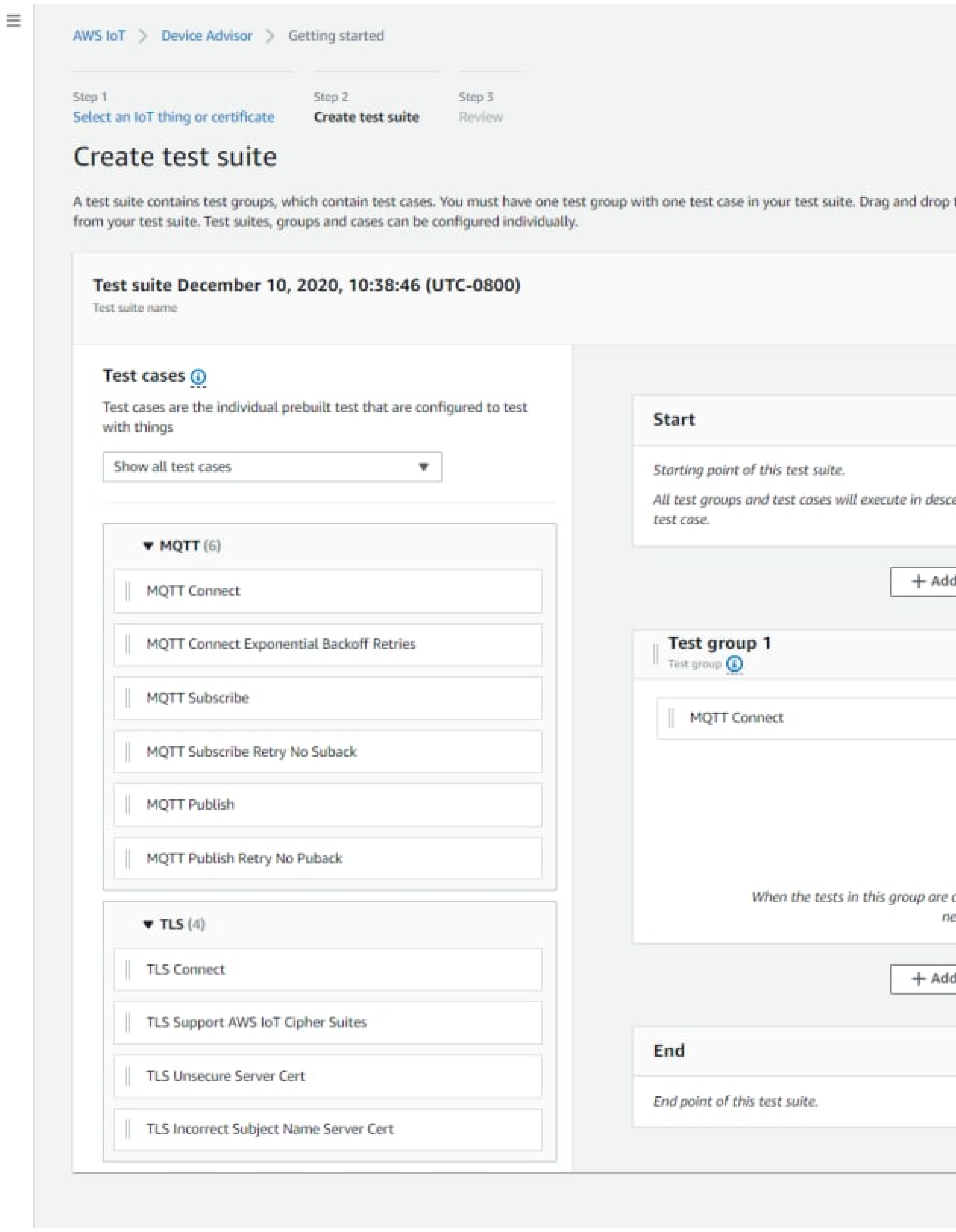
Things
Choose a thing for this test suite. To create a new thing, go to **IoT Things**.

Things (1)

Filter things

	Name
●	MyThing

4. In **Step 2**, you can create and configure a custom test suite. A custom test suite must have at least one test group, and each test group must have at least one test case. We've added the **MQTT Connect** test case for you to get started.
5. Choose **Test suite properties**. You must supply the test suite properties when you create your test suite.



AWS IoT > Device Advisor > Getting started

Step 1
Select an IoT thing or certificate

Step 2
Create test suite

Step 3
Review

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case in your test suite. Drag and drop t
from your test suite. Test suites, groups and cases can be configured individually.

Test suite December 10, 2020, 10:38:46 (UTC-0800)
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

MQTT (6)
- MQTT Connect
- MQTT Connect Exponential Backoff Retries
- MQTT Subscribe
- MQTT Subscribe Retry No Suback
- MQTT Publish
- MQTT Publish Retry No Puback

TLS (4)
- TLS Connect
- TLS Support AWS IoT Cipher Suites
- TLS Unsecure Server Cert
- TLS Incorrect Subject Name Server Cert

Start
Starting point of this test suite.
All test groups and test cases will execute in desce
test case.

+ Add

Test group 1
Test group
- MQTT Connect

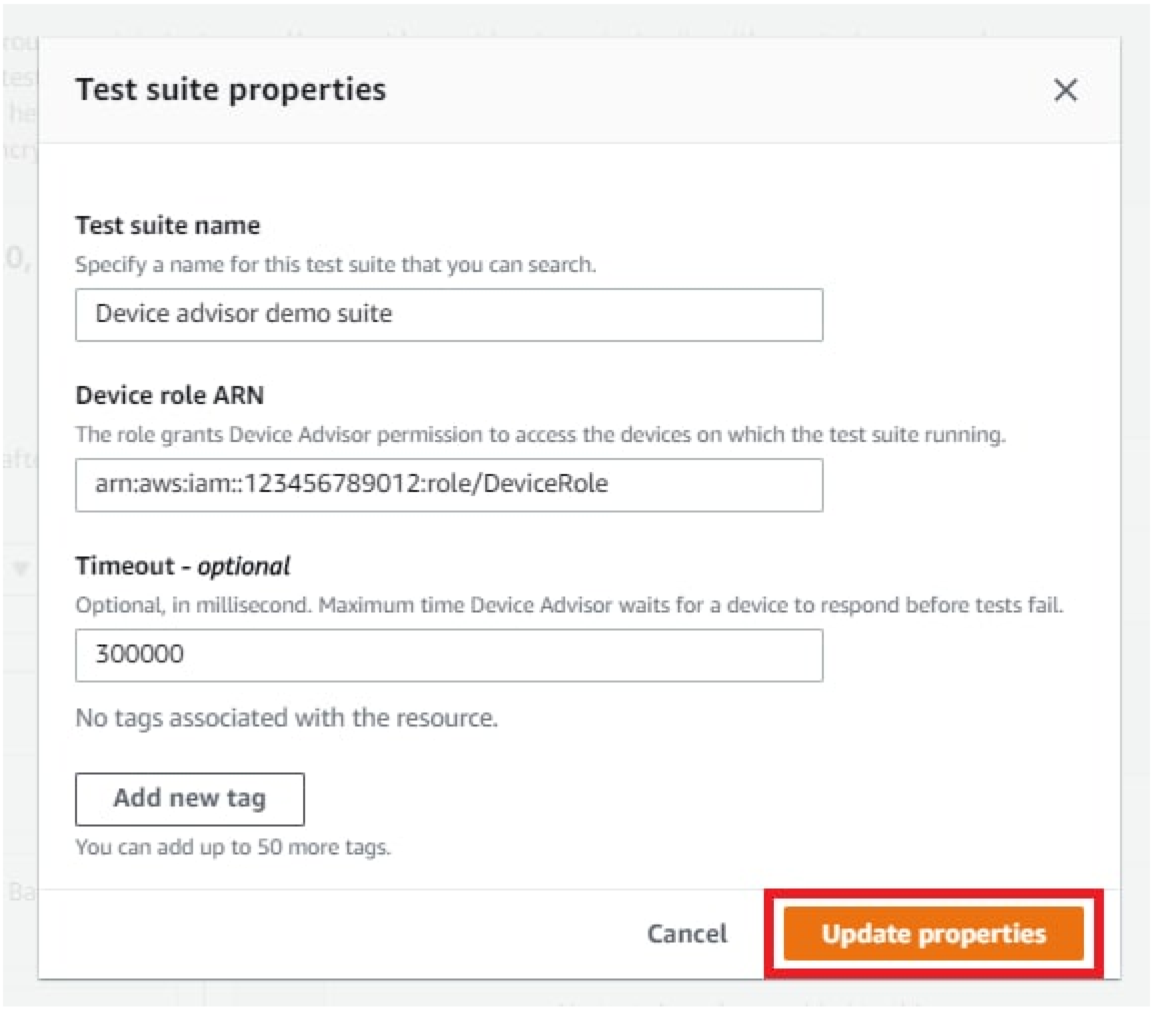
When the tests in this group are c
ne

+ Add

End
End point of this test suite.

You can configure the suite-level properties here.

- **Test suite name**: You can create the suite with a custom name.
- **Device role ARN**: Provide the device role ARN that was created as part of Setting up (p. 898).
- **Timeout** (optional): The timeout in milliseconds for each test case in the current test suite. The default timeout value will be used if no timeout value is specified.
- **Tags** (optional): Add tags to the test suite that you are going to create.



Test suite properties

Test suite name
Specify a name for this test suite that you can search.
Device advisor demo suite

Device role ARN
The role grants Device Advisor permission to access the devices on which the test suite running.
arn:aws:iam::123456789012:role/DeviceRole

Timeout - *optional*
Optional, in millisecond. Maximum time Device Advisor waits for a device to respond before tests fail.
300000

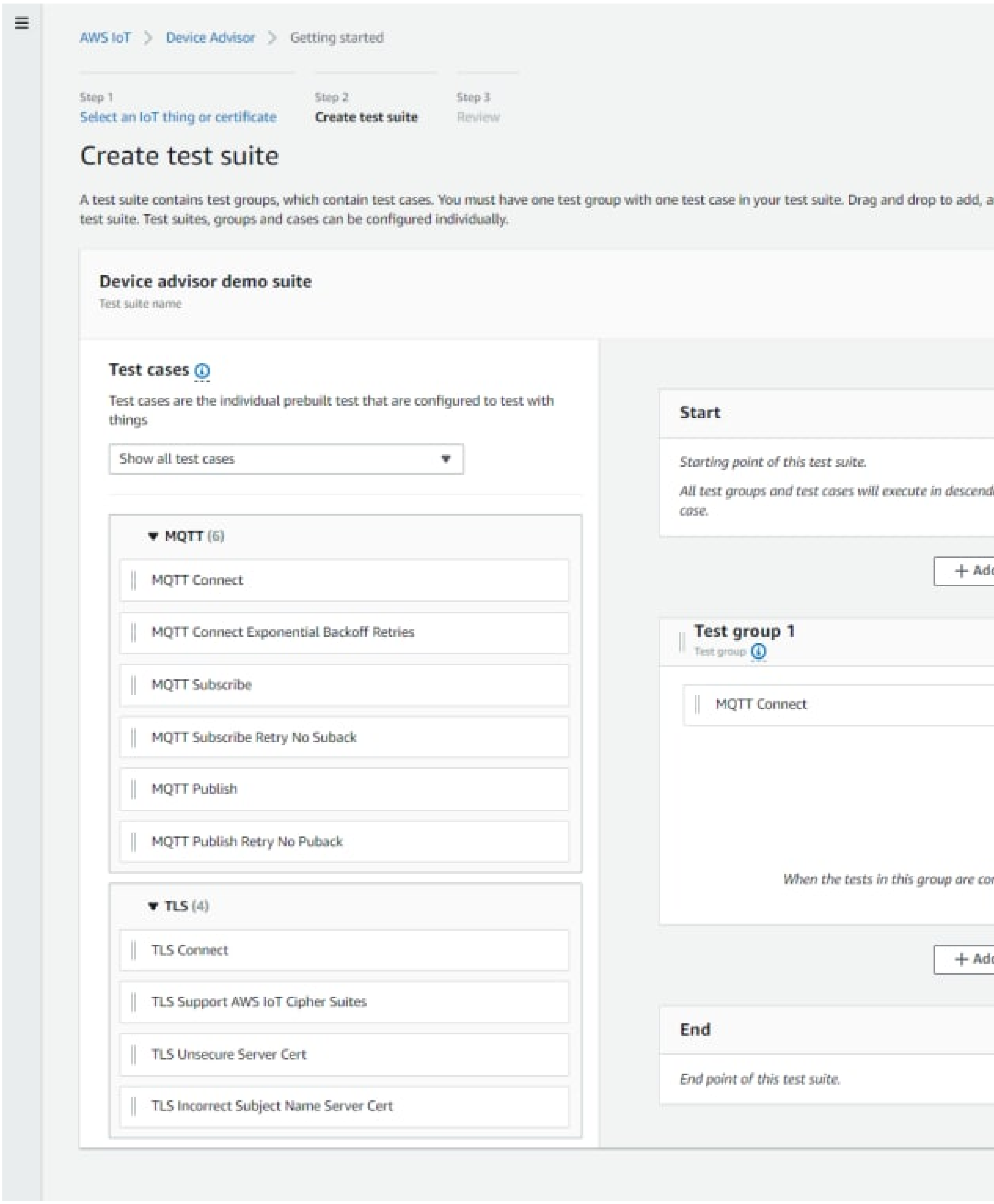
No tags associated with the resource.

You can add up to 50 more tags.

Choose **Update properties** when done.

6. *(Optional)* You can update the test suite group configuration by choosing **Edit** next to the test group name.

   - **Name**: You can update the group with a custom name.
   - **Timeout** (optional): The timeout in milliseconds for each test case in the current test suite. The default timeout value will be used if no timeout value is specified.



AWS IoT > Device Advisor > Getting started

Step 1 Select an IoT thing or certificate
Step 2 Create test suite
Step 3 Review

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case in your test suite. Drag and drop to add, a
test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

MQTT (6)
- MQTT Connect
- MQTT Connect Exponential Backoff Retries
- MQTT Subscribe
- MQTT Subscribe Retry No Suback
- MQTT Publish
- MQTT Publish Retry No Puback

TLS (4)
- TLS Connect
- TLS Support AWS IoT Cipher Suites
- TLS Unsecure Server Cert
- TLS Incorrect Subject Name Server Cert

Start
Starting point of this test suite.
All test groups and test cases will execute in descend
case.

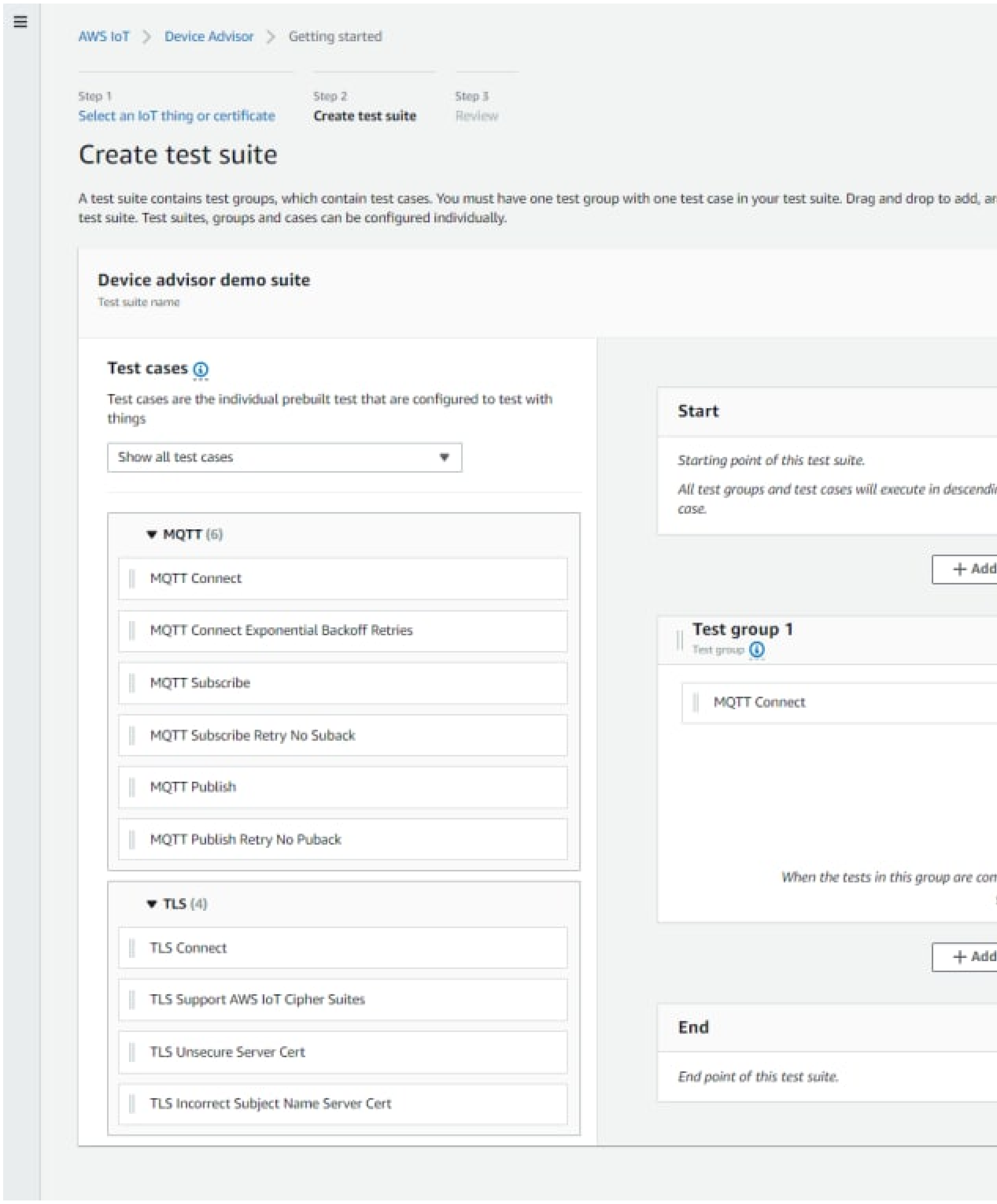
Test group 1
Test group
MQTT Connect
When the tests in this group are co

End
End point of this test suite.

Choose **Done**.

7. *(Optional)* You can update the test case configuration by choosing **Edit** next to the test case name.

- **Name**: You can update the test case with a custom name.
- **Timeout** (optional): The timeout in milliseconds for the selected test case. The default timeout value will be used if no timeout value is specified.



AWS IoT > Device Advisor > Getting started

Step 1
Select an IoT thing or certificate

Step 2
Create test suite

Step 3
Review

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case in your test suite. Drag and drop to add, ar
test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases

Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

MQTT (6)
- MQTT Connect
- MQTT Connect Exponential Backoff Retries
- MQTT Subscribe
- MQTT Subscribe Retry No Suback
- MQTT Publish
- MQTT Publish Retry No Puback

TLS (4)
- TLS Connect
- TLS Support AWS IoT Cipher Suites
- TLS Unsecure Server Cert
- TLS Incorrect Subject Name Server Cert

Start

Starting point of this test suite.

All test groups and test cases will execute in descendi
case.

+ Add

Test group 1
Test group

MQTT Connect

When the tests in this group are con

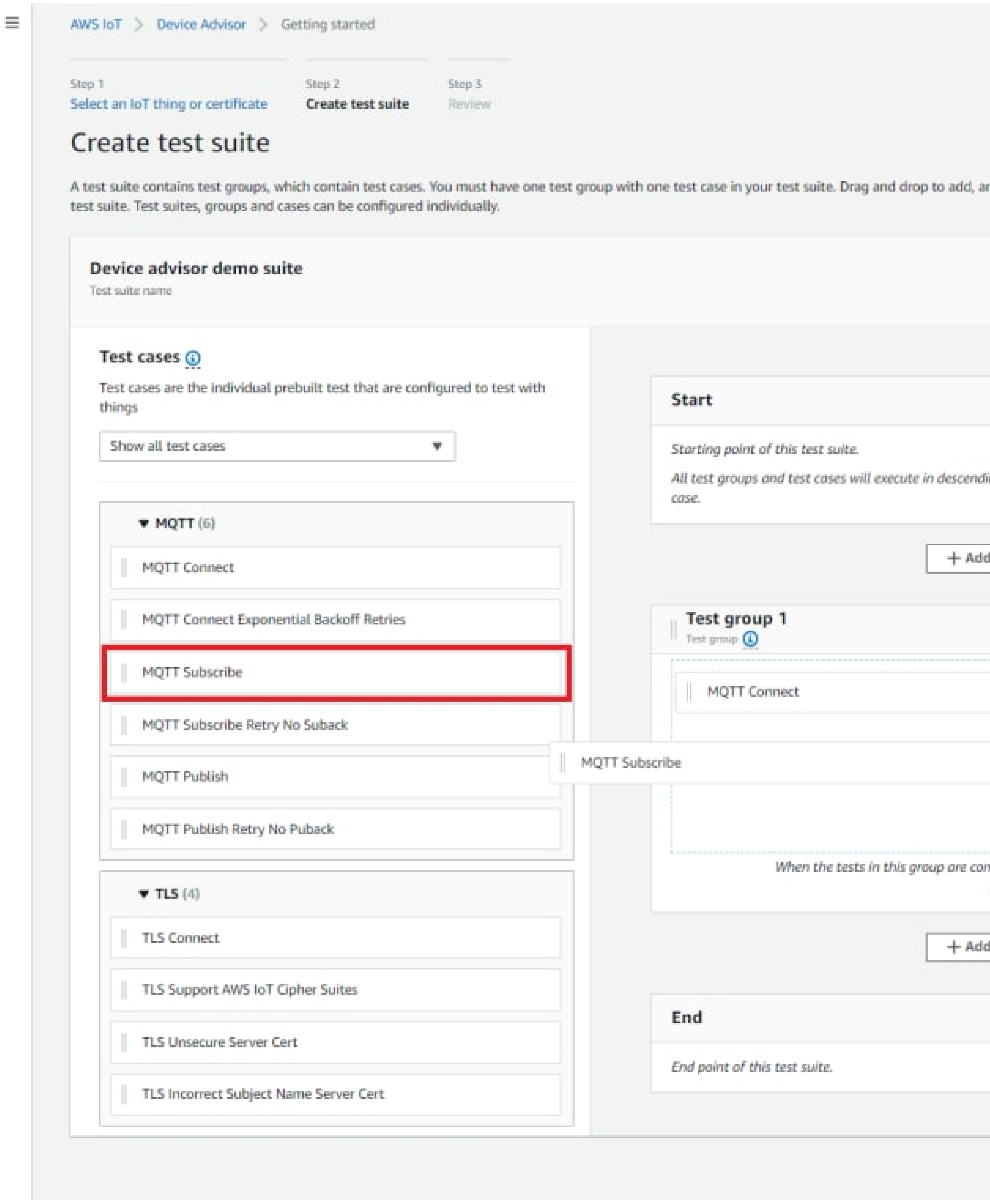
+ Add

End

End point of this test suite.

Choose **Done**.

8. *(Optional)* To add more test groups to the test suite, choose **Add test group** and configure accordingly.
9. *(Optional)* To add more test cases, drag the test cases under **Test cases** into any of the test groups you have.


AWS IoT > Device Advisor > Getting started

Step 1 Select an IoT thing or certificate
Step 2 Create test suite
Step 3 Review

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case in your test suite. Drag and drop to add, ar
test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

MQTT (6)
MQTT Connect
MQTT Connect Exponential Backoff Retries
MQTT Subscribe
MQTT Subscribe Retry No Suback
MQTT Publish
MQTT Publish Retry No Puback

TLS (4)
TLS Connect
TLS Support AWS IoT Cipher Suites
TLS Unsecure Server Cert
TLS Incorrect Subject Name Server Cert

Start
Starting point of this test suite.
All test groups and test cases will execute in descendi
case.

+ Add

Test group 1
Test group
MQTT Connect
MQTT Subscribe
When the tests in this group are con

+ Add

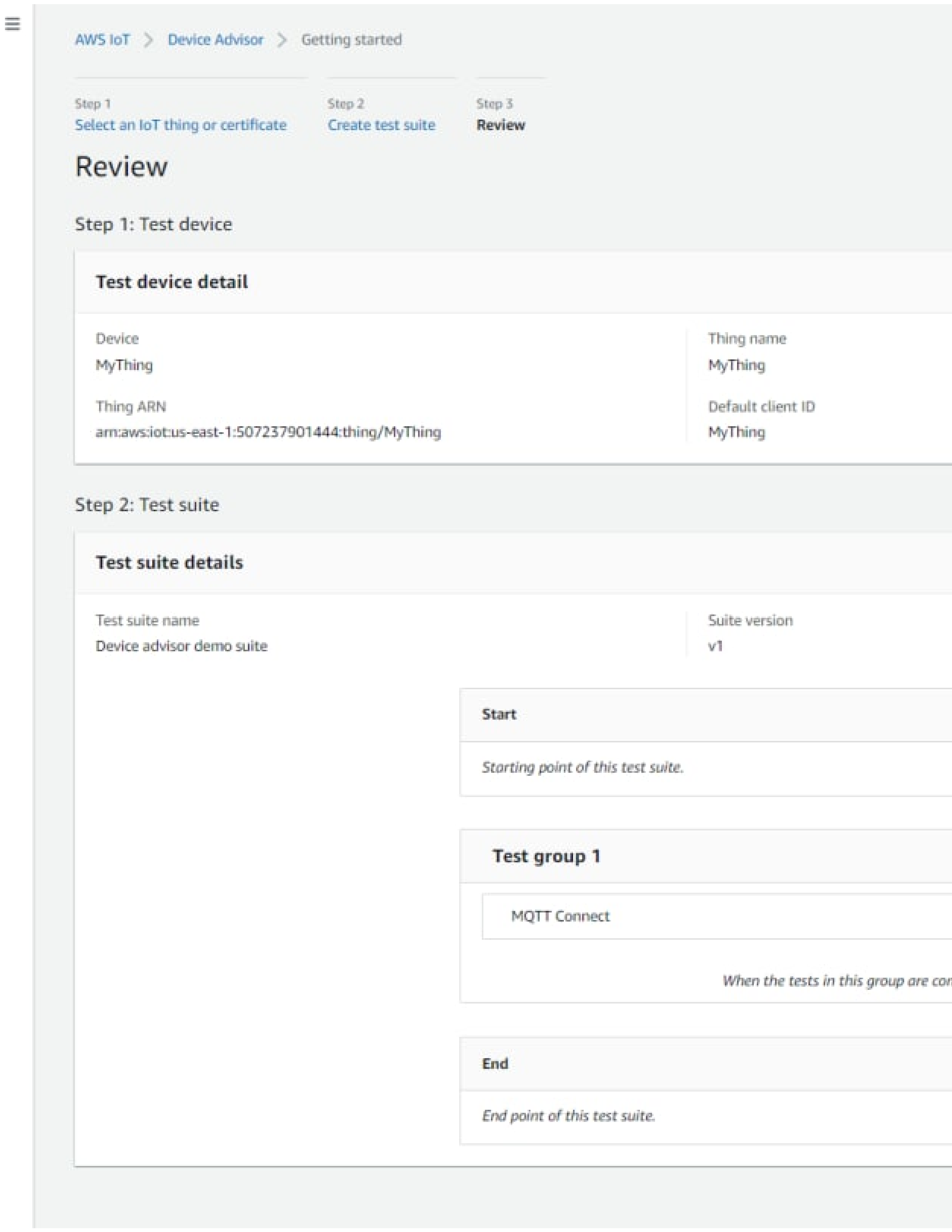
End
End point of this test suite.

10. Test groups and test cases can be reordered by dragging. Device Advisor will run tests in the order they are defined.

    After you've configured your test suite, choose **Next**.
11. **Step 3** provides you with an overview of the selected test device and test suite that you've configured. If you'd like to make changes, choose **Edit**.

    Note
    You must connect your selected test device to the Device Advisor test endpoint before starting the suite run. We recommend that you have a mechanism built for your device to try connecting to our test endpoint every 5 seconds for one to two minutes.

    Choose **Run** to create the test suite and run the tests against your device.



AWS IoT > Device Advisor > Getting started

Step 1
Select an IoT thing or certificate

Step 2
Create test suite

Step 3
Review

Review

Step 1: Test device

Test device detail

Device
MyThing

Thing ARN
arn:aws:iot:us-east-1:507237901444:thing/MyThing

Thing name
MyThing

Default client ID
MyThing

Step 2: Test suite

Test suite details

Test suite name
Device advisor demo suite

Suite version
v1

Start

Starting point of this test suite.

Test group 1

MQTT Connect

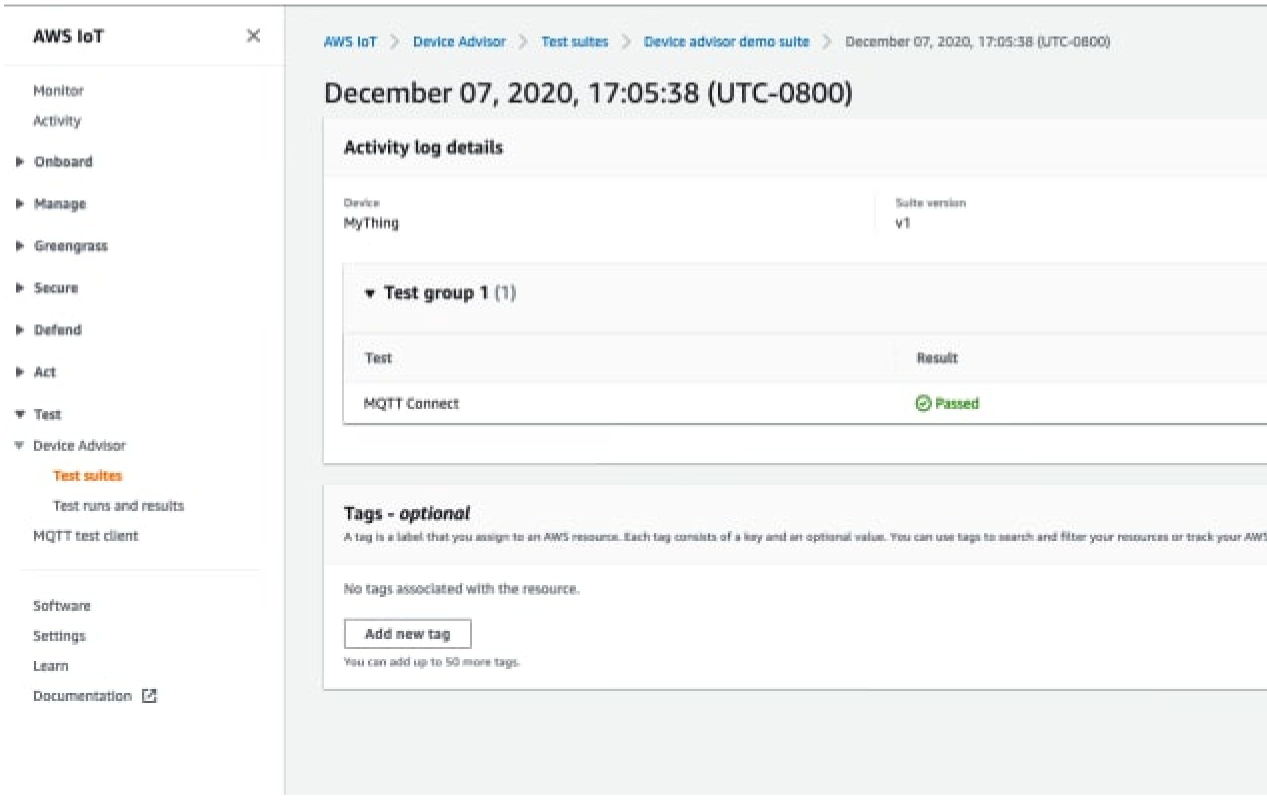
When the tests in this group are co

End

End point of this test suite.

12. In the navigation pane, expand **Test**, **Device Advisor** and choose **Test runs and results** to view the run details and logs.

    Select the test suite run that has been started to view the run details and logs.



December 07, 2020, 17:05:38 (UTC-0800)

Activity log details

Device: MyThing
Suite version: v1

Test group 1 (1)

Test	Result
MQTT Connect	Passed

Tags - *optional*

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS

No tags associated with the resource.

You can add up to 50 more tags.

13. To access the CloudWatch logs for the suite run:

    - Choose **Test suite log** to view the CloudWatch logs for the test suite run.
    - Choose **Test case log** for any test case to view test case-specific CloudWatch logs.
14. Based on your test results, troubleshoot your device until all tests pass.

Device Advisor workflow

Device Advisor is in preview and is subject to change.

This tutorial provides instructions on how to create a custom test suite and run tests against the device you want to test in the console. After the tests are complete, you can view the test results and detailed logs.

Tutorials

- Prerequisites (p. 914)
- Create a test suite definition (p. 914)
- Get a test suite run (p. 915)
- Start a test suite run (p. 916)
- Get a test suite run (p. 916)

- Get a qualification report for a successful qualification test suite run (p. 916)

Prerequisites

To complete this tutorial, you need to complete the steps outlined in Setting up (p. 898).

Create a test suite definition

First, install an AWS SDK.

Use the CreateSuiteDefinition (p. 937) API to create a test suite definition.

`rootGroup` syntax

A root group is a JSON string that specifies which test cases are included in your test suite and any necessary configurations for those test cases. Use the root group to structure and order your test suite in any way you like. The hierarchy of a test suite is:

```
test suite # test group(s) # test case(s)
```

A test suite must have at least one test group, and each test group must have at least one test case. Device Advisor runs tests in the order in which you define the test groups and test cases.

Each root group has this basic structure:

```
{
    "configuration": {  // for all tests in the test suite
        "": ""
    }
    "tests": [{
        "name": ""
        "configuration": {  // for all sub-groups in this test group
            "": ""
        },
        "tests": [{
            "name": ""
            "configuration": {  // for all test cases in this test group
                "": ""
            },
            "test": {
                "id": ""
                "version": ""
            }
        }]
    }]
}
```

A block that contains a `"name"`, `"configuration"`, and `"tests"` is referred to as a "group definition". A block that contains a `"name"`, `"configuration"`, and `"test"` is referred to as a "test case definition". Each `"test"` block that contains an `"id"` and `"version"` is referred to as a "test case".

See Device Advisor test cases (p. 931) for information on how to fill in the `"id"` and `"version"` fields for each test case (`"test"` block). That section also contains information on the available `"configuration"` settings.

The following is an example of a root group configuration that specifies the "MQTT Connect Happy Case" and "MQTT Connect Exponential Backoff Retries" test cases along with descriptions of the configuration fields.

```
{
    "configuration": {},  // Suite-level configuration
    "tests": [            // Group definitions should be provided here
      {
        "name": "My_MQTT_Connect_Group",  // Group definition name
        "configuration": {}               // Group definition-level configuration,
        "tests": [                        // Test case definitions should be provided here
        {
            "name": "My_MQTT_Connect_Happy_Case",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 300000        // Test case definition-level
 configuration
            },
            "test": {
                "id": "MQTT_Connect",              // test case id
                "version": "0.0.0"                 // test case version
            }
        },
        {
            "name": "My_MQTT_Connect_Exponential_Backoff",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 600000                 // Test case definition-level
 configuration
            },
            "test": {
                "id": "MQTT_Connect_Exponential_Backoff_Retries",  // test case id
                "version": "0.0.0"                                 // test case version
            }
        }]
    }]
}
```

You must supply the root group configuration when you create your test suite definition. Save the `suiteDefinitionId` that is returned in the response object—it's used to retrieve your test suite definition information and to run your test suite.

Here is a Java SDK example:

```
response = iotDeviceAdvisorClient.createSuiteDefinition(
        CreateSuiteDefinitionRequest.builder()
            .suiteDefinitionConfiguration(SuiteDefinitionConfiguration.builder()
                .suiteDefinitionName("your-suite-definition-name")
                .devices(
                    DeviceUnderTest.builder()
                        .thingArn("your-test-device-thing-arn")
                        .certificateArn("your-test-device-certificate-arn")
                        .build()
                )
                .rootGroup("your-root-group-configuration")
                .devicePermissionRoleArn("your-device-permission-role-arn")
                .build()
            )
            .build()
)
```

Get a test suite run

After you create your test suite definition, you will receive the `suiteDefinitionId` in the response object of the `CreateSuiteDefinition` API. Use the GetSuiteDefinition (p. 940) API to retrieve the test suite definitions information and to verify that all the information in the test suite definition is correct.

You may see that there are new `id` fields within each of the group and test case definitions in the root group that is returned. This is expected and you can use these IDs to run a subset of your test suite definition.

Java SDK example:

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
)
```

Start a test suite run

After you've successfully created a test suite definition and configured your test device to connect to your Device Advisor test endpoint, run your test suite with the StartSuiteRun (p. 957) API. Use either `certificateArn` or `thingArn` to run the test suite. If both are configured, the certificate will be used if it belongs to the thing.

SDK example:

```
response = iotDeviceAdvisorClient.startSuiteRun(StartSuiteRunRequest.builder()
.suiteDefinitionId("your-suite-definition-id")
.suiteRunConfiguration(SuiteRunConfiguration.builder()
    .primaryDevice(DeviceUnderTest.builder()
        .certificateArn("your-test-device-certificate-arn")
        .thingArn("your-test-device-thing-arn")
        .build())
    .build())
.build())
```

Save the `suiteRunId` that is returned in the response—you will use this to retrieve the results of this test suite run.

Get a test suite run

After you start a test suite run, you can check its progress and, eventually, its results with the GetSuiteRun (p. 943) API.

SDK example:

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

Get a qualification report for a successful qualification test suite run

If you successfully run a qualification test suite and all test cases have passed, you can retrieve a qualification report using the GetSuiteRunReport API. This qualification report can be used to qualify your device with the AWS IoT Core qualification program. To determine whether your test suite is a qualification test suite, you can check whether the `intendedForQualification` parameter is set to `true`. Once you call the GetSuiteRunReport API, the download URL returned is available for you to download for 90 seconds. If more than 90 seconds elapse from the previous time you called the GetSuiteRunReport API, call the API again to retrieve a valid URL.

SDK example:

```
// Using the SDK, call the getSuiteRunReport API.

response = iotDeviceAdvisorClient.getSuiteRunReport(
    GetSuiteRunReportRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .suiteRunId("your-suite-run-id")
        .build()
)
```

Device Advisor detailed console workflow

In this tutorial, you'll create a custom test suite and run tests against the device you want to test in the console. After the tests are complete, you can view the test results and detailed logs.

Tutorials

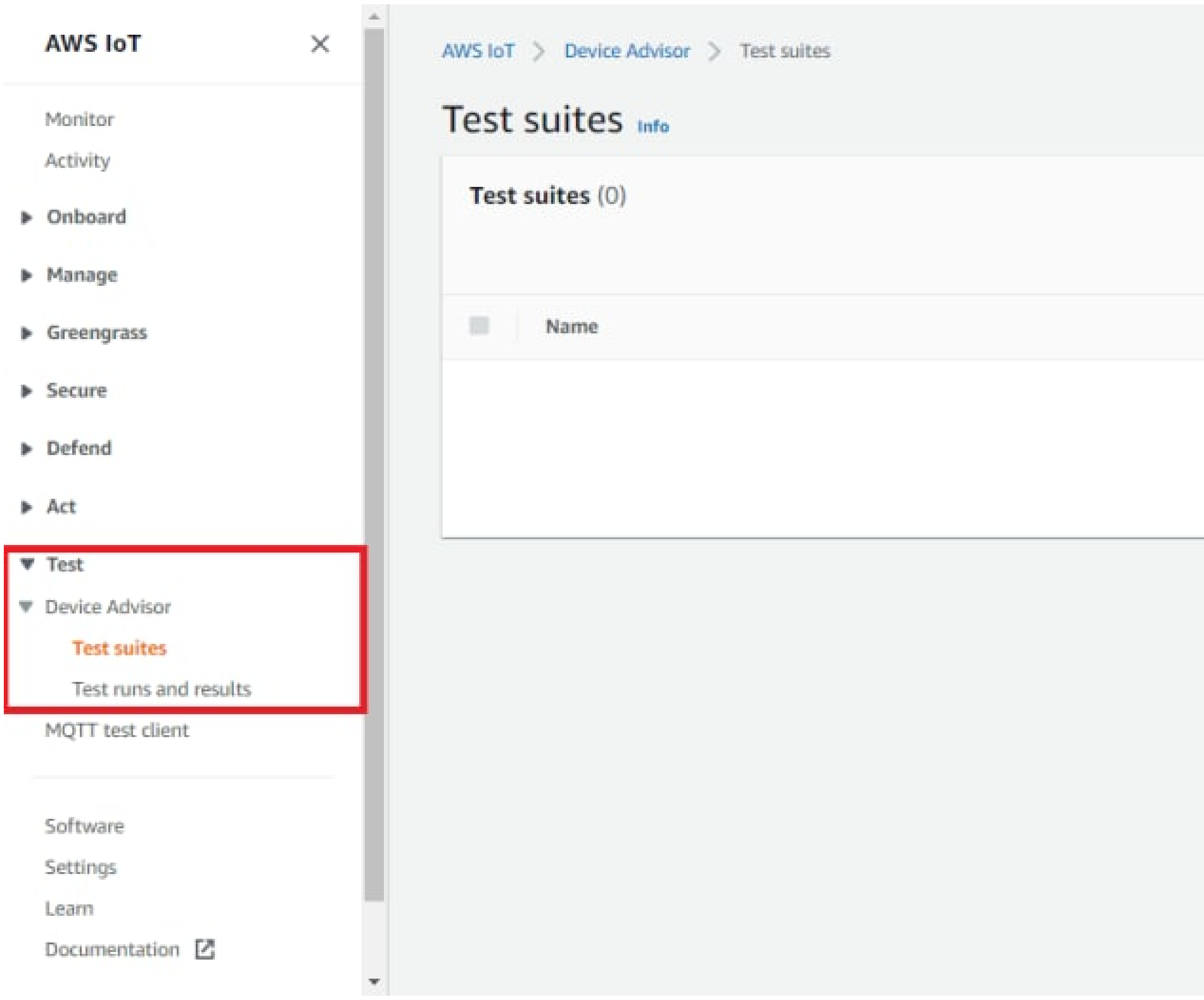
Prerequisites

To complete this tutorial, you need to complete the steps outlined in Setting up (p. 898).

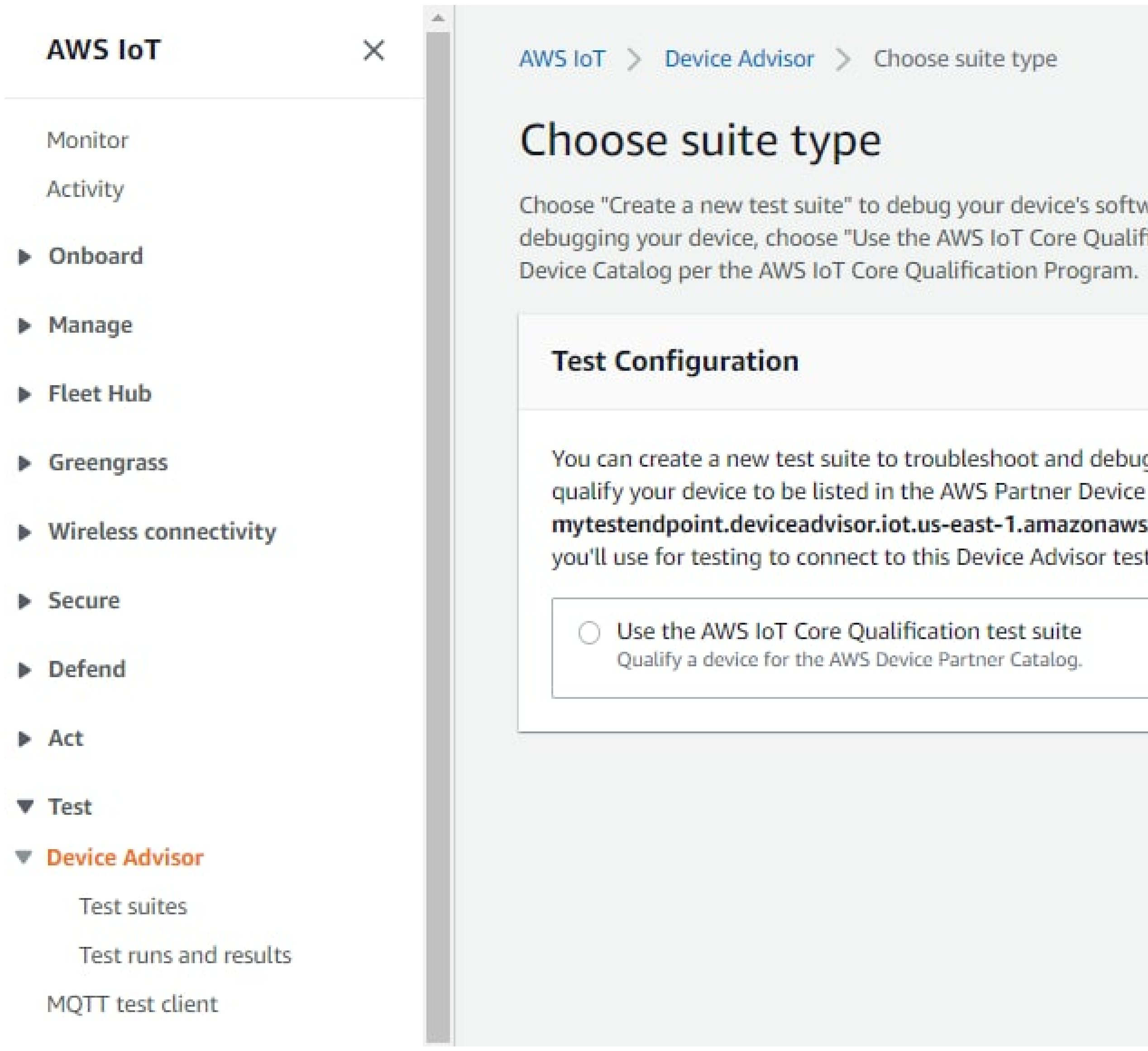
Create a test suite definition

1. In the AWS IoT console, in the navigation pane, expand **Test**, **Device Advisor** and then choose **Test suites**.

    Note
    Only the us-east-1 Region is supported for preview.



2. Select **Create Test Suite**. Choose between `Use the AWS Qualification test suite` and `Create a new test suite`.



Choose suite type

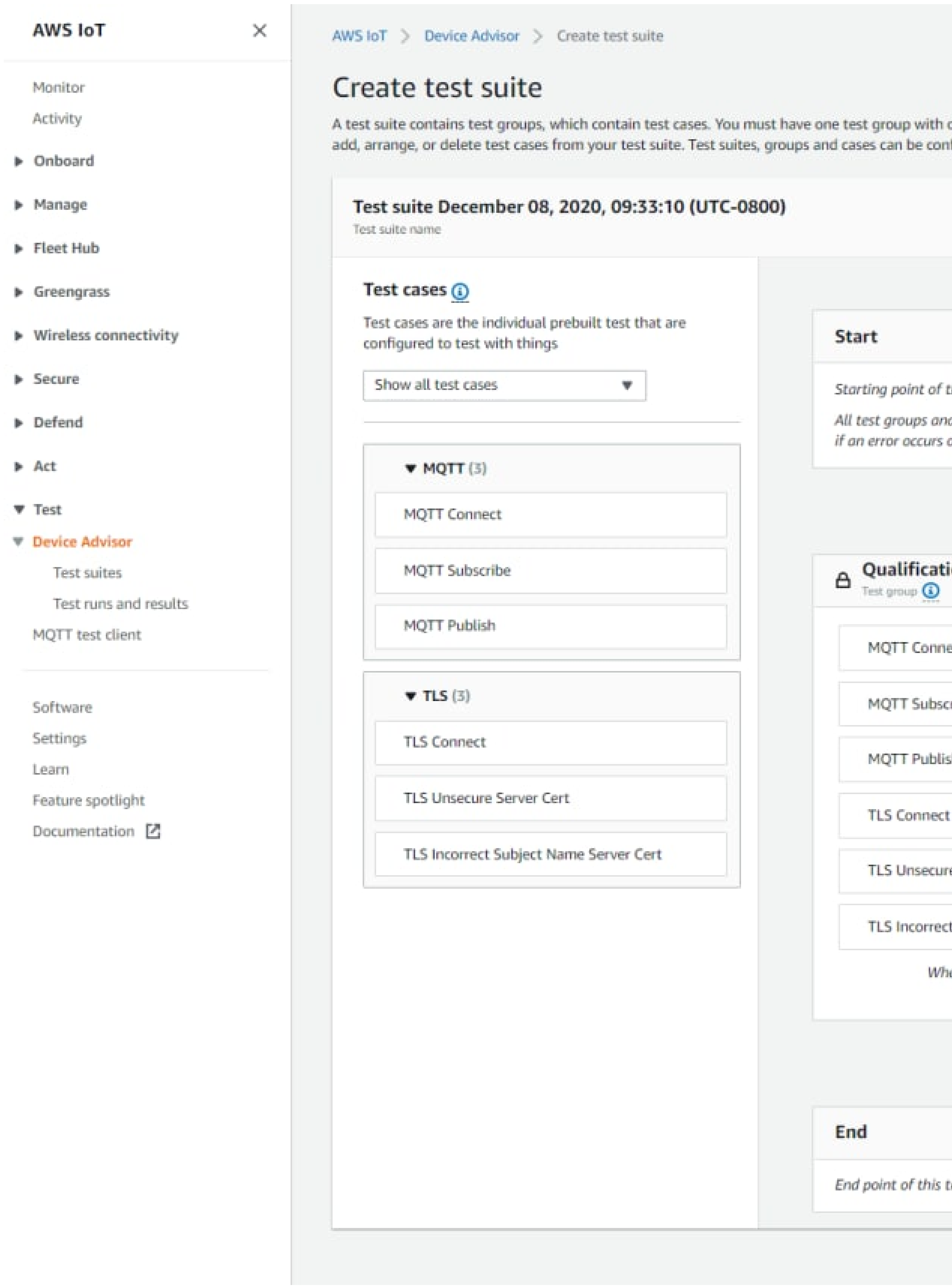
Choose "Create a new test suite" to debug your device's softw
debugging your device, choose "Use the AWS IoT Core Qualif
Device Catalog per the AWS IoT Core Qualification Program.

Test Configuration

You can create a new test suite to troubleshoot and debug
qualify your device to be listed in the AWS Partner Device
mytestendpoint.deviceadvisor.iot.us-east-1.amazonaws
you'll use for testing to connect to this Device Advisor test

Use the AWS IoT Core Qualification test suite
Qualify a device for the AWS Device Partner Catalog.

Choose `Use the AWS Qualification test suite` if you'd like to qualify and list your device to the AWS Partner Device Catalog. By choosing this option, test cases required for qualification of your device to the AWS IoT Core qualification program are pre-selected. Test groups and test cases cannot be added or removed. However, you'll still need to configure the test suite properties.



Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with c
add, arrange, or delete test cases from your test suite. Test suites, groups and cases can be con

Test suite December 08, 2020, 09:33:10 (UTC-0800)
Test suite name

Test cases

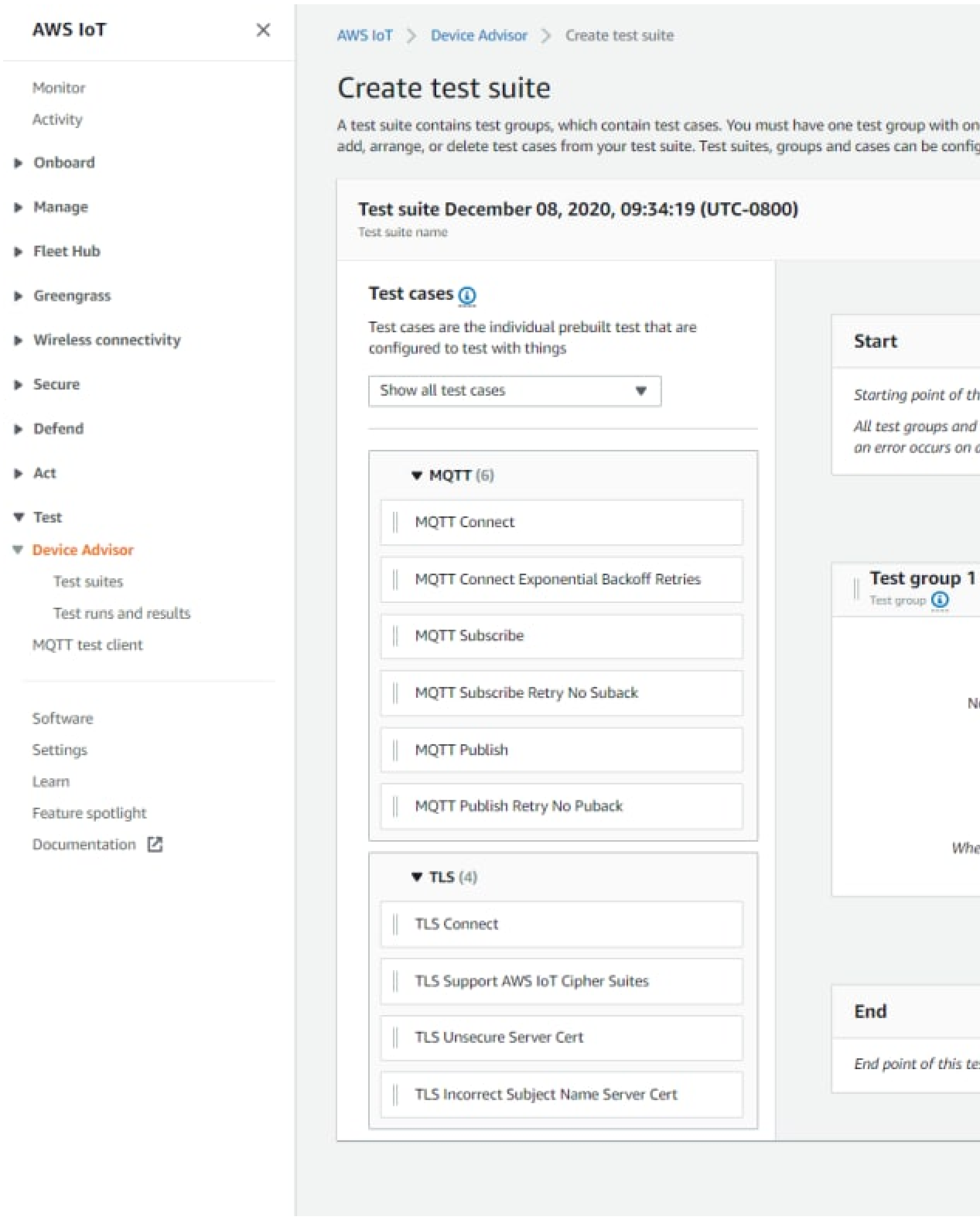
Test cases are the individual prebuilt test that are configured to test with things

Starting point of t
All test groups and
if an error occurs c

End point of this t

Choose `Create a new test suite` to create and configure a custom test suite. We recommend starting with this option for initial testing and troubleshooting. A custom test suite must have at least one test group, and each test group must have at least one test case. For the purpose of this tutorial, we'll select this option and select **Next**.

3. Choose **Test suite properties**. You must add the test suite properties when you create your test suite.



Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with on
add, arrange, or delete test cases from your test suite. Test suites, groups and cases can be config

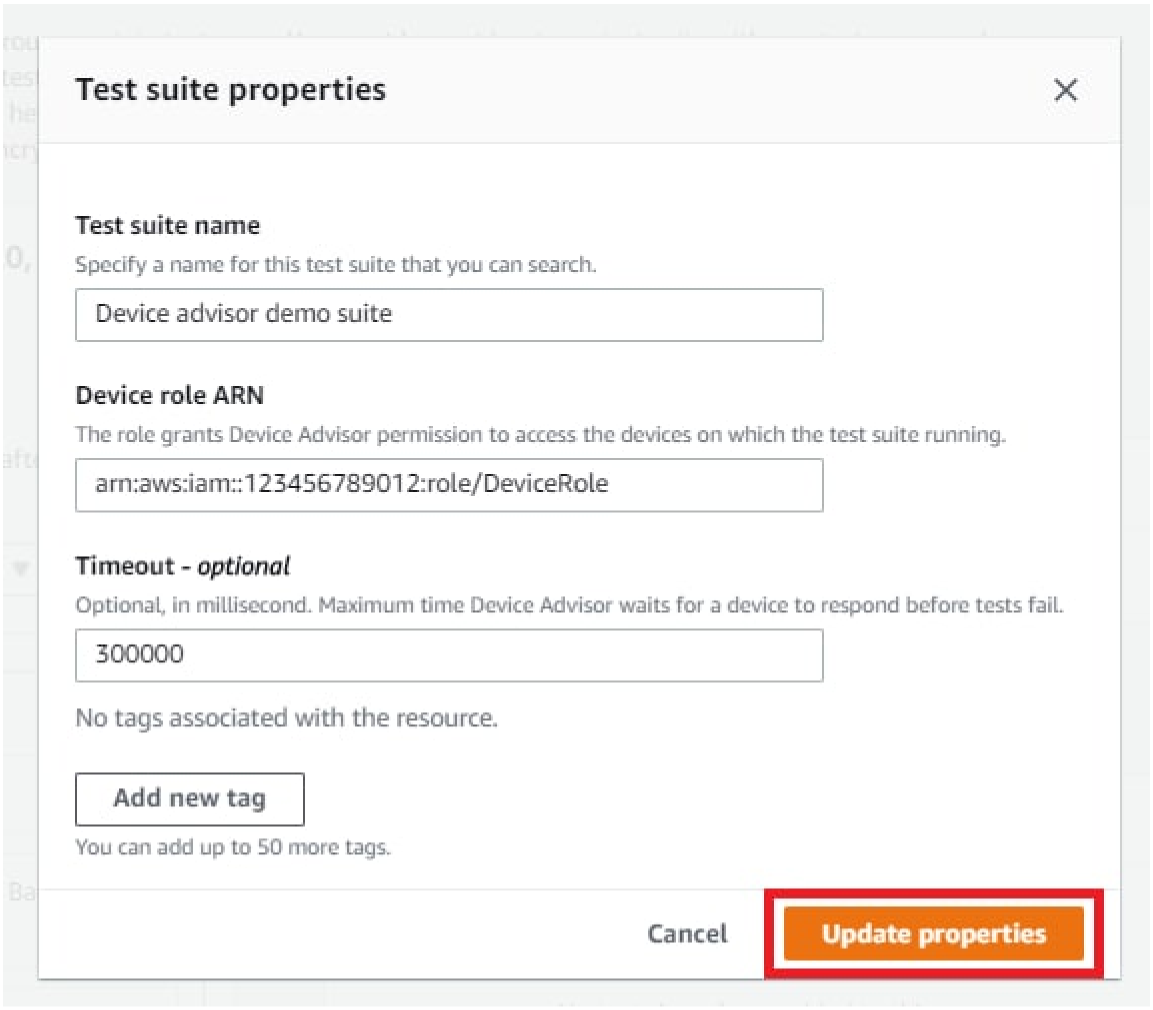
Test suite December 08, 2020, 09:34:19 (UTC-0800)
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Under **Test suite properties**, fill out the following.

- **Test suite name**: You can create the suite with a custom name.

- **Device role ARN**: Provide the device role ARN that was created as part of the prerequisites (p. 898).
- **Timeout** (optional): The timeout in milliseconds for each test case in the current test suite. The default timeout value will be used if no timeout value is specified.
- **Tags** (optional): Add tags to the test suite that you are going to create.



Test suite properties

Test suite name

Specify a name for this test suite that you can search.

Device advisor demo suite

Device role ARN

The role grants Device Advisor permission to access the devices on which the test suite running.

arn:aws:iam::123456789012:role/DeviceRole

Timeout - *optional*

Optional, in millisecond. Maximum time Device Advisor waits for a device to respond before tests fail.

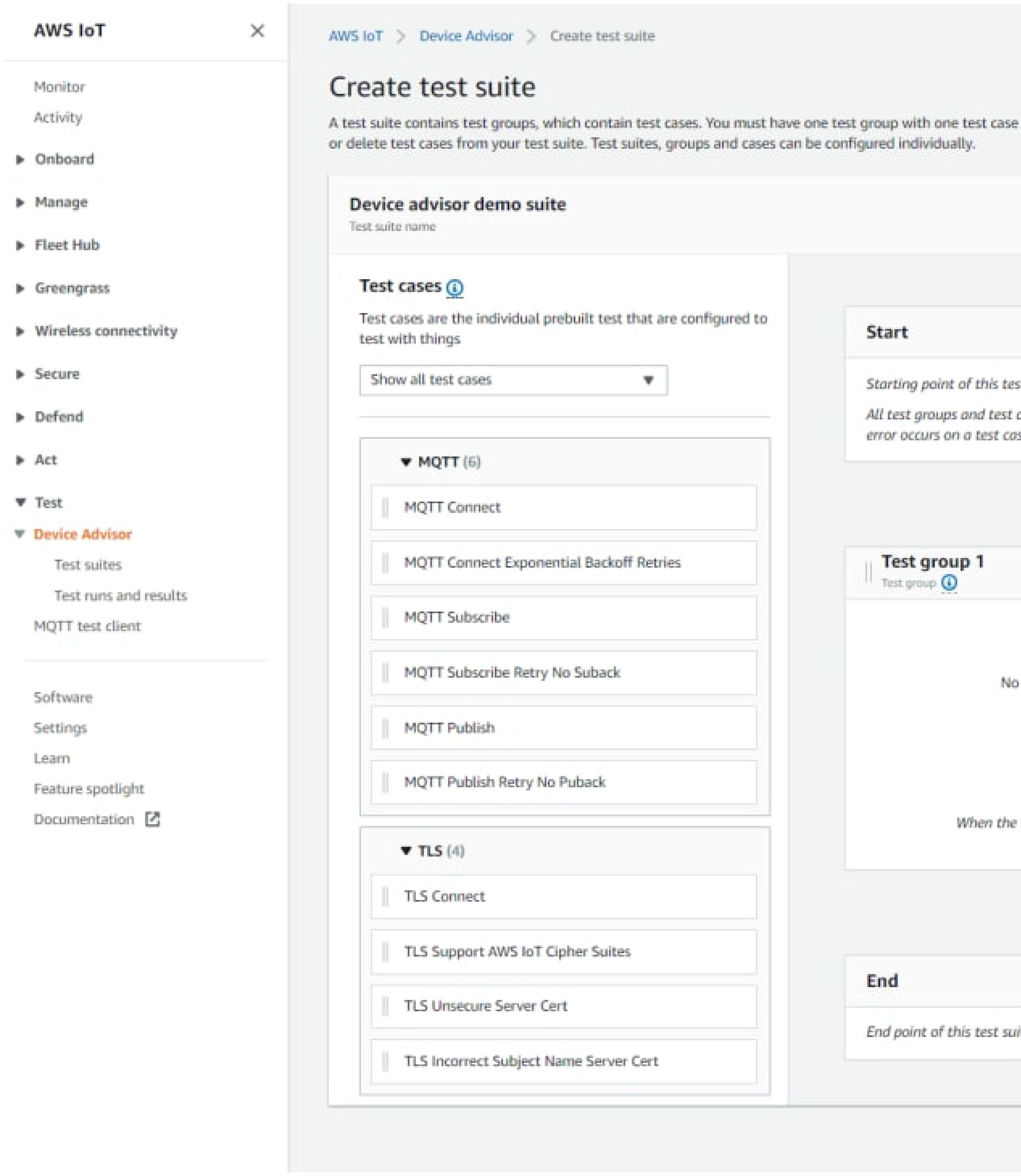
300000

No tags associated with the resource.

You can add up to 50 more tags.

Once done, choose **Update properties**.

4. Choose **Edit** under `Test group 1` to modify the group level configuration. Then, enter a **Name** to give the group a custom name. Optionally, you can also enter a **Timeout** value in milliseconds under the selected test group. The default timeout value will be used if no timeout value is specified.



AWS IoT

Monitor
Activity
▶ Onboard
▶ Manage
▶ Fleet Hub
▶ Greengrass
▶ Wireless connectivity
▶ Secure
▶ Defend
▶ Act
▼ Test
▼ Device Advisor
Test suites
Test runs and results
MQTT test client

Software
Settings
Learn
Feature spotlight
Documentation

AWS IoT > Device Advisor > Create test suite

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case
or delete test cases from your test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

▼ MQTT (6)
- MQTT Connect
- MQTT Connect Exponential Backoff Retries
- MQTT Subscribe
- MQTT Subscribe Retry No Suback
- MQTT Publish
- MQTT Publish Retry No Puback

▼ TLS (4)
- TLS Connect
- TLS Support AWS IoT Cipher Suites
- TLS Unsecure Server Cert
- TLS Incorrect Subject Name Server Cert

Start
Starting point of this tes
All test groups and test c
error occurs on a test cas

Test group 1
Test group

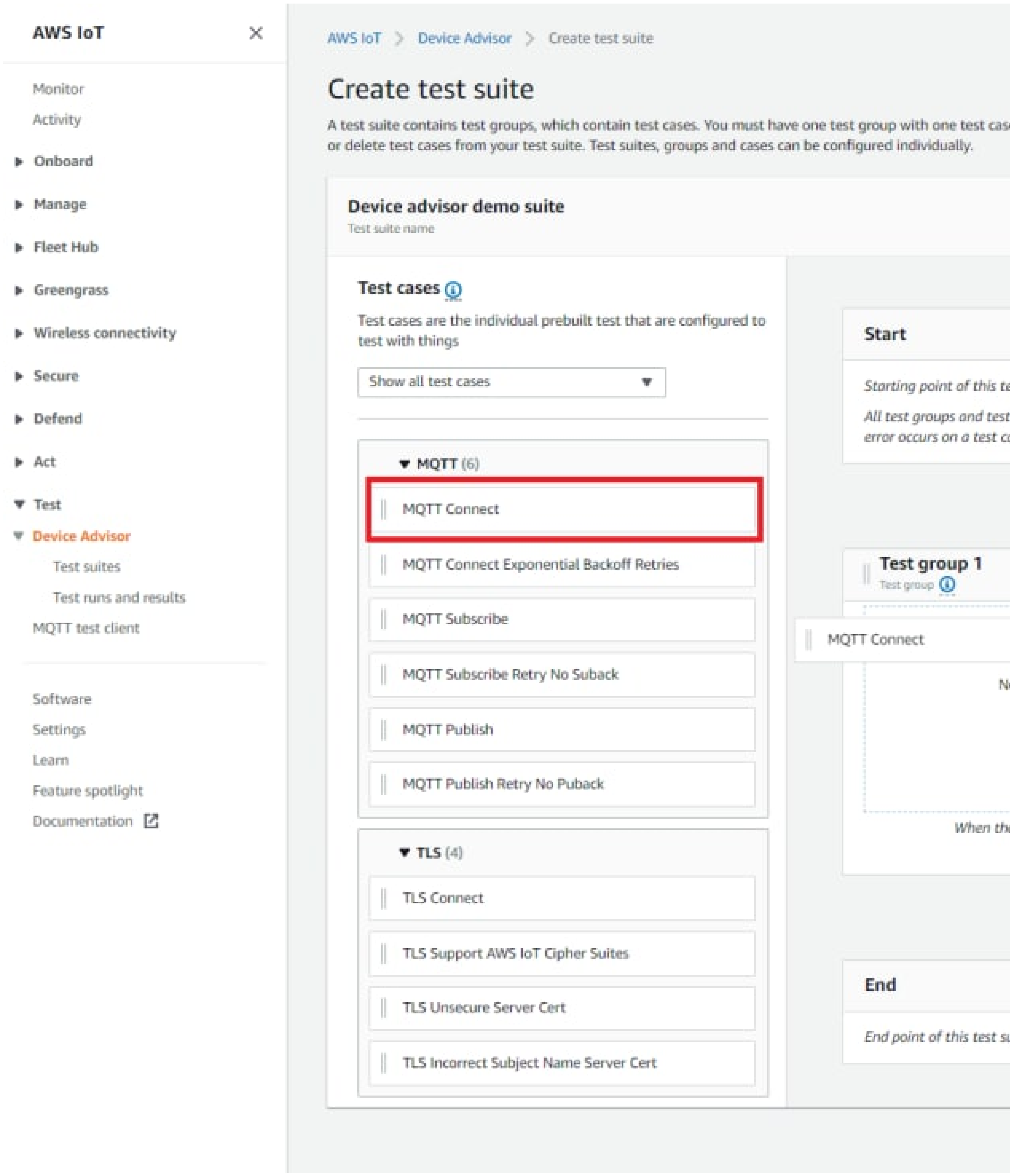
No

When the

End
End point of this test sui

Choose **Done**.

5. Then, drag one of the available test cases from **Test cases** into the test group.



Create test suite

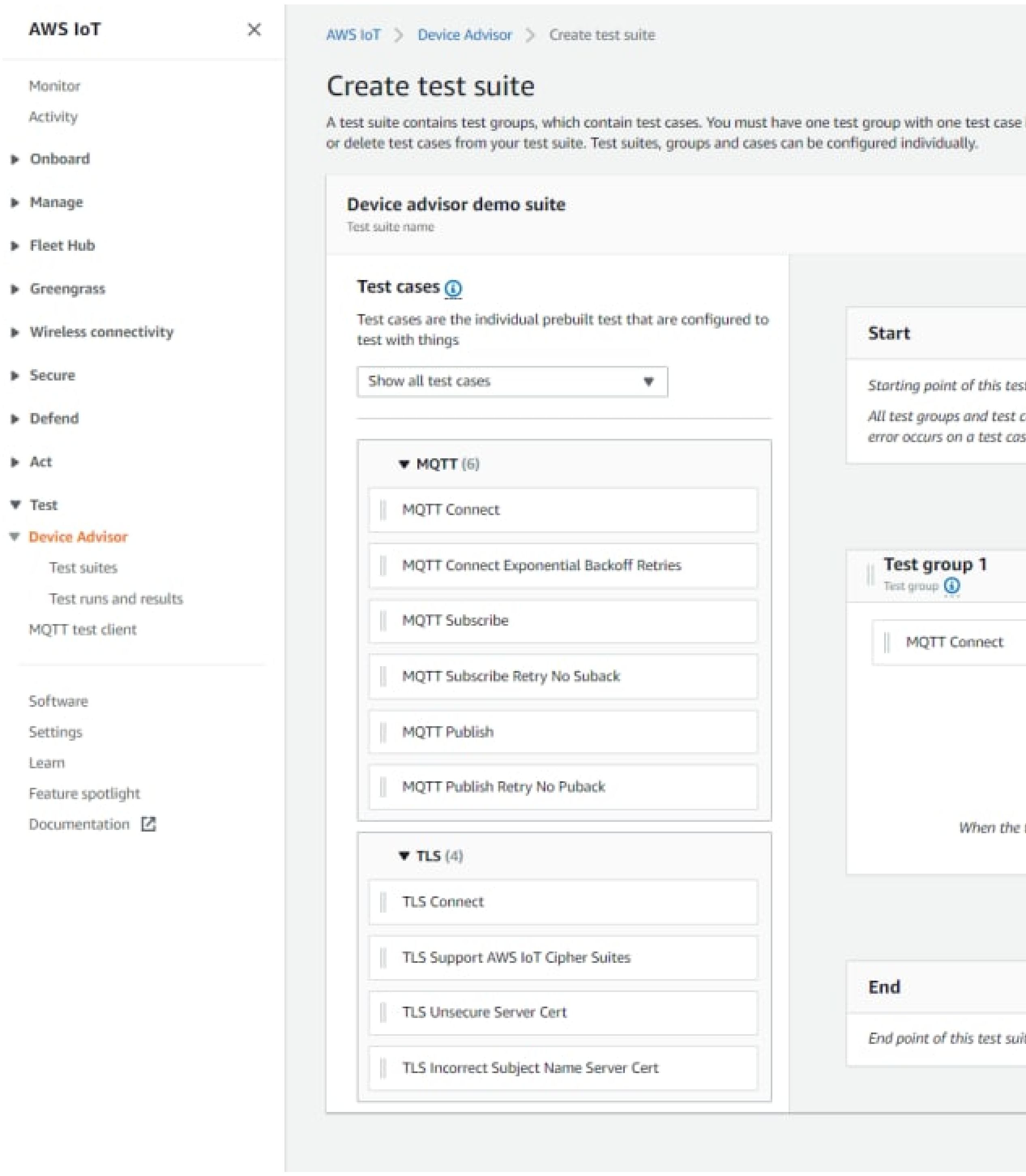
A test suite contains test groups, which contain test cases. You must have one test group with one test cas
or delete test cases from your test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases
Test cases are the individual prebuilt test that are configured to test with things

Show all test cases
MQTT (6)
MQTT Connect
MQTT Connect Exponential Backoff Retries
MQTT Subscribe
MQTT Subscribe Retry No Suback
MQTT Publish
MQTT Publish Retry No Puback
TLS (4)
TLS Connect
TLS Support AWS IoT Cipher Suites
TLS Unsecure Server Cert
TLS Incorrect Subject Name Server Cert
Start
Starting point of this te
All test groups and test
error occurs on a test c
Test group 1
Test group
MQTT Connect
N
When th
End
End point of this test su

6. Choose **Edit** on the test case under your test group to modify the test case level configuration. Then, enter a **Name** to give the group a custom name. Optionally, you can also enter a **Timeout** value in milliseconds under the selected test group. The default timeout value will be used if no timeout value is specified.



AWS IoT

Monitor
Activity
Onboard
Manage
Fleet Hub
Greengrass
Wireless connectivity
Secure
Defend
Act
Test
Device Advisor
Test suites
Test runs and results
MQTT test client

Software
Settings
Learn
Feature spotlight
Documentation

AWS IoT > Device Advisor > Create test suite

Create test suite

A test suite contains test groups, which contain test cases. You must have one test group with one test case
or delete test cases from your test suite. Test suites, groups and cases can be configured individually.

Device advisor demo suite
Test suite name

Test cases

Test cases are the individual prebuilt test that are configured to test with things

Show all test cases

MQTT (6)
- MQTT Connect
- MQTT Connect Exponential Backoff Retries
- MQTT Subscribe
- MQTT Subscribe Retry No Suback
- MQTT Publish
- MQTT Publish Retry No Puback

TLS (4)
- TLS Connect
- TLS Support AWS IoT Cipher Suites
- TLS Unsecure Server Cert
- TLS Incorrect Subject Name Server Cert

Start

Starting point of this tes

All test groups and test c
error occurs on a test cas

Test group 1
Test group

MQTT Connect

When the

End

End point of this test sui

Choose **Done**.

Note
Select **Add test group** to add more test groups to the test suite. Follow the steps above to create and configure more test groups or to add more test cases to one or more test groups. Test groups and test cases can be reordered by dragging. Device Advisor will run tests in the order in which you define the test groups and test cases.

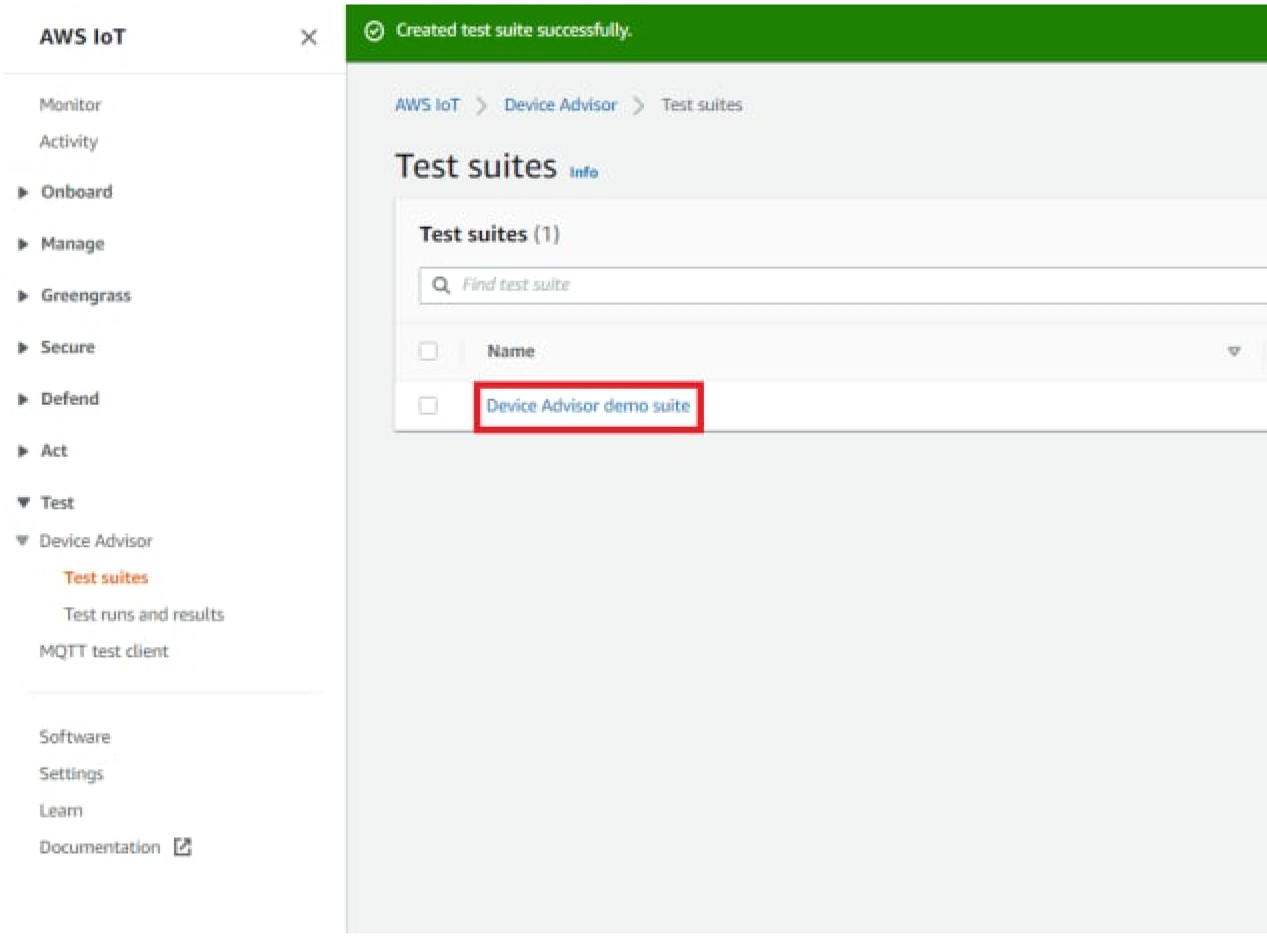
7. Choose **Create test suite**.

   The test suite should be created successfully and you'll be redirected to the **Test suites** page where you can view all the test suite that have been created.

   If the test suite creation failed, make sure the test suite, test groups and test cases have been configured according to the instructions.

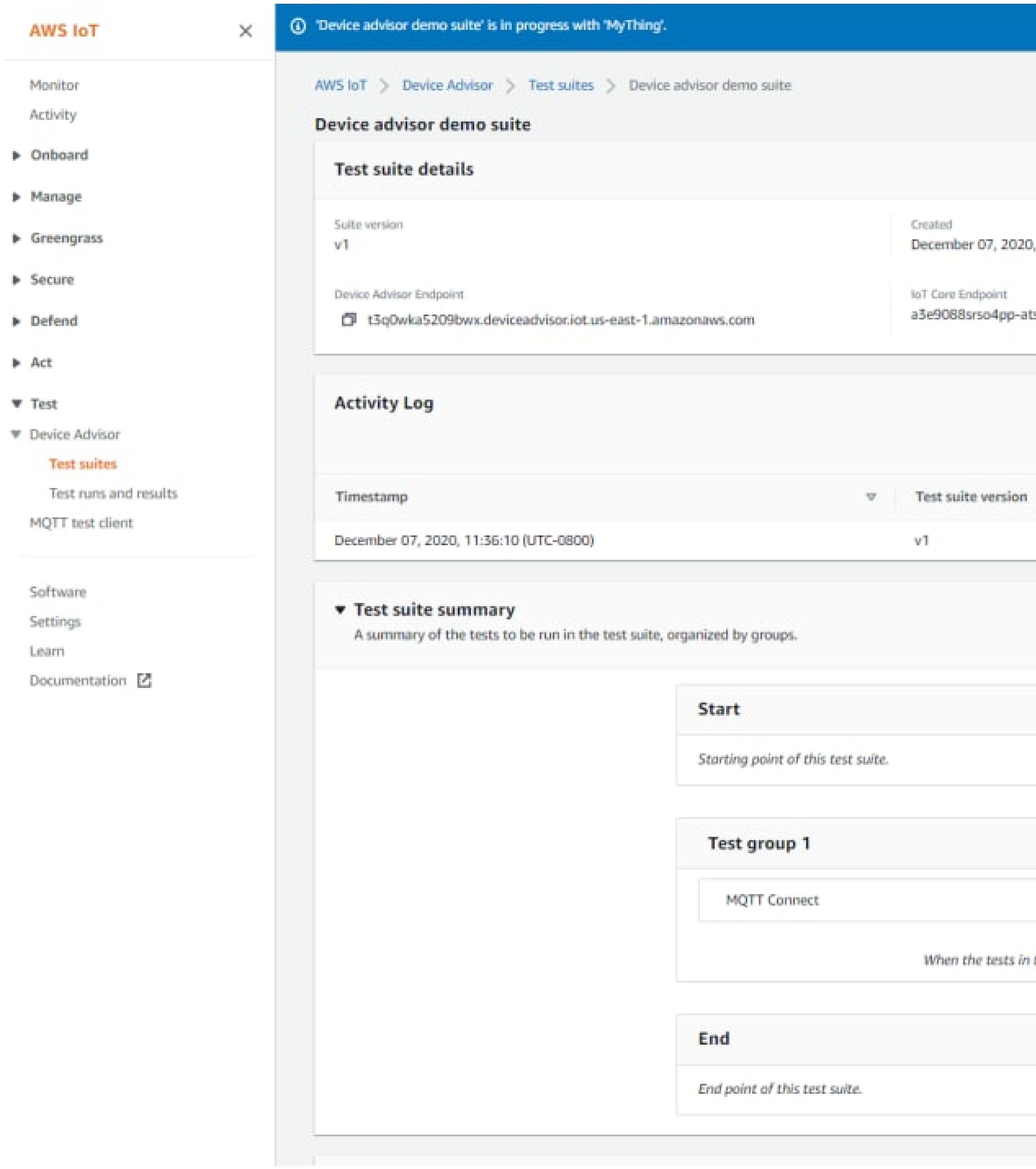
Start a test suite run

1. In the AWS IoT console, in the navigation pane, expand **Test**, **Device Advisor** and then choose **Test suites**.
2. Choose the test suite for which you'd like to view the test suite details.



Created test suite successfully.

   The test suite detail page displays all the information related to the test suite. The **Device Advisor Endpoint** displayed on this page can be used to configure the firmware/software on the device that you'll use for testing to connect to the Device Advisor test endpoint for your account.
3. Choose **Actions**, then **Run test suite**.
4. Under **Run configuration**, you'll need to select an AWS IoT thing or certificate to test using Device Advisor. If you don't have any existing things or certificates, you'll need to first create AWS IoT Core resources (p. 898). Once you select a thing or certificate, choose **Run test**.


'Device advisor demo suite' is in progress with 'MyThing'.

AWS IoT > Device Advisor > Test suites > Device advisor demo suite

Device advisor demo suite

Test suite details

Suite version
v1

Created
December 07, 2020,

Device Advisor Endpoint
t3q0wka5209bwx.deviceadvisor.iot.us-east-1.amazonaws.com

IoT Core Endpoint
a3e9088srso4pp-at

Activity Log

Timestamp	Test suite version
December 07, 2020, 11:36:10 (UTC-0800)	v1

Test suite summary
A summary of the tests to be run in the test suite, organized by groups.

Start
Starting point of this test suite.

Test group 1
MQTT Connect
When the tests in

End
End point of this test suite.

5. Choose **Go to results** on the top banner for viewing the test run details.

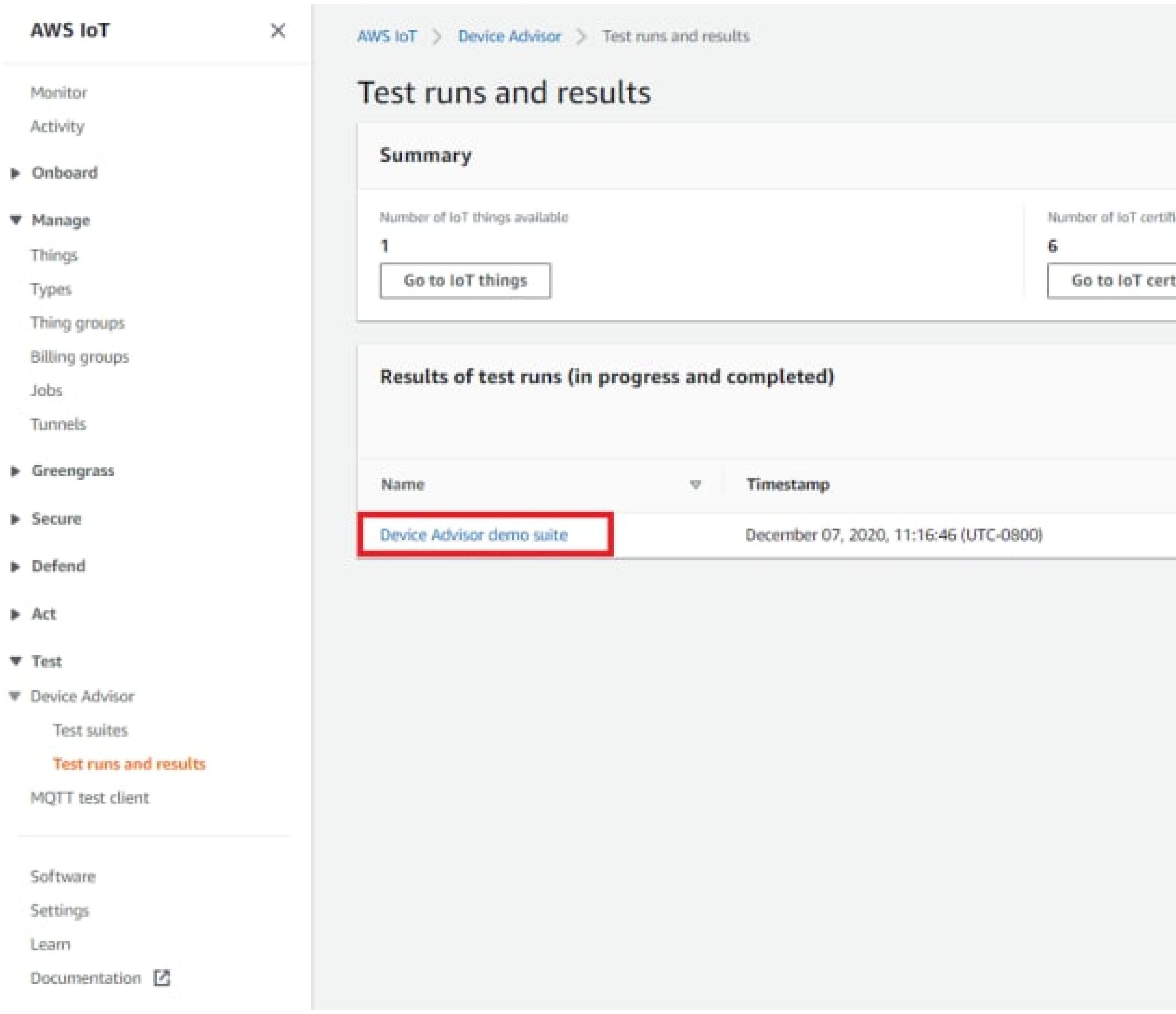
View test suite run details and logs

1. In the AWS IoT console, in the navigation pane, expand **Test**, **Device Advisor** and then choose **Test runs and results**.

This page displays:

- Number of IoT things.
- Number of IoT certificates.
- Number of test suites currently running.
- All the test suite runs that have been created.

2. Choose the test suite for which you'd like to view the run details and logs.



Test runs and results

Summary

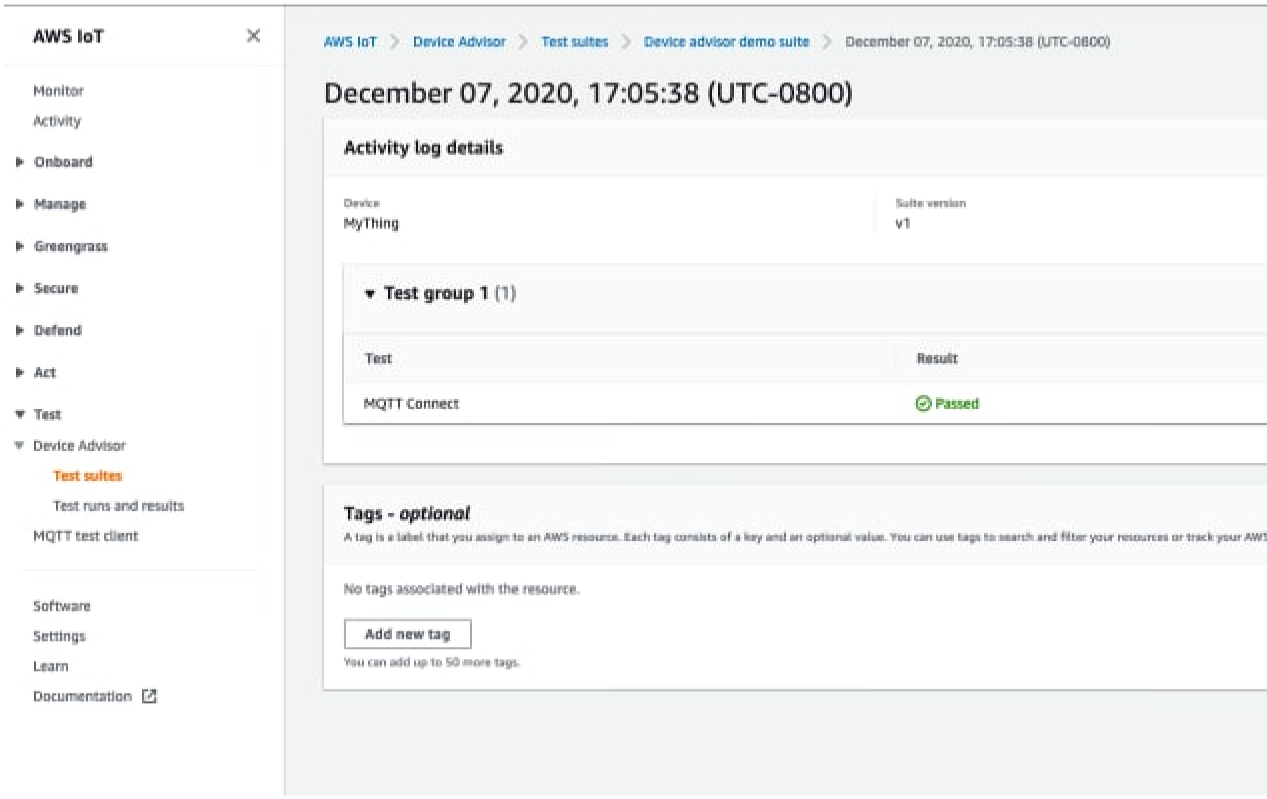
Number of IoT things available: 1

Number of IoT certifi: 6

Results of test runs (in progress and completed)

Name	Timestamp
Device Advisor demo suite	December 07, 2020, 11:16:46 (UTC-0800)

The run summary page displays the status of the current test suite run. This page auto refreshes every 10 seconds. We recommend that you have a mechanism built for your device to try connecting to our test end point every 5 seconds for one to two minutes. This will enable you to run multiple test cases in sequence in an automated manner.


AWS IoT

Monitor
Activity
Onboard
Manage
Greengrass
Secure
Defend
Act
Test
Device Advisor
Test suites
Test runs and results
MQTT test client

Software
Settings
Learn
Documentation

AWS IoT > Device Advisor > Test suites > Device advisor demo suite > December 07, 2020, 17:05:38 (UTC-0800)

December 07, 2020, 17:05:38 (UTC-0800)

Activity log details

Device: MyThing
Suite version: v1

Test group 1 (1)

Test	Result
MQTT Connect	Passed

Tags - *optional*
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS

No tags associated with the resource.

Add new tag
You can add up to 50 more tags.

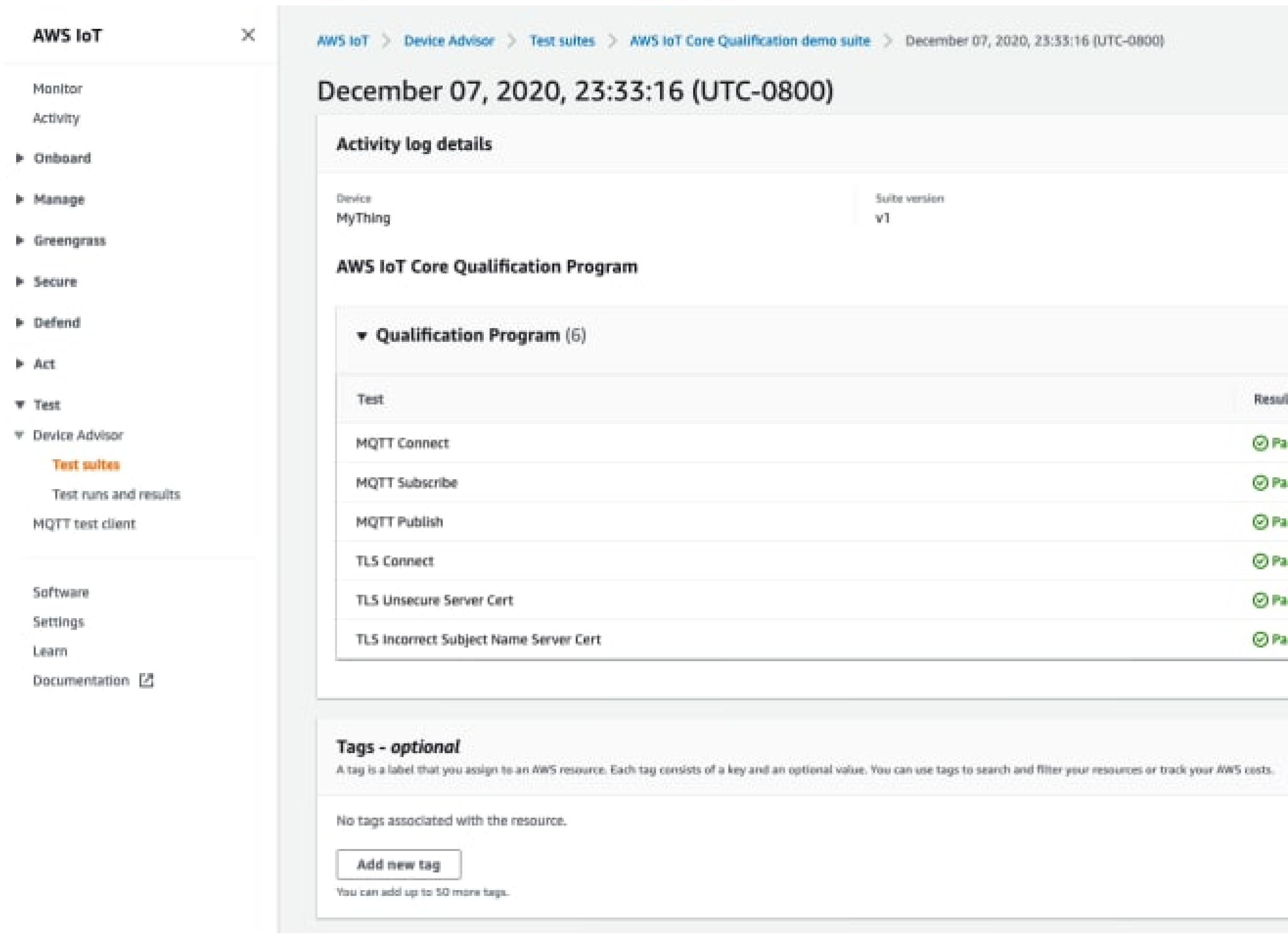
3. To access the CloudWatch logs for the test suite run, choose **Test suite log**.

   To access CloudWatch logs for any test case, choose **Test case log**.

4. Based on your test results, troubleshoot your device until all tests pass.

Download an AWS IoT qualification report

If you chose the **Use the AWS IoT Qualification test suite** option while creating a test suite and were able to successfully run a qualification test suite, you can download a qualification report by choosing **Download qualification report** in the test run summary page.



December 07, 2020, 23:33:16 (UTC-0800)

Activity log details

Device	Suite version
MyThing	v1

AWS IoT Core Qualification Program

Qualification Program (6)

Test	Resul
MQTT Connect	✓ Pa
MQTT Subscribe	✓ Pa
MQTT Publish	✓ Pa
TLS Connect	✓ Pa
TLS Unsecure Server Cert	✓ Pa
TLS Incorrect Subject Name Server Cert	✓ Pa

Tags - *optional*

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.

You can add up to 50 more tags.

Device Advisor test cases

Device Advisor is in preview and is subject to change.

Device Advisor provides prebuilt tests in three categories.

- Reliable connectivity

  Device connects and stays connected with AWS IoT Core.
- Secure connectivity

  Validate that your devices follow recommended security best practices.

Note
More test cases will be released in the future.

TLS

Cipher suites

"TLS Connect"

Validates if the client device can complete TLS handshake to AWS IoT. This test doesn't validate the MQTT implementation of the client device.

API test case definition:

`EXECUTION_TIMEOUT` is in milliseconds.

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_tls_connect_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300000",
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
]
```

Test case outputs:

Pass: The client device completed TLS handshake with AWS IoT.

Pass with warnings: The client device completed TLS handshake with AWS IoT, but there were TLS warning messages from the device or AWS IoT.

Fail: The client device failed to complete TLS handshake with AWS IoT due to handshake error.

"TLS Device Support for AWS IoT recommended Cipher Suites"

Validates that the cipher suites in the TLS Client Hello message from the client device contains AWS IoT recommended cipher suites (p. 275). It provides additional insights into cipher suites supported by the device.

API test case definition:

`EXECUTION_TIMEOUT` (optional) is in milliseconds.

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_tls_support_aws_iot_cipher_suites_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300000",
      },
      "test":{
         "id":"TLS_Support_AWS_IoT_Cipher_Suites",
         "version":"0.0.0"
      }
   }
]
```

Test case outputs:

Pass: The client device cipher suites contain at least one AWS IoT recommended cipher suite and don't contain any unsupported cipher suites.

Pass with warnings: The device cipher suites contain at least one AWS IoT cipher suite but 1) don't contain any of the recommended cipher suites, or 2) contain cipher suites not supported by AWS IoT. We suggest verifying that unsupported cipher suites are safe.

Fail: The client device cipher suites don't contain any of the AWS IoT supported cipher suites.

Bad server certificate

"Not Signed By Recognized CA"

Validates that the client device closes the connection if it's presented with a server certificate that doesn't have a valid signature from the ATS CA. A device should only connect to an endpoint that presents a valid certificate.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_tls_unsecure_server_cert_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300000",
      },
      "test":{
         "id":"TLS_Unsecure_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Test case outputs:

Pass: The client device closed the connection.

Fail: The client device completed TLS handshake with AWS IoT.

"Incorrect Subject Common Name (CN) / Subject Alternative Name (SAN)"

Validates that the client device closes the connection if it's presented with a server certificate for a domain name that is different than the one requested.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_tls_incorrect_subject_name_cert_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300000",
      },
      "test":{
         "id":"TLS_Incorrect_Subject_Name_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Test case outputs:

Pass: The client device closed the connection.

Fail: The client device completed TLS handshake with AWS IoT.

MQTT

CONNECT, DISCONNECT, and RECONNECT

"Device send CONNECT to AWS IoT Core (Happy case)"

Validates that the client device sends a CONNECT request.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_mqtt_connect_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300000",
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

"Device re-connect with exponential backoff - No CONNACK response"

Validates that the client device uses the proper exponential backoff when reconnecting with the server for at least 5 times. The server will log the timestamp of the client device's CONNECT request, perform packet validation, then pause without sending a CONNACK to the client device, and wait for the client device to resend the request. The collected timestamps are used to validate that exponential backoff is used by the client device.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 10 minutes.

```
"tests":[
   {
      "name":"my_mqtt_exponential_backoff_retries_test"
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"600000",
      },
      "test":{
         "id":"MQTT_Connect_Exponential_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

Publish

"QoS0 (Happy Case)"

Validates that the client device publishes a message with QoS0. You can also validate the topic of the message by specifying this topic value in the test settings.

API test case definition:

Note
EXECUTION_TIMEOUT has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_mqtt_publish_test":{
         // optional:
         "EXECUTION_TIMEOUT":"300000",
         "TOPIC_FOR_PUBLISH_VALIDATION": "String",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "String",
      },
      "test":{
         "id":"MQTT_Publish",
         "version":"0.0.0"
      }
   }
]
```

"QoS1 publish retry - No PUBACK"

Validates that the client device republishes a message sent with QoS1, if the server doesn't send PUBACK. You can also validate the topic of the message by specifying this topic in the test settings. The client device must not disconnect before republishing the message. This test also validates that the republished message has the same packet identifier as the original.

API test case definition:

Note
EXECUTION_TIMEOUT has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_mqtt_publish_retry_test":{
         // optional:
         "EXECUTION_TIMEOUT":"300000",
         "TOPIC_FOR_PUBLISH_VALIDATION": "String",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "String",
      },
      "test":{
         "id":"MQTT_Publish_Retry_No_Puback",
         "version":"0.0.0"
      }
   }
]
```

Subscribe

"Can Subscribe (Happy Case)"

Validates that the client device subscribes to MQTT topics. You can also validate the topic that the client device subscribes to by specifying this topic in the test settings.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_mqtt_subscribe_test":{
         // optional:
         "EXECUTION_TIMEOUT":"300000",
         "TOPIC_FOR_PUBLISH_VALIDATION":["String","String"]
      },
      "test":{
         "id":"MQTT_Subscribe",
         "version":"0.0.0"
      }
   }
]
```

"Subscribe Retry - No SUBACK"

Validates that the client device retries a failed subscription to MQTT topics. The server then waits and doesn't send a SUBACK. If the client device doesn't retry the subscription, the test fails. You can also validate the topic that the client device subscribes to by specifying this topic in the test settings.

API test case definition:

Note
`EXECUTION_TIMEOUT` has a default value of 5 minutes.

```
"tests":[
   {
      "name":"my_mqtt_subscribe_retry_test":{
         "EXECUTION_TIMEOUT":"300000",
         // optional:
         "TOPIC_FOR_PUBLISH_VALIDATION":["String","String"]
      },
      "test":{
         "id":"MQTT_Subscribe_Retry_No_Suback",
         "version":"0.0.0"
      }
   }
]
```

Device Advisor commands

Device Advisor is in preview and is subject to change.

This chapter contains the following sections:

CreateSuiteDefinition

Creates a Device Advisor test suite.

Synopsis

```
aws iotdeviceadvisor create-suite-definition \
    [--suite-definition-configuration <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionConfiguration": {
    "suiteDefinitionName": "string",
    "devices": [
      {
        "thingArn": "string",
        "certificateArn": "string"
      }
    ],
    "intendedForQualification": "boolean",
    "rootGroup": "string",
    "devicePermissionRoleArn": "string"
  }
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionConfiguration`	Configuration values that apply to all test groups and cases in the test suite. SuiteDefinitionConfiguration
`suiteDefinitionName`	The name of the test suite. string

Name	Description
	length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`devices`	The devices on which the test suite may be run. You must set up each device to use the Device Advisor test endpoint for your account. list member: DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`intendedForQualification`	True if the test suite is run to qualify the device for the AWS Device Qualification Program. boolean
`rootGroup`	The top-most group of test groups and cases in the test suite. string length- max:2048 min:1 pattern: \S+
`devicePermissionRoleArn`	The ARN of the role that grants permission to Device Advisor to access the devices on which the test suite is run. string length- max:2048 min:20 pattern: arn:*

Output

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionArn": "string",
  "suiteDefinitionName": "string",
  "createdAt": "timestamp"
}
```

CLI output fields

Name	Description
`suiteDefinitionId`	The ID of the test suite that was created. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionArn`	The ARN of the test suite that was created. string length- max:2048 min:20 pattern: arn:*
`suiteDefinitionName`	The name you gave the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`createdAt`	The time the test suite was created. timestamp

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

DeleteSuiteDefinition

Deletes a Device Advisor test suite, including all versions and all resources associated with test suite runs. CloudWatch logs are not deleted.

Synopsis

```
aws iotdeviceadvisor  delete-suite-definition \
    --suite-definition-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string"
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Output

None

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

GetSuiteDefinition

Gets information about a Device Advisor test suite.

Synopsis

```
aws iotdeviceadvisor  get-suite-definition \
    --suite-definition-id <value> \
    [--suite-definition-version <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionVersion": "string"
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string

Name	Description
	length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The version of the test suite. string length- max:255 min:2 pattern: v[0-9]+

Output

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionVersion": "string",
  "suiteDefinitionConfiguration": {
    "suiteDefinitionName": "string",
    "devices": [
      {
        "thingArn": "string",
        "certificateArn": "string"
      }
    ],
    "intendedForQualification": "boolean",
    "rootGroup": "string",
    "devicePermissionRoleArn": "string"
  },
  "createdAt": "timestamp",
  "lastModifiedAt": "timestamp"
}
```

CLI output fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The version of the test suite. string length- max:255 min:2 pattern: v[0-9]+
`suiteDefinitionConfiguration`	Configuration values for all test groups and test cases in the test suite.

Name	Description
	SuiteDefinitionConfiguration
`suiteDefinitionName`	The name of the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`devices`	The devices on which the test suite may be run. You must set up each device to use the Device Advisor test endpoint for your account. list member: DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`intendedForQualification`	True if the test suite is run to qualify the device for the AWS Device Qualification Program. boolean
`rootGroup`	The top-most group of test groups and cases in the test suite. string length- max:2048 min:1 pattern: \S+
`devicePermissionRoleArn`	The ARN of the role that grants permission to Device Advisor to access the devices on which the test suite is run. string length- max:2048 min:20 pattern: arn:*

Name	Description
`createdAt`	The time the test suite was created. timestamp
`lastModifiedAt`	The time the test suite was last modified. timestamp

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

GetSuiteRun

Gets information about a Device Advisor test suite run.

Synopsis

```
aws iotdeviceadvisor  get-suite-run \
    --suite-definition-id <value> \
    --suite-run-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string",
  "suiteRunId": "string"
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteRunId`	The ID of the test suite run. string

Name	Description
	length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Output

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionVersion": "string",
  "suiteRunId": "string",
  "suiteRunConfiguration": {
    "primaryDevice": {
      "thingArn": "string",
      "certificateArn": "string"
    },
    "secondaryDevice": {
      "thingArn": "string",
      "certificateArn": "string"
    },
    "selectedTestList": [
      "string"
    ]
  },
  "testCaseRuns": [
    {
      "testCaseDefinitionId": "string",
      "testCaseDefinitionName": "string",
      "status": "string",
      "startTime": "timestamp",
      "endTime": "timestamp",
      "logUrl": "string",
      "warnings": "string",
      "failure": "string"
    }
  ],
  "startTime": "timestamp",
  "endTime": "timestamp",
  "status": "string",
  "errorReason": "string",
}
```

CLI output fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The version of the test suite. string length- max:255 min:2

Name	Description
	pattern: v[0-9]+
`suiteRunId`	The ID of the test suite run. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteRunConfiguration`	The configuration values applied to all test groups and test cases in this test suite during this run. SuiteRunConfiguration
`primaryDevice`	The device on which the tests in the test suite are run. At this time, a Device Advisor test suite targets primarily one device, which is specified by this parameter. Some tests do compare the behavior of two devices, so if you include such a test, you must also specify a `secondaryDevice`. DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`secondaryDevice`	If a test compares the behavior of two devices (for example, "Device re-connect with Jitter (multiple device with deterministic jitter)"), this parameter specifies the second device used for the test. DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*

Name	Description
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`selectedTestList`	[optional] The ID of the specific test group or test case that will be run. list member: UUID
`testCaseRuns`	The results of each test case run. list member: TestCaseRun
`testCaseDefinitionId`	The ID of the test case. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`testCaseDefinitionName`	The name of the test case. string
`status`	The status of the test case run. string
`startTime`	The time the test case run started. timestamp
`endTime`	The time the test case run ended. timestamp
`logUrl`	The URL of the CloudWatch logs that contain the test case run results. string
`warnings`	Any warnings generated by the test case run. string
`failure`	Any failures generated by the test case run. string

Name	Description
`startTime`	The time this test suite run started. timestamp
`endTime`	The time this test suite run ended. timestamp
`status`	The status of this test suite run. string
`errorReason`	The reason an error (if any) occurred during this test suite run. string

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

GetSuiteRunReport

Gets suite run report.

Synopsis

```
aws iotdeviceadvisor  get-suite-run-report \
    --suite-definition-id <value> \
    --suite-run-id <value>  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string",
  "suiteRunId": "string"
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string

Name	Description
	length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteRunId`	The ID of the test suite run. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Output

```
{
    "qualificationReportDownloadUrl": "string"
}
```

CLI output fields

Name	Description
`qualificationReportDownloadUrl`	Gets the download URL of the qualification report. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Errors

`ValidationException`

Invalid request exception.

`InternalServerException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

ListSuiteDefinitions

Lists the Device Advisor test suites you have created.

Synopsis

```
aws iotdeviceadvisor  list-suite-definitions \
    [--max-results <value>] \
    [--next-token <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "maxResults": "integer",
  "nextToken": "string"
}
```

`cli-input-json` fields

Name	Description
`maxResults`	The maximum number of results to return at one time. integer range- max:50 min:1
`nextToken`	The token for the next set of results. string length- max:2000 pattern: \S+

Output

```
{
  "suiteDefinitionInformationList": [
    {
      "suiteDefinitionId": "string",
      "suiteDefinitionName": "string",
      "defaultDevicesUnderTest": [
        {
          "thingArn": "string",
          "certificateArn": "string"
        }
      ],
      "intendedForQualification": "boolean",
      "createdAt": "timestamp"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Description
`suiteDefinitionInformationList`	Information about the test suites. list member: SuiteDefinitionInformation

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionName`	The name of the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`defaultDevicesUnderTest`	The devices on which the test suite is run. list member: DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`intendedForQualification`	True if the test suite is intended to qualify the devices tested for the AWS Device Partner Program. boolean
`createdAt`	The time the test suite was created. timestamp

Name	Description
`nextToken`	A token that can be used to retrieve the next set of results, or `null` if there are no additional results. string length- max:2000 pattern: \S+

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

ListSuiteRuns

Lists the runs of the specified Device Advisor test suite.

You can list all runs of the test suite, or the runs of a specific test suite, or all runs of a specific test suite with a specific version.

Synopsis

```
aws iotdeviceadvisor  list-suite-runs \
    [--suite-definition-id <value>] \
    [--suite-definition-version <value>] \
    [--max-results <value>] \
    [--next-token <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionVersion": "string",
  "maxResults": "integer",
  "nextToken": "string"
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite for which runs will be listed. string

Name	Description
	length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The version of the test suite for which runs will be listed. string length- max:255 min:2 pattern: v[0-9]+
`maxResults`	The maximum number of results to return at one time. integer range- max:50 min:1
`nextToken`	The token for the next set of results. string length- max:2000 pattern: \S+

Output

```
{
  "suiteRunsList": [
    {
      "suiteDefinitionId": "string",
      "suiteDefinitionVersion": "string",
      "suiteDefinitionName": "string",
      "suiteRunId": "string",
      "createdAt": "timestamp",
      "startedAt": "timestamp",
      "endAt": "timestamp",
      "status": "string",
      "passed": "integer",
      "failed": "integer"
    }
  ],
  "nextToken": "string"
}
```

CLI output fields

Name	Description
`suiteRunsList`	Information about the test suite runs. list member: SuiteRunInformation

Name	Description
`suiteDefinitionId`	The test suite ID. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The test suite version. string length- max:255 min:2 pattern: v[0-9]+
`suiteDefinitionName`	The name you gave the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`suiteRunId`	The ID of the test suite run. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`createdAt`	The time the test suite run was created. timestamp
`startedAt`	The time the test suite run started. timestamp
`endAt`	The time the test suite run ended. timestamp
`status`	The status of the test suite run. string
`passed`	The number of test cases that passed. integer range- max:500 min:0

Name	Description
`failed`	The number of test cases that failed. integer range- max:500 min:0
`nextToken`	A token that can be used to retrieve the next set of results, or `null` if there are no additional results. string length- max:2000 pattern: \S+

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

ListTagsForResource

Lists all tags for a given Device Advisor resource.

Synopsis

```
aws iotdeviceadvisor  list-tags-for-resource \
    [--resource-arn <value>] \
```

`cli-input-json` format

```
{
  "resourceArn": "string",
}
```

`cli-input-json` fields

Name	Description
`resourceArn`	The ARN of the IoT Device Advisor resource. string length- max:2048 min:20 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Output

```
{
    "tags": "string"
}
```

CLI output fields

Name	Description
`tags`	The tags for a given resource. string

Errors

`ValidationException`

Invalid request exception.

`InternalServerException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

ListTestCases

Lists all the available test cases that you can use in a test suite.

Synopsis

```
aws iotdeviceadvisor  list-test-cases \
    [--intended-for-qualification <value>] \
    [--max-results <value>] \
    [--next-token <value>] \
```

`cli-input-json` format

```
{
  "intendedForQualification": "string",
  "maxResults": integer,
  "nextToken": "string"
}
```

`cli-input-json` fields

Name	Description
`intendedForQualification`	Lists all the qualification test cases in the test suite. string length- max:2048 min:20 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Name	Description
`maxResults`	The maximum number of results to return at one time. integer range- max:50 min:1
`nextToken`	A token that can be used to retrieve the next set of results, or `null` if there are no additional results. string length- max:2000 pattern: \S+

Output

```
{
   "categories": [
      {
         "name": "string",
         "tests": [
            {
               "configuration": {
                  "string" : "string"
               },
               "name": "string",
               "test": {
                  "id": "string",
                  "testCaseVersion": "string"
               }
            }
         ]
      }
   ],
   "groupConfiguration": {
      "string" : "string"
   },
   "nextToken": "string",
   "rootGroupConfiguration": {
      "string" : "string"
   }
}
```

CLI output fields

Name	Description
`categories`	This group of test cases belong to this category. array
`groupConfiguration`	Available configurations for test groups and cases in the the test suite.

Name	Description
`nextToken`	A token that can be used to retrieve the next set of results, or `null` if there are no additional results. string length- max:2000 pattern: \S+
`rootGroupConfiguration`	Available configurations for the top-most group of test groups and cases in the the test suite.

Errors

`InternalServerException`

An internal failure occurred.

StartSuiteRun

Starts a Device Advisor test suite run.

Synopsis

```
aws iotdeviceadvisor  start-suite-run \
    --suite-definition-id <value> \
    [--suite-definition-version <value>] \
    [--suite-run-configuration <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
  "suiteDefinitionId": "string",
  "suiteDefinitionVersion": "string",
  "suiteRunConfiguration": {
    "primaryDevice": {
      "thingArn": "string",
      "certificateArn": "string"
    },
    "secondaryDevice": {
      "thingArn": "string",
      "certificateArn": "string"
    },
    "selectedTestList": [
      "string"
    ]
  }
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionId`	The ID of the test suite. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionVersion`	The version of the test suite. (Versions are incremented whenever a test suite is updated.) string length- max:255 min:2 pattern: v[0-9]+
`suiteRunConfiguration`	Configuration values for all test groups and test cases in the test suite. SuiteRunConfiguration
`primaryDevice`	The device on which the tests in the test suite are run. [At this time, a Device Advisor test suite targets primarily one device, which is specified by this parameter. Some tests do compare the behavior of two devices, so if you include such a test, you must also specify a `secondaryDevice`.] DeviceUnderTest
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`secondaryDevice`	If a test compares the behavior of two devices (for example, "Device re-connect with Jitter (multiple device w/ deterministic jitter)"), this parameter specifies the second device used for the test. DeviceUnderTest

Name	Description
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`selectedTestList`	[optional] The ID of the specific test group or test case that will be run. list member: UUID

Output

```
{
  "suiteRunId": "string",
  "suiteRunArn": "string",
  "createdAt": "timestamp"
}
```

CLI output fields

Name	Description
`suiteRunId`	The ID of the test suite run. string length- max:36 min:36 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteRunArn`	The ARN of the test suite run. string length- max:2048 min:20 pattern: arn:*
`createdAt`	The time and date when the test suite was created.

Name	Description
	timestamp

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

`ActiveRunningException`

There is an active suite running.

TagResource

Adds to and modifies existing tags of an IoT Device Advisor resource.

Synopsis

```
aws iotdeviceadvisor  tag-resource \
    --resource-arn <value> \
```

`cli-input-json` format

```
{
  "resourceArn": "string",
}
```

`cli-input-json` fields

Name	Description
`resourceArn`	The ARN of the IoT Device Advisor resource. string length- max:2048 min:20 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}

Output

None.

CLI output fields

Name	Description
`tags`	The tags for a given resource. string

Errors

`ValidationException`

Invalid request exception.

`InternalServerException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

UntagResource

Removes tags from an IoT Device Advisor resource.

Synopsis

```
aws iotdeviceadvisor  untag-resource \
    --resource-arn <value> \
    --tag-keys <value> \
```

`cli-input-json` format

```
{
  "resourceArn": "string",
  "tagKeys": integer,
}
```

`cli-input-json` fields

Name	Description
`resourceArn`	The ARN of the IoT Device Advisor resource. string length- max:2048 min:20 pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`tagKeys`	List of tag keys to remove from the IoT Device Advisor resource. array

Output

None.

Errors

`ValidationException`

Invalid request exception.

`InternalServerException`

An internal failure occurred.

`ResourceNotFoundException`

A resource could not be found.

UpdateSuiteDefinition

Updates a Device Advisor test suite.

Synopsis

```
aws iotdeviceadvisor update-suite-definition \
    [--suite-definition-configuration <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format

```
{
    "suiteDefinitionConfiguration": {
        "suiteDefinitionName": "string",
        "devices": [
            {
                "thingArn": "string",
                "certificateArn": "string"
            }
        ],
        "rootGroup": "string",
        "devicePermissionRoleArn": "string"
    }
}
```

`cli-input-json` fields

Name	Description
`suiteDefinitionConfiguration`	Configuration values that apply to all test groups and cases in the test suite. SuiteDefinitionConfiguration
`suiteDefinitionName`	The name of the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`devices`	The devices on which the test suite can be run. You must set up each device to use the Device Advisor test endpoint for your account. list member: DeviceUnderTest

Name	Description
`thingArn`	The ARN of the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`certificateArn`	The ARN of the certificate associated with the AWS IoT thing (device) on which the test is run. string length- max:2048 min:20 pattern: arn:*
`rootGroup`	The top-most group of test groups and cases in the test suite. string length- max:2048 min:1 pattern: \S+
`devicePermissionRoleArn`	The ARN of the role that grants permission to Device Advisor to access the devices on which the test suite is run. string length- max:2048 min:20 pattern: arn:*

Output

```
{
   "suiteDefinitionId": "string",
   "suiteDefinitionArn": "string",
   "suiteDefinitionName": "string",
   "suiteDefinitionVersion": "string",
   "createdAt": "string",
   "lastUpdatedAt": "string"
}
```

CLI output fields

Name	Description
`suiteDefinitionId`	The ID of the test suite that was created. string length- max:36 min:36

Name	Description
	pattern: [a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}
`suiteDefinitionArn`	The ARN of the test suite that was created. string length- max:2048 min:20 pattern: arn:*
`suiteDefinitionName`	The name you gave the test suite. string length- max:256 min:1 pattern: [a-zA-Z0-9:_-]+
`createdAt`	The time the test suite was created. timestamp
`lastUpdatedAt`	The time the test suite was last updated. timestamp

Errors

`InvalidRequestException`

The request was invalid.

`InternalFailureException`

An internal failure occurred.

Event messages

This section contains information about messages published by AWS IoT when things or jobs are updated or changed. For information about the AWS IoT Events service that allows you to create detectors to monitor your devices for failures or changes in operation, and to trigger actions when they occur, see AWS IoT Events.

AWS IoT publishes event messages when certain events occur. For example, events are generated by the registry when things are added, updated, or deleted. Each event causes a single event message to be sent. Event messages are published over MQTT with a JSON payload. The content of the payload depends on the type of event.

> **Note**
> Event messages are guaranteed to be published once. It is possible for them to be published more than once. The ordering of event messages is not guaranteed.

To receive event messages, your device must use an appropriate policy that allows it to connect to the AWS IoT device gateway and subscribe to MQTT event topics. You must also subscribe to the appropriate topic filters.

The following is an example of the policy required for receiving lifecycle events:

```
{
    "Version":"2012-10-17",
    "Statement":[{
        "Effect":"Allow",
        "Action":[
            "iot:Subscribe",
            "iot:Receive"
        ],
        "Resource":[
            "arn:aws:iot:region:account:/$aws/events/*"
        ]
    }]
}
```

You control which event types are published by calling the UpdateEventConfigurations API or by using the **update-event-configurations** CLI command. For example:

aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\": true}}"

> **Note**
> All quotation marks (") are escaped with a backslash (\).

You can get the current event configuration by calling the DescribeEventConfigurations API or by using the **describe-event-configurations** CLI command. For example:.

aws iot describe-event-configurations

The output of the **describe-event-configurations** command looks like the following:

```
{
    "lastModifiedDate": 1552671347.841,
    "eventConfigurations": {
        "THING_TYPE": {
            "Enabled": false
        },
        "JOB_EXECUTION": {
            "Enabled": false
        },
        "THING_GROUP_HIERARCHY": {
            "Enabled": false
        },
        "THING_TYPE_ASSOCIATION": {
            "Enabled": false
        },
        "THING_GROUP_MEMBERSHIP": {
            "Enabled": false
        },
        "THING": {
            "Enabled": true
        },
        "JOB": {
            "Enabled": false
        },
        "THING_GROUP": {
            "Enabled": false
        }
    },
    "creationDate": 1552671347.84
}
```

Registry events

The registry publishes event messages when things, thing types, and thing groups are created, updated, or deleted. The registry currently supports the following event types:

Thing Created/Updated/Deleted

The registry publishes the following event messages when things are created, updated, or deleted:

- `$aws/events/thing/`*`thingName`*`/created`
- `$aws/events/thing/`*`thingName`*`/updated`
- `$aws/events/thing/`*`thingName`*`/deleted`

The messages contain the following example payload:

```
{
    "eventType" : "THING_EVENT",
    "eventId" : "f5ae9b94-8b8e-4d8e-8c8f-b3266dd89853",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName" : "MyThing",
    "versionNumber" : 1,
    "thingTypeName" : null,
    "attributes": {
                "attribute3": "value3",
                "attribute1": "value1",
                "attribute2": "value2"
    }
}
```

The payloads contain the following attributes:

eventType

Set to "THING_EVENT".

eventId

A unique event ID (string).

timestamp

The UNIX timestamp of when the event occurred.

operation

The operation that triggered the event. Valid values are:

- CREATED
- UPDATED
- DELETED

accountId

Your AWS account ID.

thingId

The ID of the thing being created, updated, or deleted.

thingName

The name of the thing being created, updated, or deleted.

versionNumber

The version of the thing being created, updated, or deleted. This value is set to 1 when a thing is created. It is incremented by 1 each time the thing is updated.

thingTypeName

The thing type associated with the thing, if one exists. Otherwise, `null`.

attributes

A collection of name-value pairs associated with the thing.

Thing Type Created/Deprecated/Undeprecated/Deleted

The registry publishes the following event messages when thing types are created, deprecated, undeprecated, or deleted:

- `$aws/events/thingType/`*`thingTypeName`*`/created`
- `$aws/events/thingType/`*`thingTypeName`*`/updated`
- `$aws/events/thingType/`*`thingTypeName`*`/deleted`

The message contains the following example payload:

```
{
    "eventType" : "THING_TYPE_EVENT",
    "eventId" : "8827376c-4b05-49a3-9b3b-733729df7ed5",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingTypeId" : "c530ae83-32aa-4592-94d3-da29879d1aac",
    "thingTypeName" : "MyThingType",
    "isDeprecated" : false|true,
    "deprecationDate" : null,
    "searchableAttributes" : [ "attribute1", "attribute2", "attribute3" ],
    "description" : "My thing type"
}
```

The payloads contain the following attributes:

eventType

Set to "THING_TYPE_EVENT".

eventId

A unique event ID (string).

timestamp

The UNIX timestamp of when the event occurred.

operation

The operation that triggered the event. Valid values are:

- CREATED
- UPDATED
- DELETED

accountId

Your AWS account ID.

thingTypeId

The ID of the thing type being created, deprecated, or deleted.

thingTypeName

The name of the thing type being created, deprecated, or deleted.

isDeprecated

`true` if the thing type is deprecated. Otherwise, `false`.

deprecationDate

The UNIX timestamp for when the thing type was deprecated.

searchableAttributes

A collection of name-value pairs associated with the thing type that can be used for searching.

description

A description of the thing type.

Thing Type Associated or Disassociated with a Thing

The registry publishes the following event messages when a thing type is associated or disassociated with a thing.

- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/`*`typeName`*

The messages contain the following example payload:

```
{
    "eventId" : "87f8e095-531c-47b3-aab5-5171364d138d",
    "eventType" : "THING_TYPE_ASSOCIATION_EVENT",
    "operation" : "CREATED|DELETED",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName": "myThing",
    "thingTypeName" : "MyThingType",
    "timestamp" : 1234567890123,
}
```

The payloads contain the following attributes:

eventId

A unique event ID (string).

eventType

Set to "THING_TYPE_ASSOCIATION_EVENT".

operation

The operation that triggered the event. Valid values are:

- CREATED
- DELETED

thingId

The ID of the thing whose type association was changed.

thingName

The name of the thing whose type association was changed.

thingTypeName

The thing type associated with, or no longer associated with, the thing.

timestamp

The UNIX timestamp of when the event occurred.

Thing Group Created/Updated/Deleted

The registry publishes the following event messages when a thing group is created, updated, or deleted.

- `$aws/events/thingGroup/`*`groupName`*`/created`
- `$aws/events/thingGroup/`*`groupName`*`/updated`
- `$aws/events/thingGroup/`*`groupName`*`/deleted`

The following is an example of an `updated` payload. Payloads for `created` and `deleted` messages are similar.

```
{
  "eventType": "THING_GROUP_EVENT",
  "eventId": "8b9ea8626aeaa1e42100f3f32b975899",
  "timestamp": 1603995417409,
  "operation": "UPDATED",
  "accountId": "571EXAMPLE833",
  "thingGroupId": "8757eec8-bb37-4cca-a6fa-403b003d139f",
  "thingGroupName": "Tg_level5",
  "versionNumber": 3,
  "parentGroupName": "Tg_level4",
  "parentGroupId": "5fce366a-7875-4c0e-870b-79d8d1dce119",
  "description": "New description for Tg_level5",
  "rootToParentThingGroups": [
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/TgTopLevel",
      "groupId": "36aa0482-f80d-4e13-9bff-1c0a75c055f6"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level1",
      "groupId": "bc1643e1-5a85-4eac-b45a-92509cbe2a77"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level2",
      "groupId": "0476f3d2-9beb-48bb-ae2c-ea8bd6458158"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level3",
      "groupId": "1d9d4ffe-a6b0-48d6-9de6-2e54d1eae78f"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level4",
      "groupId": "5fce366a-7875-4c0e-870b-79d8d1dce119"
    }
  ],
  "attributes": {
    "attribute1": "value1",
    "attribute3": "value3",
    "attribute2": "value2"
  },
  "dynamicGroupMappingId": null
}
```

The payloads contain the following attributes:

eventType

  Set to "THING_GROUP_EVENT".

eventId

  A unique event ID (string).

timestamp

  The UNIX timestamp of when the event occurred.

operation

  The operation that triggered the event. Valid values are:

  - CREATED
  - UPDATED
  - DELETED

accountId

  Your AWS account ID.

thingGroupId

  The ID of the thing group being created, updated, or deleted.

thingGroupName

  The name of the thing group being created, updated, or deleted.

versionNumber

  The version of the thing group. This value is set to 1 when a thing group is created. It is incremented by 1 each time the thing group is updated.

parentGroupName

  The name of the parent thing group, if one exists.

parentGroupId

The ID of the parent thing group, if one exists.

description

A description of the thing group.

rootToParentThingGroups

An array of information about the parent thing group. There is one element for each parent thing group, starting from the root thing group and continuing to the thing group's parent. Each entry contains the thing group's `groupArn` and `groupId`.

attributes

A collection of name-value pairs associated with the thing group.

Thing Added to or Removed from a Thing Group

The registry publishes the following event messages when a thing is added to or removed from a thing group.

- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/added`
- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/removed`

The messages contain the following example payload:

```
{
    "eventType" : "THING_GROUP_MEMBERSHIP_EVENT",
    "eventId" : "d684bd5f-6f6e-48e1-950c-766ac7f02fd1",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "groupArn" : "arn:aws:iot:ap-northeast-2:123456789012:thinggroup/
MyChildThingGroup",
    "groupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "thingArn" : "arn:aws:iot:ap-northeast-2:123456789012:thing/MyThing",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "membershipId" : "8505ebf8-4d32-4286-80e9-c23a4a16bbd8"
}
```

The payloads contain the following attributes:

eventType

Set to "THING_GROUP_MEMBERSHIP_EVENT".

eventId

The event ID.

timestamp

The UNIX timestamp for when the event occurred.

operation

`ADDED` when a thing is added to a thing group. `REMOVED` when a thing is removed from a thing group.

accountId

Your AWS account ID.

groupArn

The ARN of the thing group.

groupId

The ID of the group.

thingArn

The ARN of the thing that was added or removed from the thing group.

thingId

The ID of the thing that was added or removed from the thing group.

membershipId

An ID that represents the relationship between the thing and the thing group. This value is generated when you add a thing to a thing group.

Thing Group Added to or Deleted from a Thing Group

The registry publishes the following event messages when a thing group is added to or removed from another thing group.

- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/added`
- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/removed`

The message contains the following example payload:

```
{
    "eventType" : "THING_GROUP_HIERARCHY_EVENT",
    "eventId" : "264192c7-b573-46ef-ab7b-489fcd47da41",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "thingGroupId" : "8f82a106-6b1d-4331-8984-a84db5f6f8cb",
    "thingGroupName" : "MyRootThingGroup",
    "childGroupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "childGroupName" : "MyChildThingGroup"
}
```

The payloads contain the following attributes:

eventType

Set to "THING_GROUP_HIERARCHY_EVENT".

eventId

The event ID.

timestamp

The UNIX timestamp for when the event occurred.

operation

`ADDED` when a thing is added to a thing group. `REMOVED` when a thing is removed from a thing group.

accountId

Your AWS account ID.

thingGroupId

The ID of the parent thing group.

thingGroupName

The name of the parent thing group.

childGroupId

The ID of the child thing group.

childGroupName

The name of the child thing group.

Jobs events

The AWS IoT Jobs service publishes to reserved topics on the MQTT protocol when jobs are pending, completed, or canceled, and when a device reports success or failure when executing a job. Devices or management and monitoring applications can keep track of the status of jobs by subscribing to these topics. Use the UpdateEventConfigurations API to control the kinds of job events you receive.

Because it can take some time to cancel or delete a job, two messages are sent to indicate the start and end of a request. For example, when a cancellation request starts, a message is sent to the `$aws/events/job/jobID/cancellation_in_progress` topic. When the cancellation request is complete, a message is sent to the `$aws/events/job/jobID/canceled` topic. A similar process occurs for a job deletion request. Management and monitoring applications can subscribe to these topics to keep track of the status of jobs.

For more information about publishing and subscribing to MQTT topics, see the section called "Device communication protocols" (p. 76).

Job Completed/Canceled/Deleted

The AWS IoT Jobs service publishes a message on an MQTT topic when a job is completed, canceled, deleted, or when cancellation or deletion are in progress:

- `$aws/events/job/`*`jobID`*`/completed`
- `$aws/events/job/`*`jobID`*`/canceled`
- `$aws/events/job/`*`jobID`*`/deleted`
- `$aws/events/job/`*`jobID`*`/cancellation_in_progress`
- `$aws/events/job/`*`jobID`*`/deletion_in_progress`

The `completed` message contains the following example payload:

```
{
  "eventType": "JOB",
  "eventId": "7364ffd1-8b65-4824-85d5-6c14686c97c6",
  "timestamp": 1234567890,
  "operation": "completed",
  "jobId": "27450507-bf6f-4012-92af-bb8a1c8c4484",
  "status": "COMPLETED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/a39f6f91-70cf-4bd2-a381-9c66df1a80d0",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/2fc4c0a4-6e45-4525-
a238-0fe8d3dd21bb"
  ],
  "description": "My Job Description",
  "completedAt": 1234567890123,
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123,
  "jobProcessDetails": {
    "numberOfCanceledThings": 0,
    "numberOfRejectedThings": 0,
    "numberOfFailedThings": 0,
    "numberOfRemovedThings": 0,
    "numberOfSucceededThings": 3
  }
}
```

The `canceled` message contains the following example payload:

```
{
  "eventType": "JOB",
  "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
  "timestamp": 1234567890,
  "operation": "canceled",
  "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
  "status": "CANCELED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/ThingGroup1-95c644d5-1621-41a6-9aa5-
ad2de581d18f"
  ],
  "description": "My job description",
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123
}
```

The `deleted` message contains the following example payload:

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deleted",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETED",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

The `cancellation_in_progress` message contains the following example payload:

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "cancellation_in_progress",
          "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
          "status": "CANCELLATION_IN_PROGRESS",
          "targetSelection": "SNAPSHOT|CONTINUOUS",
          "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
    cd33d0145a0f",
            "arn:aws:iot:us-east-1:123456789012:thinggroup/
    ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
          ],
          "description": "My job description",
          "createdAt": 1234567890123,
          "lastUpdatedAt": 1234567890123,
          "comment": "Comment for this operation"
        }
```

The `deletion_in_progress` message contains the following example payload:

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deletion_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

Job Execution Terminal Status

The AWS IoT Jobs service publishes a message when a device updates a job execution to terminal status:

- `$aws/events/jobExecution/`*`jobID`*`/succeeded`
- `$aws/events/jobExecution/`*`jobID`*`/failed`
- `$aws/events/jobExecution/`*`jobID`*`/rejected`
- `$aws/events/jobExecution/`*`jobID`*`/canceled`
- `$aws/events/jobExecution/`*`jobID`*`/timed_out`
- `$aws/events/jobExecution/`*`jobID`*`/removed`
- `$aws/events/jobExecution/`*`jobID`*`/deleted`

The message contains the following example payload:

```
{
  "eventType": "JOB_EXECUTION",
  "eventId": "cca89fa5-8a7f-4ced-8c20-5e653afb3572",
  "timestamp": 1234567890,
  "operation": "succeeded|failed|rejected|canceled|removed|timed_out",
  "jobId": "154b39e5-60b0-48a4-9b73-f6f8dd032d27",
  "thingArn": "arn:aws:iot:us-east-1:123456789012:myThing/6d639fbc-8f85-4a90-924d-
a2867f8366a7",
  "status": "SUCCEEDED|FAILED|REJECTED|CANCELED|REMOVED|TIMED_OUT",
  "statusDetails": {
    "key": "value"
  }
}
```

Lifecycle events

AWS IoT publishes lifecycle events on the MQTT topics discussed in the following sections. These messages allow you to be notified of lifecycle events from the message broker.

Note
Lifecycle messages might be sent out of order. You might receive duplicate messages.

Connect/Disconnect events

AWS IoT publishes a message to the following MQTT topics when a client connects or disconnects:

- `$aws/events/presence/connected/clientId` – A client connected to the message broker.
- `$aws/events/presence/disconnected/clientId` – A client disconnected from the message broker.

The following is a list of JSON elements that are contained in the connection/disconnection messages published to the `$aws/events/presence/connected/clientId` topic.

clientId

The client ID of the connecting or disconnecting client.

Note
Client IDs that contain # or + do not receive lifecycle events.

clientInitiatedDisconnect

True if the client initiated the disconnect. Otherwise, false. Found in disconnection messages only.

disconnectReason

The reason why the client is disconnecting. Found in disconnect messages only. The following table contains valid values.

Disconnect reason	Description
`AUTH_ERROR`	The client failed to authenticate or authorization failed.
`CLIENT_INITIATED_DISCONNECT`	The client indicates that it will disconnect. The client can do this by sending either a MQTT `DISCONNECT` control packet or a `Close frame` if the client is using a WebSocket connection.
`CLIENT_ERROR`	The client did something wrong that causes it to disconnect. For example, a client will be disconnected for sending more than 1 MQTT `CONNECT` packet on the same connection or if the client attempts to publish with a payload that exceeds the payload limit.

Disconnect reason	Description
`CONNECTION_LOST`	The client-server connection is cut off. This can happen during a period of high network latency or when the internet connection is lost.
`DUPLICATE_CLIENTID`	The client is using a client ID that is already in use. In this case, the client that is already connected will be disconnected with this disconnect reason.
`FORBIDDEN_ACCESS`	The client is not allowed to be connected. For example, a client with a denied IP address will fail to connect.
`MQTT_KEEP_ALIVE_TIMEOUT`	If there is no client-server communication for 1.5x of the client's keep-alive time, the client is disconnected.
`SERVER_ERROR`	Disconnected due to unexpected server issues.
`SERVER_INITIATED_DISCONNECT`	Server intentionally disconnects a client for operational reasons.
`THROTTLED`	The client is disconnected for exceeding a throttling limit.
`WEBSOCKET_TTL_EXPIRATION`	The client is disconnected because a WebSocket has been connected longer than its time-to-live value.

eventType

The type of event. Valid values are `connected` or `disconnected`.

ipAddress

The IP address of the connecting client. This can be in IPv4 or IPv6 format. Found in connection messages only.

principalIdentifier

The credential used to authenticate. For TLS mutual authentication certificates, this is the certificate ID. For other connections, this is IAM credentials.

sessionIdentifier

A globally unique identifier in AWS IoT that exists for the life of the session.

timestamp

An approximation of when the event occurred, expressed in milliseconds since the Unix epoch. The accuracy of the timestamp is +/- 2 minutes.

versionNumber

The version number for the lifecycle event. This is a monotonically increasing long integer value for each client ID connection. The version number can be used by a subscriber to infer the order of lifecycle events.

Note
The connect and disconnect messages for a client connection have the same version number.

The version number might skip values and is not guaranteed to be consistently increasing by 1 for each event.
If a client is not connected for approximately one hour, the version number is reset to 0. For persistent sessions, the version number is reset to 0 after a client has been disconnected longer than the configured time-to-live (TTL) for the persistent session.

A connect message has the following structure.

```
 {
    "clientId": "186b5",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "a4666d2a7d844ae4ac5d7b38c9cb7967",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

A disconnect message has the following structure.

```
{
    "clientId": "186b5",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "a4666d2a7d844ae4ac5d7b38c9cb7967",
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

Handling client disconnections

The best practice is to always have a wait state implemented for lifecycle events, including Last Will and Testament (LWT) messages. When a disconnect message is received, your code should wait a period of time and verify a device is still offline before taking action. One way to do this is by using SQS Delay Queues. When a client receives a LWT or a lifecycle event, you can enqueue a message (for example, for 5 seconds). When that message becomes available and is processed (by Lambda or another service), you can first check if the device is still offline before taking further action.

Subscribe/Unsubscribe events

AWS IoT publishes a message to the following MQTT topic when a client subscribes or unsubscribes to an MQTT topic:

```
$aws/events/subscriptions/subscribed/clientId
```

or

```
$aws/events/subscriptions/unsubscribed/clientId
```

Where `clientId` is the MQTT client ID that connects to the AWS IoT message broker.

The message published to this topic has the following structure:

```
{
    "clientId": "186b5",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "000000000000/ABCDEFGHIJKLMNOPQRSTU:some-user/
ABCDEFGHIJKLMNOPQRSTU:some-user",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

The following is a list of JSON elements that are contained in the subscribed and unsubscribed messages published to the `$aws/events/subscriptions/subscribed/`*`clientId`* and `$aws/events/subscriptions/unsubscribed/`*`clientId`* topics.

clientId

The client ID of the subscribing or unsubscribing client.

Note
Client IDs that contain # or + do not receive lifecycle events.

eventType

The type of event. Valid values are `subscribed` or `unsubscribed`.

principalIdentifier

The credential used to authenticate. For TLS mutual authentication certificates, this is the certificate ID. For other connections, this is IAM credentials.

sessionIdentifier

A globally unique identifier in AWS IoT that exists for the life of the session.

timestamp

An approximation of when the event occurred, expressed in milliseconds since the Unix epoch. The accuracy of the timestamp is +/- 2 minutes.

topics

An array of the MQTT topics to which the client has subscribed.

Note
Lifecycle messages might be sent out of order. You might receive duplicate messages.

Alexa Voice Service (AVS) Integration for AWS IoT

Alexa Voice Service (AVS) Integration for AWS IoT is a new feature that cost-effectively brings Alexa Voice to any connected device without incurring messaging costs. AVS for AWS IoT reduces the cost and complexity of integrating Alexa. This feature leverages AWS IoT to offload intensive computational and memory audio tasks from the device to the cloud. Because of the resulting reduction in the engineering bill of materials (eBoM) cost, device makers can now cost-effectively bring Alexa to resource-constrained IoT devices and make it possible for consumers to talk directly to Alexa in parts of their home, office, or hotel rooms for an ambient experience.

Currently, smart home IoT devices are built with low-cost microcontrollers (MCU) that have limited memory to run real-time operating systems. Previously, AVS solutions for Alexa built-in products required expensive application processor-based devices with more than 50 MB memory running on Linux or Android. These expensive hardware requirements made it cost-prohibitive to integrate Alexa Voice on resource-constrained IoT devices. AVS for AWS IoT enables Alexa built-in functionality on MCUs, such as the Arm Cortex-M series processors with less than 1 MB embedded RAM. To do so, AVS offloads memory and compute tasks to a virtual Alexa built-in device in the cloud. This reduces eBoM cost by up to 50 percent.

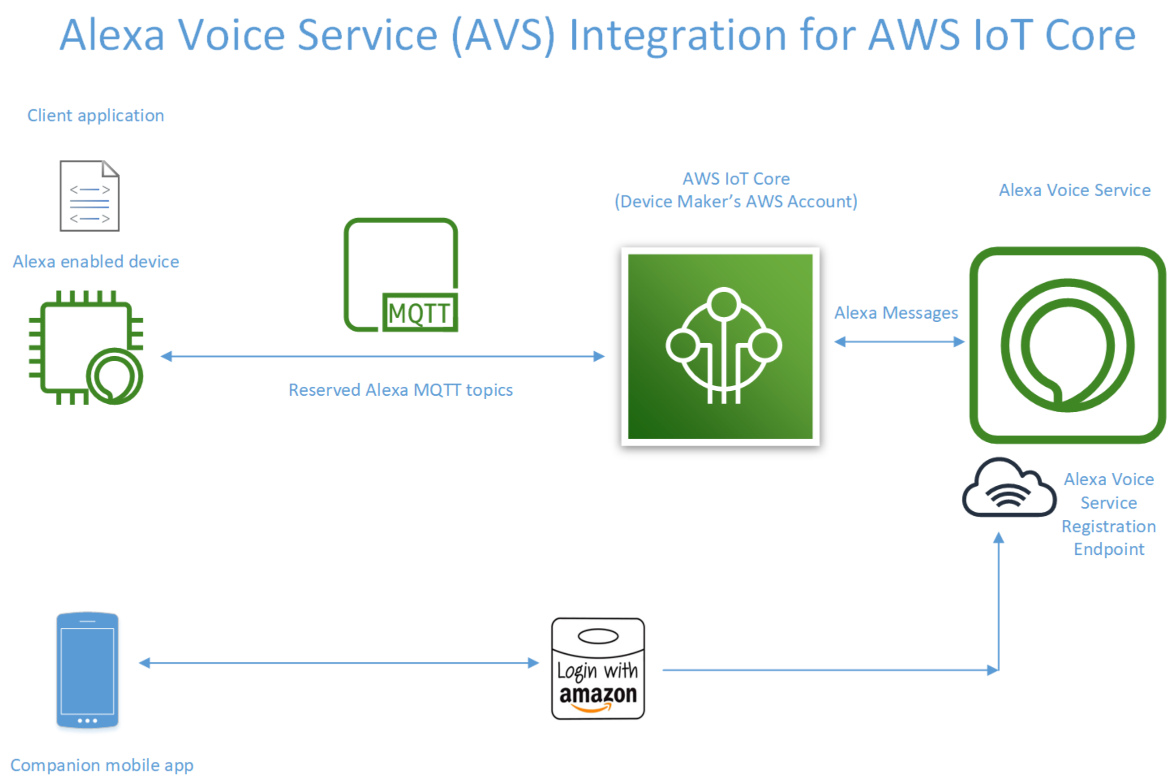
For more information about the Arm Cortex-M series processors, see Arm or Wikipedia. For more information about hardware requirements for Alexa built-in products, see Sizing Up CPU, Memory, and Storage for Your Alexa Built-in Device on the Amazon Alexa developer portal.

Note
AVS for AWS IoT is available in all AWS Regions where AWS IoT is available except in the China (Beijing and Ningxia) Regions. For the current list of AWS Regions, see the AWS Region Table.

AVS for AWS IoT has three components:

- A set of reserved MQTT topics to transfer audio messages between Alexa enabled devices and AVS.
- A virtual Alexa enabled device in the cloud that shifts tasks related to media retrieval, audio decoding, audio mixing, and state management from the physical device to the virtual device.
- A set of APIs that support receiving and sending messages over the reserved topics, interfacing with the device microphone and speaker, and managing device state.

The following diagram illustrates how these components work together. It also demonstrates how device makers use the Login with Amazon to service to authenticate AVS.


Alexa Voice Service (AVS) Integration for AWS IoT Core
Client application
Alexa enabled device
MQTT
Reserved Alexa MQTT topics
AWS IoT Core
(Device Maker's AWS Account)
Alexa Messages
Alexa Voice Service
Alexa Voice Service Registration Endpoint
Login with amazon
Companion mobile app

Device manufacturers have two options to get started with AVS Integration for AWS IoT .

- **Development kits** – Development kits launched by our partners make it easy to get started. The NXP i.MX RT 106 A and Qualcomm Home Hub 100 Development Kit for Amazon AVS are the first two kits available on the market. You can find them on Development Kits for AVS. The kits include out-of-the box connectivity to AWS IoT, AVS qualified Audio Algorithms for Far-Field voice pickup, Echo Cancellation, Alexa Wake Word, and AVS for AWS IoT application code. You can use the feature application code to quickly prototype a device and port the implementation to your chosen MCU design for testing and device production when you're ready.
- **Custom device-side application code** – Developers can also write a custom AVS for AWS IoT application by using the publicly available API. Documentation for this API is available on the AVS developer page. You can download the FreeRTOS and AWS IoT Device SDK from the FreeRTOS console (https://console.aws.amazon.com/freertos/) or GitHub.

To get started with an NXP i.MX 106A development kit, see Getting Started with Alexa Voice Service (AVS) Integration for AWS IoT on an NXP Device.

Getting started with Alexa Voice Service (AVS) Integration for AWS IoT on an NXP device

The NXP i.MX 106A development kit enables you to preview Alexa Voice Service (AVS) Integration for AWS IoT using a preconfigured NXP account. After you preview the functionality with the NXP account, you need to customize the firmware, application source code, and the NXP mobile application provided with the kit to use your own account. This topic walks you through the steps to preview with the preconfigured account and to customize your device with your own account.

Topics

Preview Alexa Voice Service (AVS) Integration for AWS IoT with a preconfigured NXP account

Prerequisites

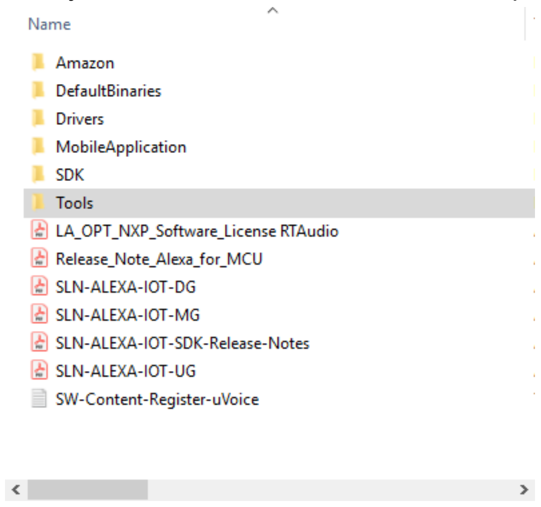
To follow these steps, you need the following resources.

- NXP i.MX 106A development kit
- A Mac, Windows 7/10, or Linux computer
- A serial driver that works with your computer
- An Android or iOS mobile device
- Android Debug Bridge (ADB)
- An Amazon Alexa account

Install and boot the development kit

1. Activate the device kit. The kit comes with a Get Started Onboarding card. This card contains instructions to activate the kit and acquire the necessary software package and the reference design files. The software package contains the SDK source code and the companion Android application (`VoiceCompanionApp.apk`). If you're using an iOS device, you must request access to the TestFlight iOS application from your local NXP representative.

   After you download and extract the software package, you see the following file structure.

   Name

   Amazon
   DefaultBinaries
   Drivers
   MobileApplication
   SDK
   Tools
   LA_OPT_NXP_Software_License RTAudio
   Release_Note_Alexa_for_MCU
   SLN-ALEXA-IOT-DG
   SLN-ALEXA-IOT-MG
   SLN-ALEXA-IOT-SDK-Release-Notes
   SLN-ALEXA-IOT-UG
   SW-Content-Register-uVoice

   The `MobileApplication` folder contains the Android mobile application.

   The `Tools` folder contains scripts that you use to configure your device.

   You can also find the following documentation:

- `SLN-ALEXA-IOT-DG.pdf` in the *NXP Developer Guide*
- `SLN-ALEXA-IOT-MG.pdf` in the *NXP Migration Guide*
- `SLN-ALEXA-IOT-UG.pdf` in the *NXP User Guide*

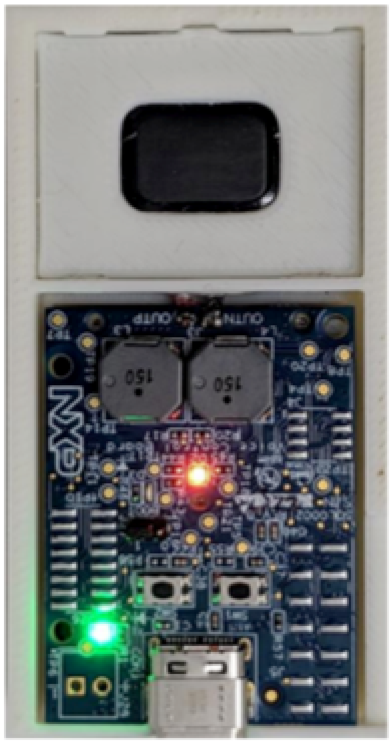
2. Boot the device kit. The USB splitter cable that comes with the kit has power and data connections.

   a. Connect both of the USB-A connections to your computer.

   b. Connect the USB-C connector to your kit. Your configuration might look the following image.

When the board has power, the D1 LED lights up and displays green. The D2 LED (closest to the speaker) indicates the state of the device. When the device turns on, this LED flashes multiple colors. When the device runs the application code, the D2 LED blinks yellow. When initialization of the device is complete, the D2 LED displays orange to indicate that it's waiting for Wi-Fi credentials to be added to the device.

The NXP User Guide contains a description of the device's physical controls.



3. Use the terminal application to understand how the client application on the device works. You can also use it to debug and make sure that the device is in the correct state.

   Enter the following commands to get started with the application:

   - `help` – Displays a list of the available commands.
   - `enable_usb_log` – Enables logging on the device. This helps you understand what the application is doing.
   - `logs` – Displays the last few lines of the logs. At this point, the logs should indicate that the device is waiting for Wi-Fi credentials.

Connect the device to AWS IoT and Amazon Alexa

1. Install the appropriate mobile application. If you're using an Android device, you can use ADB in the terminal to install the `VoiceCompanionApp.apk` file. If you're using an iOS device, use the TestFlight application.
2. Provision the Wi-Fi credentials. You can provision the Wi-Fi credentials by using either the companion mobile application or the terminal.

   a. Follow these steps to provision the Wi-Fi credentials by using the mobile application.

      i. When the device boots up from its factory new state, it creates a Wi-Fi access point. Open the companion mobile application, and choose **WIFI PROVISION** to connect to this access point.

      ii. Choose a network from the list.

      iii. Enter your password, and choose **Send** to transmit the credentials to the device kit.

   b. In the terminal application, enter the following command.

```
setup YourWiFiNetworkName YourWiFiNetworkPassword
```

   c. Choose **Enter**.

3. Authenticate the device kit and connect to AWS IoT and Amazon Alexa. Make sure that your mobile device and the device kit are using the same Wi-Fi network so that the two devices can communicate.

   To do so, press the **Discover** button in the mobile application. When discovery is complete, you see a list of serial numbers for the available device kits near you. If more than one is available, you can confirm your kit's unique serial number by entering `serial_number` in the terminal application.

4. Choose the device kit that to use. The mobile application uses the Login with Amazon service to prompt you for your Amazon user credentials. The device kit begins to register the device with AVS. The D2 LED displays purple to indicate that device registration has started.

   Note
   If your mobile device already has the Amazon shopping application installed, the mobile companion application automatically uses the Amazon account that is logged into the shopping application.

After you sign in to your Amazon account, the companion application performs the following steps:

1. The device kit receives the access token from the Login with Amazon service and begins to connect the device to AWS IoT and AVS. The mobile application displays a completion percentage. The D2 LED on the device displays orange.
2. The D1 LED blinks green every 500 milliseconds until the device connects to AWS IoT.
3. When the device connects to AWS IoT, the device kit begins to connect the device to AVS. The D1 LED blinks green every 250 milliseconds.
4. When the device connects to AVS, the color of the device kit's serial number changes to yellow in the mobile application. The mobile application displays a **Complete** message. The D1 LED turns off, and the device plays a chime. The device is now connected to NXP's AWS IoT account.

You can try a few Alexa Voice commands, such as the following examples:

- "Alexa, what's the weather like?"
- "Alexa, play a news briefing."
- "Alexa, play music."

Use your AWS and Alexa Voice Service developer accounts to set up AVS for AWS IoT

Prerequisites

For this procedure, you need the following resources.

- NXP i.MX 106A development kit
- MCUXPresso v10.3.1
- Segger J-Link debug probe
- A Mac computer
- An Android mobile device
- Android Studio

- Android Debug Bridge
- An Amazon Alexa account
- An AWS account

Configure your AWS IoT account and provision your device

1. Generate credentials to authenticate with AWS IoT. All devices must have a device certificate, private key, and root CA certificate installed in their firmware to communicate with AWS IoT. Follow the instructions in Register a Device in the Registry to register your device with AWS IoT. For more information about X.509 certificates, see X.509 client certificates (p. 203).
2. Navigate to the root folder and open the NXP Developer Guide. Follow the instructions in "Section 9 (Filesystem)" to convert your client certificates and keys to a binary format that the NXP filesystem can read.

   Use the following script and commands to convert the certificates and keys. The `bin_dump.py` script is in the `Tools\Ivaldi.zip\Scripts\sln_alexa_iot_utils` folder.

   ```
   python3 bin_dump.py CertificateName-certificate.pem.crt CertificateName-
   certificate.pem.crt.bin
   ```

   ```
   python3 bin_dump.py CertificateName-private.pem.key CertificateName-private.pem.key.bin
   ```

For information about options for generating credentials for production devices at scale, see Device provisioning (p. 658).

To set up your device in the Alexa Voice Service developer console

1. Navigate to the root folder and open the NXP Migration Guide.
2. To get the MD5 and SHA256 signatures that you need to create an AVS API key, follow the instructions in "Section 2. Creating a Keystore for Android AVS Companion Application" in the *NXP Migration Guide*. (This includes navigating to the Alexa Voice Service Developer Console.)
3. To create a Login with Amazon security profile and create an AVS product, follow the instructions in "Section 3" in the *NXP Migration Guide*.

To rebuild the Android mobile app

1. Open the VoiceCompanionApp project in Android Studio. The `VoiceCompanionApp.apk` file is included in the NXP software package.
2. In Android Studio, add the API key that you generated in the previous procedure to the `api_key.txt` file in the **assets** folder. This synchronizes the companion mobile application with your AVS security profile. The application uses the API key when you log in by using the Login with Amazon service. When the application gets an authorization code from Amazon, it transfers the code to the device firmware. The device then obtains access and refresh tokens from the Login with Amazon service to make calls to AVS.
3. In Android Studio, choose **Build Project** and **Generate a Signed Bundle/APK**. For more information, see "Section 4. Building and Generating the Mobile Application" in the *NXP Migration Guide*. For more information about authorizing with Alexa in this way, see Authorize from a Companion App (Android/iOS).
4. Install the updated mobile application to your mobile device. Set up ADB on your Mac computer and install the updated Android APK file to your mobile phone by using the terminal console. If you installed the mobile application to preview the AVS for AWS IoT service with the preconfigured NXP account, you must uninstall and then reinstall the application.

To update the client application with your AWS credentials

1. Follow the instructions in Section 3 and Section 4 in the *NXP Developer Guide* to make the required Segger J-Link driver modifications. These instructions also explain how to import the `Bootloader`, `ais_demo`, and `bootstrap` projects by using the MCUXPresso IDE.
2. Follow the instructions for updating the source code and rebuilding the application in "Section 7. RT106A Firmware Changes" in the *NXP Migration Guide*. We recommend that you use MCUXPresso v10.3.1.

   These source code changes enable the device firmware to communicate with your AWS account and your AVS product instead of the default NXP account.

To set up and connect your device to AWS IoT and Amazon Alexa

1. Reset the device to factory defaults. If you previewed the AVS for AWS IoT service with the preconfigured NXP account, you must reset the device firmware before you log in again with the companion mobile application. This ensures that the mobile application and the device use your security profile. Make sure that the device is connected to your computer. Push **SW1** for 10 seconds to initiate the factory reset.

   The *NXP User Guide* that is included in the NXP software package contains a description of the device's physical controls.
2. Reprogram the firmware to use the certificate and key that you generated for your device. Add the key and certificate to the updated `Bootstrap` and `Ais_demo` binaries that you created in the previous procedure. We also recommend reprogramming the firmware with the default `Intelligent toolbox`, `app_crt` and `CA_crt` binaries.

   For instructions, see "Section 5. Building and Programming" in the *NXP Developer Guide*.
3. Follow the instructions in the previous Connect the Device to AWS and Amazon Alexa (p. 984) procedure. This connects you to AVS for AWS IoT with your own security profile and credentials.

Amazon Sidewalk Integration for AWS IoT Core

Amazon Sidewalk is a shared network that improves connectivity options to help devices work together better. Amazon Sidewalk supports a wide range of customer devices such as those that locate pets or valuables, those that provide smart home security and lighting control, and those that provide remote diagnostics for appliances and tools. Amazon Sidewalk Integration for AWS IoT Core makes it possible for device manufacturers to add their Sidewalk device fleet to the AWS Cloud.

Getting started with Amazon Sidewalk Integration for AWS IoT Core

If you are a device manufacturer, here's how you can add your Sidewalk-enabled devices to AWS IoT Core.

1. **Review the Sidewalk SDK and documentation**

   Learn more about Amazon Sidewalk and how your devices can use it.

   a. Review the Amazon Sidewalk Quick Start Guide.
   b. Download an SDK for Amazon Sidewalk.
   c. Open the Sidewalk Developer Service (SDS) console.
2. **Register your prototype device**

   In the SDS console, register your prototype device with Amazon Sidewalk.
3. **Associate your Sidewalk Amazon ID with your AWS account**

   In the AWS IoT console, associate your Sidewalk Amazon ID with your AWS account.

   Your Amazon Sidewalk devices appear in the **Sidewalk** tab of the **Devices** hub of the AWS IoT console.
4. **Complete the Amazon Sidewalk device configuration in the AWS IoT console**

   Create the AWS IoT Core for LoRaWAN destinations and rules your device needs to route and format data for AWS services.

After your Amazon Sidewalk devices are authenticated, their messages are sent to AWS IoT Core. You can start developing your business applications on the AWS Cloud that use the data from your Amazon Sidewalk devices.

Learn more

These resources can help you integrate your Amazon Sidewalk devices with AWS IoT solutions.

- **Amazon Sidewalk Quick Start Guide**

  The *Amazon Sidewalk Quick Start Guide* describes how to set up your hardware, design your product, connect to AWS IoT Core, and other steps necessary to create and test your device.
- **AWS IoT Wireless API actions for Amazon Sidewalk Integration for AWS IoT Core (p. 989)**

  Lists the AWS IoT Wireless API actions that support the Amazon Sidewalk Integration for AWS IoT Core, with links to their API reference topics.
- **Sidewalk Developer Service (SDS) console**

Add and manage your devices that support Amazon Sidewalk.

- **AWS IoT console**

  Manage your Sidewalk devices and other AWS IoT resources.

AWS IoT Wireless API actions for Amazon Sidewalk Integration for AWS IoT Core

These actions of the AWS IoT Wireless API support the Amazon Sidewalk Integration for AWS IoT Core.

- AssociateAwsAccountWithPartnerAccount
- DisasociateAwsAccountFromPartnerAccount
- GetPartnerAccount
- ListPartnerAccounts
- UpdatePartnerAccount

For the complete listing of the actions and data types that the AWS IoT Wireless API supports, see the AWS IoT Wireless API Reference.

AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client

This page summarizes the AWS IoT Device SDKs, open-source libraries, developer guides, sample apps, and porting guides to help you build innovative IoT solutions with AWS IoT and your choice of hardware platforms.

These SDKs are for use on your IoT device. If you're developing an IoT app for use on a mobile device, see the AWS Mobile SDKs (p. 992). If you're developing an IoT app or server-side program, see the AWS SDKs (p. 68).

AWS IoT Device SDKs

The AWS IoT Device SDKs include open-source libraries, developer guides with samples, and porting guides so that you can build innovative IoT products or solutions on your choice of hardware platforms.

These SDKs help you connect your IoT devices to AWS IoT using the MQTT and WSS protocols.

C++

AWS IoT C++ Device SDK

The AWS IoT C++ Device SDK allows developers to build connected applications using AWS and the AWS IoT APIs. Specifically, this SDK was designed for devices that are not resource constrained and require advanced features such as message queuing, multi-threading support, and the latest language features. For more information, see the following:

- AWS IoT C++ Device SDK v2 on GitHub
- AWS IoT C++ Device SDK v2 Readme
- AWS IoT C++ Device SDK v2 Samples

Python

AWS IoT Device SDK for Python

The AWS IoT Device SDK for Python makes it possible for developers to write Python scripts to use their devices to access the AWS IoT platform through MQTT or MQTT over the WebSocket protocol. By connecting their devices to AWS IoT, users can securely work with the message broker, rules, and shadows provided by AWS IoT and with other AWS services like AWS Lambda, Kinesis, and Amazon S3, and more.

- AWS IoT Device SDK for Python v2 on GitHub
- AWS IoT Device SDK for Python v2 Readme
- AWS IoT Device SDK for Python v2 Samples
- AWS IoT Device SDK for Python v2 API documentation

JavaScript

AWS IoT Device SDK for JavaScript

The aws-iot-device-sdk.js package makes it possible for developers to write JavaScript applications that access AWS IoT using MQTT or MQTT over the WebSocket protocol. It can be used in Node.js environments and browser applications. For more information, see the following:

- AWS IoT Device SDK for JavaScript v2 on GitHub
- AWS IoT Device SDK for JavaScript v2 Readme
- AWS IoT Device SDK for JavaScript v2 Samples
- AWS IoT Device SDK for JavaScript v2 API documentation

Java

AWS IoT Device SDK for Java

The AWS IoT Device SDK for Java makes it possible for Java developers to access the AWS IoT platform through MQTT or MQTT over the WebSocket protocol. The SDK is built with shadow support. You can access shadows by using HTTP methods, including GET, UPDATE, and DELETE. The SDK also supports a simplified shadow access model, which allows developers to exchange data with shadows by just using getter and setter methods, without having to serialize or deserialize any JSON documents. For more information, see the following:

- AWS IoT Device SDK for Java v2 on GitHub
- AWS IoT Device SDK for Java v2 Readme
- AWS IoT Device SDK for Java v2 Samples

AWS IoT Device SDK for Embedded C

Note
This SDK is intended for use by experienced embedded-software developers.

The AWS IoT Device SDK for Embedded C (C-SDK) is a collection of C source files under the MIT open source license that can be used in embedded applications to securely connect IoT devices to AWS IoT Core. It includes an MQTT, JSON Parser, and AWS IoT Device Shadow library. It is distributed in source form and intended to be built into customer firmware along with application code, other libraries and, optionally, an RTOS (Real Time Operating System).

For Fleet Provisioning, use the `v4_beta_deprecated` version of the AWS IoT Device SDK for Embedded C at https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/v4_beta_deprecated. Please review the README in this branch for more details.

The AWS IoT Device SDK for Embedded C is generally targeted at resource constrained devices that require an optimized C language runtime. You can use the SDK on any operating system and host it on any processor type (for example, MCUs and MPUs). However, if your devices have sufficient memory and processing resources, we recommend that you use one of the higher order the section called “AWS IoT Device SDKs” (p. 990).

For more information, see the following:

- AWS IoT Device SDK for Embedded C on GitHub
- AWS IoT Device SDK for Embedded C Readme
- AWS IoT Device SDK for Embedded C Samples

Earlier AWS IoT Device SDKs versions

These are earlier versions of AWS IoT Device SDKs that have been replaced by the newer versions listed above. These SDKs are receiving only maintenance and security updates. They will not be updated to include new features and should not be used on new projects.

- AWS IoT C++ Device SDK on GitHub
- AWS IoT C++ Device SDK Readme
- AWS IoT Device SDK for Python v1 on GitHub
- AWS IoT Device SDK for Python v1 Readme
- AWS IoT Device SDK for Java on GitHub
- AWS IoT Device SDK for Java Readme
- AWS IoT Device SDK for JavaScript on GitHub
- AWS IoT Device SDK for JavaScript Readme
- Arduino Yún SDK on GitHub
- Arduino Yún SDK Readme

AWS Mobile SDKs

The AWS Mobile SDKs provide mobile app developers platform-specific support for the APIs of the AWS IoT Core services, IoT device communication using MQTT, and the APIs of other AWS services.

Android

AWS SDK for Android

The AWS SDK for Android contains a library, samples, and documentation for developers to build connected mobile applications using AWS. This SDK also includes support for MQTT device communications and calling the APIs of the AWS IoT Core services. For more information, see the following:

- AWS Mobile SDK for Android on GitHub
- AWS Mobile SDK for Android Readme
- AWS Mobile SDK for Android Samples
- AWS SDK for Android API reference
- AWSIoTClient Class reference documentation

iOS

AWS SDK for iOS

The AWS SDK for iOS is an open-source software development kit, distributed under an Apache Open Source license. The SDK for iOS provides a library, code samples, and documentation to help developers build connected mobile applications using AWS. This SDK also includes support for MQTT device communications and calling the APIs of the AWS IoT Core services. For more information, see the following:

- AWS SDK for iOS on GitHub
- AWS SDK for iOS Readme
- AWS SDK for iOS Samples
- AWSIoT Class reference docs in the AWS SDK for iOS

AWS IoT Device Client

The AWS IoT Device Client provides code to help your device connect to AWS IoT, perform fleet provisioning tasks, support device security policies, connect using secure tunneling, and process jobs on your device. You can install this software on your device to handle these routine device tasks so you can focus on your specific solution.

Note
The AWS IoT Device Client works with microprocessor-based IoT devices with x86_64 or ARM processors and common Linux operating systems.

C++

AWS IoT Device Client

For more information about the AWS IoT Device Client in C++, see the following:

- AWS IoT Device Client in C++ source code on GitHub
- AWS IoT Device Client in C++ Readme

Troubleshooting AWS IoT

The following information might help you troubleshoot common issues in AWS IoT.

Tasks

Diagnosing connectivity issues

A successful connection to AWS IoT requires:

- A valid connection
- A valid and active certificate
- A policy that allows the desired connection and operation

Connection

How do I find the correct endpoint?

- The `endpointAddress` returned by **aws iot describe-endpoint --endpoint-type iot:Data-ATS**

  or

- The `domainName` returned by **aws iot describe-domain-configuration –-domain-configuration-name "*`domain_configuration_name`*"**

How do I find the correct Server Name Indication (SNI) value?

The correct SNI value is the `endpointAddress` returned by the **describe-endpoint** or **describe-domain-configuration** commands. It's the same address as the endpoint in the previous step.

Authentication

Devices must be authenticated (p. 203) to connect to AWS IoT endpoints. For devices that use X.509 client certificates (p. 203) for authentication, the certificates must be registered with AWS IoT and be active.

How do my devices authenticate AWS IoT endpoints?

Add the AWS IoT CA certificate to your client's trust store. Refer to the documentation on Server Authentication in AWS IoT Core and then follow the links to download the appropriate CA certificate.

What is checked when a device connects to AWS IoT?

When a device attempts to connect to AWS IoT:

1. AWS IoT checks for a valid certificate and Server Name Indication (SNI) value.
2. AWS IoT checks to see that the certificate used is registered with the AWS Account and that it has been activated.
3. When a device attempts to perform any action in AWS IoT, such as to subscribe to or publish a message, the policy attached to the certificate it used to connect is checked to confirm that the device is authorized to perform that action.

How can I validate a correctly configured certificate?

Use the OpenSSL `s_client` command to test a connection to the AWS IoT endpoint:

```
 openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

For more information about using `openssl s_client`, see OpenSSL s_client documentation.

How do I check the status of a certificate?

- **List the certificates**

  If you don't know the certificate ID, you can see the status of all your certificates by using the **aws iot list-certificates** command.
- **Show a certificate's details**

  If you know the certificate's ID, this command shows you more detailed information about the certificate.

  ```
  aws iot describe-certificate --certificate-id "certificateId"
  ```
- **Review the certificate in the AWS IoT Console**

  In the AWS IoT console, in the left menu, choose **Secure**, and then choose **Certificates**.

  Choose the certificate that you are using to connect from the list to open its detail page.

  In the certificate's detail page, you can see its current status.

  The certificate's status can be changed by using the **Actions** menu in the upper-right corner of the details page.

Authorization

AWS IoT resources use AWS IoT Core policies (p. 235) to authorize those resources to perform actions (p. 236). For an action to be authorized, the specified AWS IoT resources must have a policy document attached to it that grants permission to perform that action.

I received a `PUBNACK` or `SUBNACK` response from the broker. What do I do?

Make sure that there is a policy attached to the certificate you are using to call AWS IoT. All publish/subscribe operations are denied by default.

Make sure the attached policy authorizes the actions (p. 236) you are trying to perform.

Make sure the attached policy authorizes the resources (p. 237) that are trying to perform the authorized actions.

I have an *AUTHORIZATION_FAILURE* entry in my logs.

Make sure that there is a policy attached to the certificate you are using to call AWS IoT. All publish/subscribe operations are denied by default.

Make sure the attached policy authorizes the actions (p. 236) you are trying to perform.

Make sure the attached policy authorizes the resources (p. 237) that are trying to perform the authorized actions.

How do I check what the policy authorizes?

In the AWS IoT console, in the left menu, choose **Secure**, and then choose **Certificates**.

Choose the certificate that you are using to connect from the list to open its detail page.

In the certificate's detail page, you can see its current status.

In the left menu of the certificate's detail page, choose **Policies** to see the policies attached to the certificate.

Choose the desired policy to see its details page.

In the policy's details page, review the policy's **Policy document** to see what it authorizes.

Choose **Edit policy document** to make changes to the policy document.

Diagnosing rules issues

This section describes some of the things to check when you encounter a problem with rule.

Configuring CloudWatch Logs for troubleshooting

The best way to debug issues you are having with rules is to use CloudWatch Logs. When you enable CloudWatch Logs for AWS IoT, you can see which rules are triggered and their success or failure. You also get information about whether WHERE clause conditions match. For more information, see Monitor AWS IoT using CloudWatch Logs (p. 330).

The most common rules issue is authorization. The logs show if your role is not authorized to perform AssumeRole on the resource. Here is an example log generated by fine-grained logging (p. 315):

```
{
    "timestamp": "2017-12-09 22:49:17.954",
    "logLevel": "ERROR",
    "traceId": "ff563525-6469-506a-e141-78d40375fc4e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecution",
    "clientId": "iotconsole-123456789012-3",
    "topicName": "test-topic",
    "ruleName": "rule1",
    "ruleAction": "DynamoAction",
    "resources": {
        "ItemHashKeyField": "id",
        "Table": "trashbin",
        "Operation": "Insert",
        "ItemHashKeyValue": "id",
        "IsPayloadJSON": "true"
    },
    "principalId": "ABCDEFG1234567ABCD890:outis",
    "details": "User: arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJH
 is not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/testbin (Service: AmazonDynamoDBv2; Status Code: 400; Error Code:
 AccessDeniedException; Request ID: AKQJ987654321AKQJ123456789AKQJ987654321AKQJ987654321)"
}
```

Here is a similar example log generated by global logging (p. 314):

```
2017-12-09 22:49:17.954 TRACEID:ff562535-6964-506a-e141-78d40375fc4e
PRINCIPALID:ABCDEFG1234567ABCD890:outis [ERROR] EVENT:DynamoActionFailure
TOPICNAME:test-topic CLIENTID:iotconsole-123456789012-3
MESSAGE:Dynamo Insert record failed. The error received was User:
 arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJI is not authorized to
 perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
testbin
(Service: AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException; Request
 ID: AKQJ987654321AKQJ987654321AKQJ987654321AKQJ987654321).
Message arrived on: test-topic, Action: dynamo, Table: trashbin, HashKeyField: id,
 HashKeyValue: id, RangeKeyField: None, RangeKeyValue: 123456789012
No newer events found at the moment. Retry.
```

For more information, see the section called “Viewing AWS IoT logs in the CloudWatch console” (p. 330).

Diagnosing external services

External services are controlled by the end user. Before rule execution, make sure that the external services you have linked to your rule are set up and have enough throughput and capacity units for your application.

Diagnosing SQL problems

If your SQL query is not returning the data you expect:

- **Review the logs for error messages.**
- **Confirm that your SQL syntax matches the JSON document in the message.**

  Review the object and property names used in the query with those used in the JSON document of the topic's message payload. For more information about the JSON formatting in SQL queries, see JSON extensions (p. 484).
- **Check to see if the JSON object or property names include reserved or numeric characters.**

  For more information about reserved characters in JSON object references in SQL queries, see JSON extensions (p. 484).

Diagnosing problems with shadows

Diagnosing shadows

Issue	Troubleshooting guidelines
A device's shadow document is rejected with `Invalid JSON document`.	If you are unfamiliar with JSON, modify the examples provided in this guide for your own use. For more information, see Shadow document examples (p. 527).
I submitted correct JSON, but none or only parts of it are stored in the device's shadow document.	Be sure you are following the JSON formatting guidelines. Only JSON fields in the `desired` and

Issue	Troubleshooting guidelines
	`reported` sections are stored. JSON content (even if formally correct) outside of those sections is ignored.
I received an error that the device's shadow exceeds the allowed size.	The device's shadow supports 8 KB of data only. Try shortening field names inside of your JSON document or simply create more shadows by creating more things. A device can have an unlimited number of things/shadows associated with it. The only requirement is that each thing name must be unique in your account.
When I receive a device's shadow, it is larger than 8 KB. How can this happen?	Upon receipt, the AWS IoT service adds metadata to the device's shadow. The service includes this data in its response, but it does not count toward the limit of 8 KB. Only the data for `desired` and `reported` state inside the state document sent to the device's shadow counts toward the limit.
My request has been rejected due to incorrect version. What should I do?	Perform a GET operation to sync to the latest state document version. When using MQTT, subscribe to the ./update/accepted topic to be notified about state changes and receive the latest version of the JSON document.
The timestamp is off by several seconds.	The timestamp for individual fields and the whole JSON document is updated when the document is received by the AWS IoT service or when the state document is published onto the ./update/accepted and ./update/delta message. Messages can be delayed over the network, which can cause the timestamp to be off by a few seconds.
My device can publish and subscribe on the corresponding shadow topics, but when I attempt to update the shadow document over the HTTP REST API, I get HTTP 403.	Be sure you have created policies in IAM to allow access to these topics and for the corresponding action (UPDATE/GET/DELETE) for the credentials you are using. IAM policies and certificate policies are independent.
Other issues.	The Device Shadow service logs errors to CloudWatch Logs. To identify device and configuration issues, enable CloudWatch Logs and view the logs for debug information.

Diagnosing Salesforce IoT input stream action issues

Execution trace

How do I see the execution trace of a Salesforce action?

See the Monitor AWS IoT using CloudWatch Logs (p. 330) section. After you have activated the logs, you can see the execution trace of the Salesforce action.

Action success and failure

How do I check that messages have been sent successfully to a Salesforce IoT input stream?

View the logs generated by execution of the Salesforce action in CloudWatch Logs. If you see `Action executed successfully`, then it means that the AWS IoT rules engine received confirmation from the Salesforce IoT that the message was successfully pushed to the targeted input stream.

If you are experiencing problems with the Salesforce IoT platform, contact Salesforce IoT support.

What do I do if messages have not been sent successfully to a Salesforce IoT input stream?

View the logs generated by execution of the Salesforce action in CloudWatch Logs. Depending on the log entry, you can try the following actions:

`Failed to locate the host`

Check that the `url` parameter of the action is correct and that your Salesforce IoT input stream exists.

`Received Internal Server Error from Salesforce`

Retry. If the problem persists, contact Salesforce IoT Support.

`Received Bad Request Exception from Salesforce`

Check the payload you are sending for errors.

`Received Unsupported Media Type Exception from Salesforce`

Salesforce IoT does not support a binary payload at this time. Check that you are sending a JSON payload.

`Received Unauthorized Exception from Salesforce`

Check that the `token` parameter of the action is correct and that your token is still valid.

`Received Not Found Exception from Salesforce`

Check that the `url` parameter of the action is correct and that your Salesforce IoT input stream exists.

If you receive an error that is not listed here, contact AWS Support.

Troubleshooting aggregation queries for the fleet indexing service

If you are having type mismatch errors, you can use CloudWatch Logs to troubleshoot the problem. CloudWatch Logs must be enabled before logs are written by the Fleet Indexing service. For more information, see Monitor AWS IoT using CloudWatch Logs (p. 330).

When you make aggregation queries on non-managed fields, you can only specify a field you defined in the `customFields` argument passed to `UpdateIndexingConfiguration` or **update-indexing-configuration**. If the field value is inconsistent with the configured field data type, this value is ignored when you perform an aggregation query.

The Fleet Indexing service emits an error log to CloudWatch Logs when a field cannot be indexed because of a mismatched type. The error log contains the field name, the value that could not be converted, and the thing name for the device. The following is an example error log:

```
{
  "timestamp": "2017-02-20 20:31:22.932",
  "logLevel": "ERROR",
  "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
  "accountId": "000000000000",
  "status": "SucceededWithIssues",
  "eventType": "IndexingCustomFieldFailed",
  "thingName": "thing0",
  "failedCustomFields": [
    {
      "Name": "attributeName1",
      "Value": "apple",
      "ExpectedType": "String"
    },
    {
      "Name": "attributeName2",
      "Value": "2",
      "ExpectedType": "Boolean"
    }
  ]
}
```

If a device has been disconnected for approximately an hour, the connectivity status `timestamp` value might be missing. For persistent sessions, the value might be missing after a client has been disconnected longer than the configured time-to-live (TTL) for the persistent session. The connectivity status data is indexed only for connections where the client ID has a matching thing name. (The client ID is the value used to connect a device to AWS IoT Core.)

Troubleshooting "Stream limit exceeded for your AWS account"

If you see `"Error: You have exceeded the limit for the number of streams in your AWS account."`, you can clean up the unused streams in your account instead of requesting a limit increase.

To clean up an unused stream that you created using the AWS CLI or SDK:

```
aws iot delete-stream -stream-id value
```

For more details, see delete-stream.

Note
You can use the `list-streams` command to find the stream IDs.

AWS IoT Device Defender troubleshooting guide

General

Q: Are there any prerequisites for using AWS IoT Device Defender?

A: If you want to use device-reported metrics, you must first deploy an agent on your AWS IoT connected devices or device gateways. Devices must provide a consistent client identifier or thing name.

Audit

Q: I enabled a check and my audit has been showing "In-Progress" for a long time. Is something wrong? When can I expect results?

A: When a check is enabled, data collection starts immediately. However, if your account has a large amount of data to collect (certificates, things, policies, and so on), the results of the check might not be available for some time after you have enabled it.

Detect

Q: How do I know the thresholds to set in an AWS IoT Device Defender security profile behavior?

A: Start by creating a security profile behavior with low thresholds and attach it to a thing group that contains a representative set of devices. You can use AWS IoT Device Defender to view the current metrics, and then fine-tune the device behavior thresholds to match your use case.

Q: I created a behavior, but it is not triggering a violation when I expect it to. How should I fix it?

A: When you define a behavior, you are specifying how you expect your device to behave normally. For example, if you have a security camera that only connects to one central server on TCP port 8888, you don't expect it to make any other connections. To be alerted if the camera makes a connection on another port, you define a behavior like this:

```
{
  "name": "Listening TCP Ports",
  "metric": "aws:listening-tcp-ports",
  "criteria": {
    "comparisonOperator": "in-port-set",
    "value": {
      "ports": [ 8888 ]
    }
  }
}
```

If the camera makes a TCP connection on TCP port 443, the device behavior would be violated and an alert would be triggered.

Q: One or more of my behaviors are in violation. How do I clear the violation?

A: Alarms clear after the device returns to expected behavior, as defined in the behavior profiles. Behavior profiles are evaluated upon receipt of metrics data for your device. If the device doesn't publish any metrics for more than two days, the violation event is set to `alarm-invalidated` automatically.

Q: I deleted a behavior that was in violation, but how do I stop the alerts?

A: Deleting a behavior stops all future violations and alerts for that behavior. Earlier alerts must be drained from your notification mechanism. When you delete a behavior, the record of violations of that behavior is retained for the same time period as all other violations in your account.

Device Metrics

Q: I'm submitting metrics reports that I know violate my behaviors, but no violations are being triggered. What's wrong?

A: Check that your metrics reports are being accepted by subscribing to the following MQTT topics:

```
$aws/things/THING_NAME/defender/metrics/FORMAT/rejected
$aws/things/THING_NAME/defender/metrics/FORMAT/accepted
```

where `THING_NAME` is the name of the thing reporting the metric and `FORMAT` is either "json" or "cbor", depending on the format of the metrics report submitted by the thing.

After you have subscribed, you should receive messages on these topics for each metric report submitted. A `rejected` message indicates that there was a problem parsing the metric report. An error message is included in the message payload to help you correct any errors in your metric report. An `accepted` message indicates the metric report was parsed properly.

Q: What happens if I send an empty metric in my metric report?

A: An empty list of ports or IP addresses is always considered in conformity with the corresponding behavior. If the corresponding behavior was in violation, the violation is cleared.

Q: Why do my device metric reports contain messages for devices that aren't in the AWS IoT registry?

If you have one or more security profiles attached to all things or to all unregistered things, AWS IoT Device Defender includes metrics from unregistered things. If you want to exclude metrics from unregistered things, you can attach the profiles to all registered devices instead of all devices.

Q: I'm not seeing messages from one or more unregistered devices even though I applied a security profile to all unregistered devices or all devices. How can I fix it?

Verify that you are sending a well-formed metrics report using one of the supported formats, For information, see Device metrics document specification (p. 854). Verify that the unregistered devices are using a consistent client identifier or thing name. Messages reported by devices are rejected if the thing name contains control characters or if the thing name is longer than 128 bytes of UTF-8 encoded characters.

Q: What happens if an unregistered device is added to the registry or a registered device becomes unregistered?

A: If a device is added to or removed from the registry:

- You see two separate violations for the device (one under its registered thing name, one under its unregistered identity) if it continues to publish metrics for violations. Active violations for the old identity stop appearing after two days, but are available in violations history for up to 14 days.

Q: Which value should I supply in the report ID field of my device metrics report?

A: Use a unique value for each metric report, expressed as a positive integer. A common practice is to use a Unix epoch timestamp.

Q: Should I create a dedicated MQTT connection for AWS IoT Device Defender metrics?

A: A separate MQTT connection is not required.

Q: Which client ID should I use when connecting to publish device metrics?

For devices (things) that are in the AWS IoT registry, use the registered thing name. For devices that are not in the AWS IoT registry, use a consistent identifier when you connect to AWS IoT. This practice helps match the violations to the thing name.

Q: Can I publish metrics for a device with a different client ID?

It is possible to publish metrics on behalf of another thing. You can do this by publishing the metrics to the AWS IoT Device Defender reserved topic for that device. For example, `Thing-1` would like to publish metrics for itself and also on behalf of `Thing-2`. `Thing-1` collects its own metrics and publishes them on the MQTT topic:

```
$aws/things/Thing-1/defender/metrics/json
```

`Thing-1` then obtains metrics from `Thing-2` and publishes those metrics on the MQTT topic:

```
$aws/things/Thing-2/defender/metrics/json
```

Q: How many security profiles and behaviors can I have in my account?

A: See AWS IoT Device Defender Endpoints and Quotas.

Q: What does a prototypical target role for an alert target look like?

A: A role that allows AWS IoT Device Defender to publish alerts on an alert target (SNS topic) requires two things:

- A trust relationship that specifies iot.amazonaws.com as the trusted entity.
- An attached policy that grants AWS IoT permission to publish on a specified SNS topic. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "<sns-topic-arn>"
        }
    ]
}
```

- If the SNS topic used for publishing alerts is an encrypted topic, then along with the permission to publish to SNS topic, AWS IoT needs to be granted two more permissions. For example:

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "sns:Publish",
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": "<sns-topic-arn>"
       }
    ]
}
```

Device Advisor troubleshooting guide

General

Q: Can I run multiple test suites in parallel?

A: No. Currently Device Advisor does not support running multiple test suites since only one Device Advisor endpoint is available per account. However, you can test multiple device types by sequencing the test suites one after the other.

Q: I saw from my device that the TLS connection got denied by Device Advisor. Is this expected?

A: Yes. Device Advisor denies the TLS connection before and after each test run. We recommend users to implement device retry mechanism in order to have a full-automated testing experience with Device Advisor. If you execute a test suite with more than one test case say - TLS connect, MQTT connect and MQTT publish then we recommend that you have a mechanism built for your device to try connecting to our test end point every 5 seconds for a minute to two. This will enable you to run multiple test cases in sequence in automated manner.

Q: Can I get a history of Device Advisor API calls made on my account for security analysis and operational troubleshooting purposes?

A: Yes. To receive a history of Device Advisor API calls made on your account, you simply turn on CloudTrail in the AWS Management Console and filter the event source to be `iotdeviceadvisor.amazonaws.com`.

Q: How do I view Device Advisor logs in CloudWatch?

Logs generated during a test suite run are uploaded to CloudWatch if you add the required policy (for example, **CloudWatchFullAccess**) to your service role (see Setting up (p.)). A log group "aws/iot/deviceadvisor/$testSuiteId" will be created. In this log group, two log streams will be created if there is at least one test case in your test suite. One is named "$testRunId" and includes logs of actions taken before and after executing the test cases in your test suite, such as setup and cleanup steps. Another is "$suiteRunId_$testRunId" which is specific to a test suite run. Events sent from devices and AWS IoT Core will be logged to this log stream.

AWS IoT errors

This section lists the error codes sent by AWS IoT.

Message broker error codes

Error code	Error description
400	Bad request.
401	Unauthorized.
403	Forbidden.
503	Service unavailable.

Identity and security error codes

Error code	Error description
401	Unauthorized.

Device shadow error codes

Error code	Error description
400	Bad request.
401	Unauthorized.
403	Forbidden.
404	Not found.
409	Conflict.
413	Request too large.
422	Failed to process request.

Error code	Error description
429	Too many requests.
500	Internal error.
503	Service unavailable.

AWS IoT quotas

For AWS IoT Core quotas information, see AWS IoT Core Endpoints and Quotas in the *AWS General Reference*.