



Developer Guide

AWS IoT Wireless



AWS IoT Wireless: Developer Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS IoT Wireless?	1
Features of AWS IoT Wireless	1
Onboard LoRaWAN and Sidewalk devices	1
Integration with AWS IoT Core	2
For first-time users of AWS IoT Wireless	2
Related services	3
Accessing AWS IoT Wireless	3
Getting started	5
Setting up AWS IoT Wireless	5
Set up your AWS account	5
Installing Python and the AWS CLI	8
Describing your wireless resources	10
Resource names and description	11
Resource tags	12
AWS IoT Core for LoRaWAN	13
Introduction	13
Accessing AWS IoT Core for LoRaWAN	13
AWS IoT Core for LoRaWAN Regions and endpoints	14
AWS IoT Core for LoRaWAN pricing	14
What is AWS IoT Core for LoRaWAN?	15
Features of AWS IoT Core for LoRaWAN	15
What is LoRaWAN?	16
How AWS IoT Core for LoRaWAN works	17
Connecting to AWS IoT Core for LoRaWAN	19
Naming conventions for your devices, gateways, profiles, and destinations	19
Mapping of device data to service data	20
Using the console to onboard your device and gateway to AWS IoT Core for LoRaWAN	20
Onboard LoRaWAN gateways	21
Onboard LoRaWAN devices	33
Configuring position for LoRaWAN resources	53
How positioning works for LoRaWAN devices	54
Overview of positioning workflow	55
Configuring your resource position	56
Configuring the position of LoRaWAN gateways	56

Configuring position of LoRaWAN devices	59
Managing LoRaWAN gateways	65
LoRa Basics Station software requirement	65
Using qualified gateways from the AWS Partner Device Catalog	66
Using CUPS and LNS protocols	66
Configure beaconing and filtering capabilities of your LoRaWAN gateways	67
Update gateway firmware using CUPS	73
Choosing gateways to receive the LoRaWAN downlink data traffic	88
Managing LoRaWAN devices	90
Device considerations	91
Using devices with gateways qualified for AWS IoT Core for LoRaWAN	91
LoRaWAN version	91
Activation modes	91
Device classes	92
Performing ADR for LoRaWAN devices	92
Managing LoRaWAN device communication	95
Managing LoRaWAN traffic from public LoRaWAN devices networks (Everynet)	104
FUOTA for LoRaWAN devices and multicast groups	116
Prepare devices for multicast and FUOTA configuration	117
Create multicast groups	121
FUOTA for LoRaWAN devices	132
Monitoring LoRaWAN resources with network analyzer	147
Add necessary IAM role for network analyzer	149
Create a network analyzer configuration and add resources	151
Stream trace messages with WebSockets	160
Monitor trace messages in real time	172
Debug your multicast groups and FUOTA tasks using network analyzer	178
AWS IoT Core for LoRaWAN metrics	181
What metrics can I view?	181
How to view summary metrics?	182
LoRaWAN metrics	184
LoRaWAN VPC endpoints	190
Considerations for AWS IoT Wireless VPC endpoints	190
AWS IoT Core for LoRaWAN privatelink architecture	190
AWS IoT Core for LoRaWAN endpoints	191
Onboard control plane endpoint	192

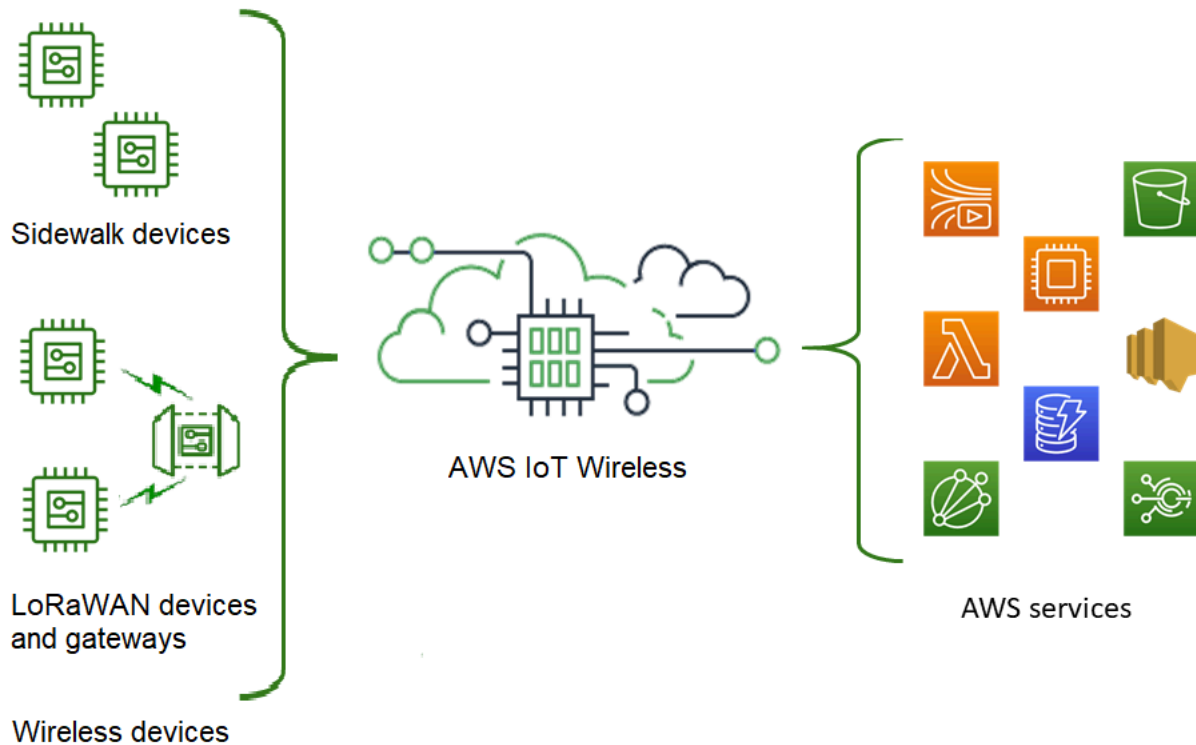
Onboard data plane endpoints	195
AWS IoT Core for Amazon Sidewalk	205
Accessing AWS IoT Core for Amazon Sidewalk	205
AWS IoT Core for Amazon Sidewalk Regions and endpoints	205
AWS IoT Core for Amazon Sidewalk pricing	206
What is AWS IoT Core for Amazon Sidewalk?	206
Features of AWS IoT Core for Amazon Sidewalk	206
What is Amazon Sidewalk?	207
How AWS IoT Core for Amazon Sidewalk works	208
Getting started with AWS IoT Core for Amazon Sidewalk	210
Try the sensor monitoring tutorial	211
Introduction to onboarding your Sidewalk devices	211
Connecting to AWS IoT Core for Amazon Sidewalk	215
Prerequisites	216
Describing your Sidewalk resources	216
Add your Sidewalk device	217
Add a destination for Sidewalk device	227
Connect your Sidewalk device	234
Bulk provisioning Sidewalk devices	237
Amazon Sidewalk bulk provisioning workflow	237
Creating device profiles with factory support	242
Provisioning Sidewalk devices using import tasks	246
Security	259
Data protection	259
Data encryption in AWS IoT Wireless	260
LoRaWAN data and transport security	260
Identity and access management	262
Audience	263
Authenticating with identities	263
Managing access using policies	266
How AWS IoT Wireless works with IAM	269
Identity-based policy examples	276
AWS managed policies	280
Troubleshooting	286
Compliance validation	288
Resilience	289

Infrastructure security	289
Monitoring	291
Monitoring tools	291
How to monitor resources using Amazon CloudWatch	292
Configure logging	292
Create logging role and policy	293
Configure logging for resources	296
Monitor using CloudWatch Logs	309
View log entries	310
Use CloudWatch Insights to filter logs	318
Event notifications	322
How your resources can be notified of events	322
Events and resource types	322
Policy for receiving wireless event notifications	323
Format of MQTT topics for wireless events	323
Pricing for wireless events	327
Enable events for wireless resources	327
Event configurations	327
Prerequisites	328
Enable notifications using the AWS Management Console	328
Enable notifications using the AWS CLI	329
Event notifications for LoRaWAN resources	331
Event types for LoRaWAN resources	332
LoRaWAN join events	332
Connection status events	335
Event notifications for Sidewalk resources	338
Event types for Sidewalk resources	338
Device registration state events	338
Proximity events	341
Message delivery status events	344
AWS IoT Wireless API operations	349
API operations for device profiles	349
List device profiles in your AWS account	349
Delete device profiles from your AWS account	350
API operations for LoRaWAN and Sidewalk devices	351
Associate wireless devices in your AWS account to an IoT thing	351

List wireless devices in your AWS account	352
Delete wireless devices from your AWS account	353
API operations for destinations for wireless devices	353
Get information about your destination	353
Update properties of your destination	354
List destinations in your AWS account	354
Delete destinations from your AWS account	355
API operations for bulk provisioning	355
Get information about your import task	356
Get import task device summary	357
Add devices to import task	357
List import tasks in your AWS account	358
Delete import tasks from your AWS account	359
AWS CloudFormation resources	361
AWS IoT Wireless and AWS CloudFormation templates	361
Learn more about AWS CloudFormation	361
Quotas	362
Tagging your wireless resources	363
Tag basics	363
Create and manage tags	363
Update tags or list tags for resources	364
Tag restrictions and limitations	364
Using tags with IAM policies	365
Document history	368

What is AWS IoT Wireless?

AWS IoT Wireless provides the cloud services that connect your wireless devices to other devices and AWS Cloud services. By connecting your devices to AWS IoT Wireless, you can integrate your devices into AWS IoT-based solutions. Using AWS IoT Wireless, you can onboard both LoRaWAN and Sidewalk devices to AWS IoT. These wireless devices use the Low Power Wide Area Networking (LPWAN) communication protocol to communicate with AWS IoT.



Features of AWS IoT Wireless

AWS IoT Wireless provides the following features:

Onboard LoRaWAN and Sidewalk devices

You can onboard both LoRaWAN and Sidewalk devices to AWS IoT Wireless.

- **AWS IoT Core for LoRaWAN**

To onboard your LoRaWAN devices and gateways to AWS IoT Wireless, use AWS IoT Core for LoRaWAN. It is a fully managed LoRaWAN network server (LNS) that eliminates the need for you

to set up and operate a private LNS. AWS IoT Core for LoRaWAN provides gateway management using the Configuration and Update Server (CUPS) and Firmware Updates Over-The-Air (FUOTA) capabilities. For more information, see [What is AWS IoT Core for LoRaWAN?](#).

- **AWS IoT Core for Amazon Sidewalk**

To onboard your Sidewalk devices to AWS IoT Wireless, you can use the capabilities offered by AWS IoT Core for Amazon Sidewalk. [Amazon Sidewalk](#) is a shared network that connects devices such as Amazon Echo, Ring security cams, outdoor lights, and can support other Sidewalk devices in your community. For more information, see [What is AWS IoT Core for Amazon Sidewalk?](#).

Integration with AWS IoT Core

You can use the following capabilities offered by the AWS IoT Wireless integration with AWS IoT Core:

- **Associate devices with an AWS IoT thing**

You can associate your wireless devices and gateways to an AWS IoT *thing*, which helps you store a representation of your device on the Cloud. You can use things in AWS IoT to more easily search and manage your devices, and access other AWS IoT Core features. For more information, see [Managing devices with AWS IoT](#) in the *AWS IoT Core developer guide*.

- **Use AWS IoT rules to route messages**

You can use the rules feature of AWS IoT to interact with other AWS services and applications. Uplink messages that are sent from your devices to the cloud can be routed to these services and other applications. For more information, see [AWS IoT rules](#) in the *AWS IoT Core developer guide*.

For first-time users of AWS IoT Wireless

If you are a first-time user of AWS IoT Wireless, we recommend that you begin by reading the following sections:

- [What is AWS IoT Core for LoRaWAN?](#)

This section gives an overview of the LoRaWAN technology and how AWS IoT Core for LoRaWAN works. It also provides resources to help you learn more.

- [What is AWS IoT Core for Amazon Sidewalk?](#)

This section gives an overview of the Amazon Sidewalk technology and how AWS IoT Core for Amazon Sidewalk works. It also provides resources to help you learn more.

- [Getting started with AWS IoT Core for Amazon Sidewalk](#)

Read this section to learn about using AWS IoT Core for Amazon Sidewalk and how to onboard your Amazon Sidewalk devices.

- [Connecting gateways and devices to AWS IoT Core for LoRaWAN](#)

Next, you can learn more about how to onboard your LoRaWAN devices by using the console and the API.

Related services

- [Amazon CloudWatch](#)

After you onboard your LoRaWAN or Sidewalk devices to AWS IoT Wireless, you can use Amazon CloudWatch to log and monitor your wireless devices and gateways in real time. To monitor your LoRaWAN devices and gateways, you can also use network analyzer, which reduces the time it takes to set up a connection and start receiving trace messages.

- [AWS IoT Core](#)

You can also use the AWS IoT Core integration to connect to AWS services that can be accessed from the rules engine. For more information, see [AWS services used by the rules engine](#).

Accessing AWS IoT Wireless

You can use the console, the API, or the CLI to onboard both your LoRaWAN and Sidewalk devices.

- **Using the AWS IoT console**

To onboard your wireless devices, use the [AWS IoT Wireless](#) page of the AWS Management Console.

- **Using the AWS IoT Wireless API**

You can onboard both Sidewalk and LoRaWAN devices by using the [AWS IoT Wireless](#) API. The AWS IoT Wireless API that AWS IoT Core is built on is supported by the AWS SDK. For more information, see [AWS SDKs and Toolkits](#).

- **Using the AWS CLI**

You can use the AWS CLI to run commands for onboarding and managing your LoRaWAN and Amazon Sidewalk devices. For more information, see [AWS IoT Wireless CLI reference](#).

Getting started with AWS IoT Wireless

You can get started with AWS IoT Wireless by signing up for an AWS account and following the steps to create an IAM user. After you've signed up, you can then use the AWS Management Console, the AWS IoT Wireless API, or the AWS CLI to onboard your Sidewalk and LoRaWAN devices and gateways. When onboarding your devices, consider how to describe and tag your resources to help you identify them more easily.

The following topics show how to get started with AWS IoT Wireless.

Topics

- [Setting up AWS IoT Wireless](#)
- [Describing your AWS IoT Wireless resources](#)

Setting up AWS IoT Wireless

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including AWS IoT Wireless. You are charged only for the services that you use.

To set up AWS IoT Wireless, perform the steps in the following section:

Topics

- [Set up your AWS account](#)
- [Installing Python and the AWS CLI](#)

Set up your AWS account

Before you use AWS IoT Core for LoRaWAN or AWS IoT Core for Amazon Sidewalk for the first time, complete the following tasks to set up your AWS account.

Topics

- [Sign up for an AWS account](#)
- [Create an IAM user](#)
- [Sign in as an IAM user](#)

Sign up for an 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.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

Create an IAM user

To create an administrator user, choose one of the following options.

Choose one way to manage your administrator	To	By	You can also
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices . For information about best practices , see Security best	Following the instructions in Getting started in the <i>AWS IAM Identity Center User Guide</i> .	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i> .

Choose one way to manage your administrator	To	By	You can also
	practices in IAM in the <i>IAM User Guide</i> .		
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Creating your first IAM admin user and user group in the <i>IAM User Guide</i> .	Configure programmatic access by Managing access keys for IAM users in the <i>IAM User Guide</i> .

Sign in as an IAM user

After you create an IAM user, you can sign in to AWS with your IAM user name and password.

Before you sign in as an IAM user, you can verify the sign-in link for IAM users in the IAM console. On the IAM Dashboard, under IAM users sign-in link, you can see the sign-in link for your AWS account. The URL for your sign-in link contains your AWS account ID without dashes (-).

If you don't want the URL for your sign-in link to contain your AWS account ID, you can create an account alias. For more information, see [Creating, deleting, and listing an AWS account alias](#) in the *IAM User Guide*.

To sign in as an IAM user

1. Sign out of the AWS Management Console.
2. Enter your sign-in link, which includes your AWS account ID (without dashes) or your AWS account alias.

```
https://aws_account_id_or_alias.signin.aws.amazon.com/console
```

3. Enter the IAM user name and password that you just created.

When you're signed in, the navigation bar displays "*your_user_name @ your_aws_account_id*".

Installing Python and the AWS CLI

Before you connect your LoRaWAN or Sidewalk end device, you must set up and install Python, and configure the AWS CLI.

Important

To perform the entire onboarding workflow for provisioning and registering your Sidewalk end device, you must also set up your Sidewalk gateway and the HDK. For instructions, see [Setting up the hardware development kit \(HDK\)](#) and [Setting up a Sidewalk gateway](#) in the *Amazon Sidewalk documentation*.

Topics

- [Install Python and Python3-pip](#)
- [Setting up your AWS CLI](#)

Install Python and Python3-pip

To use the AWS CLI and boto3 as described in the next section, you must use a Python version 3.6 or later. If you want to onboard your end devices using the AWS IoT console, you can skip this section and continue to setting up your AWS account. To check whether you've already installed Python and Python3-pip, run the following commands. If running these commands return the version, it means that Python and Python3-pip have been installed correctly.

```
python3 -V
pip3 --version
```

If this command returns an error, it might be because Python is not installed, or your operating system calls the Python v3.x executable as Python3. In that case, replace all instances of python with python3 when you run the commands. If it still produces an error, either download and run the [Python installer](#), or install Python depending on your operating system as described below.

Windows

On your Windows machine, download Python from the [Python website](#) and then run the installer to install Python on your machine.

Linux

On your Ubuntu machine, run the following sudo command to install Python.

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

macOS

On your Mac machine, use Homebrew to install Python. Homebrew also installs pip, which then points to the installed Python3 version.

```
$ brew install python
```

Setting up your AWS CLI

The following steps show you how to configure your AWS CLI and boto3 (AWS SDK for Python). Before you follow these steps, you must sign up for an AWS account and create an administrative user. For instructions, see [Setting up AWS IoT Wireless](#).

1. Install and configure the AWS CLI

You can use the AWS CLI to programmatically onboard your Sidewalk end devices to AWS IoT Core for Amazon Sidewalk. If you want to onboard your devices using the AWS IoT console, you can skip this section. Open the [AWS IoT Core console](#) and then continue to the next section to get started with connecting your devices to AWS IoT Core for Amazon Sidewalk. For instructions on configuring the AWS CLI, see [Installing and configuring the AWS CLI](#).

2. Install boto3 (AWS SDK for Python)

The following commands show you how to install boto3 (AWS SDK for Python) and the AWS CLI. You'll also install botocore, which is required to run boto3. For detailed instructions, see [Installing Boto3](#) in the *Boto3 Documentation Guide*.

Note

awscli version 1.26.6 requires PyYAML version that's 3.10 or later, but not later than 5.5.

```
python3 -m pip install botocore-version-py3-none-any.whl
python3 -m pip install boto3-version-py3-none-any.whl
```

3. Configure your credentials and default Region

Configure your credentials and default Region in the `~/.aws/credentials` and `~/.aws/config` files. The boto3 library uses these credentials to identify your AWS account and authorize API calls. For configuration instructions, see:

- [Configuration](#) in the *Boto3 Documentation Guide*
- [Configuration and credentials file settings](#) in the *AWS CLI Documentation Guide*

Describing your AWS IoT Wireless resources

Before you get started with onboarding your LoRaWAN or Sidewalk devices, consider the naming convention of your devices, gateways, and destination. AWS IoT Wireless provides several options to help you identify the resources you create. While AWS IoT Wireless 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. To make it more convenient to select, identify, and manage your resources, you can assign a name, add a description, and attach tags and tag values to most AWS IoT Wireless resources.

- [Resource names and description](#)

For devices, gateways, 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](#)

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 and description

AWS IoT Wireless resource support for name

Resource	Name field support	
Destination	Name is unique ID of resource and can't be changed.	
Wireless device	Name is optional descriptor of resource and can be changed.	
LoRaWAN 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 Wireless resource support for AWS tags

Resource	AWS tag support	
Destination	Up to 50 AWS tags can be added to the resource.	
Wireless device	This resource doesn't support AWS tags.	
LoRaWAN 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 in AWS IoT Wireless, see [Tagging your AWS IoT Wireless resources](#).

For more information about tagging and tagging strategies, see [Tag editor](#).

AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN is a fully managed LoRaWAN network server (LNS) that provides gateway management using the Configuration and Update Server (CUPS) and Firmware Updates Over-The-Air (FUOTA) capabilities. You can replace your private LNS with AWS IoT Core for LoRaWAN and connect your Long Range Wide Area Network (LoRaWAN) devices and gateways to AWS IoT Core. By doing so, you'll reduce the maintenance, operational costs, setup time, and overhead costs.

Note

AWS IoT Core for LoRaWAN supports only the IPv4 address format. It doesn't support the IPv6 or the dual stack configuration (IPv4 and IPv6). For more information, see [AWS services that support IPv6](#).

Introduction

LoRaWAN devices are long-range, low-power, battery-operated devices that use the LoRaWAN protocol to operate in a license-free radio spectrum. LoRaWAN is a Low Power Wide Area Network (LPWAN) communication protocol that is built on LoRa. LoRa is the physical layer protocol that enables low power, wide-area communication between devices.

To connect your LoRaWAN devices to AWS IoT, you must use a LoRaWAN gateway. The gateway acts as a bridge to connect your device to AWS IoT Core for LoRaWAN and to exchange messages. AWS IoT Core for LoRaWAN uses the AWS IoT rules engine to route the messages from your LoRaWAN devices to other AWS IoT services.

To reduce development effort and quickly onboard your devices to AWS IoT Core for LoRaWAN, we recommend that you use LoRaWAN-certified end devices. For more information, see the [AWS IoT Core for LoRaWAN product overview](#) page. For information about getting your devices LoRaWAN certified, see [Certifying LoRaWAN products](#).

Accessing AWS IoT Core for LoRaWAN

You can quickly onboard your LoRaWAN devices and gateways to AWS IoT Core for LoRaWAN by using the console or the AWS IoT Wireless API.

Using the console

To onboard your LoRaWAN devices and gateways by using the AWS Management Console, sign in to the AWS Management Console and navigate to the [AWS IoT Core for LoRaWAN](#) page in the AWS IoT console. You can then use the **Intro** section to add your gateways and devices to AWS IoT Core for LoRaWAN. For more information, see [Using the console to onboard your device and gateway to AWS IoT Core for LoRaWAN](#).

Using the API or CLI

You can onboard both LoRaWAN and Sidewalk devices by using the [AWS IoT Wireless](#) API. 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](#).

You can use the AWS CLI to run commands for onboarding and managing your LoRaWAN gateways and devices. For more information, see [AWS IoT Wireless CLI reference](#).

AWS IoT Core for LoRaWAN Regions and endpoints

AWS IoT Core for LoRaWAN provides support for control plane and data plane API endpoints that are specific to your AWS Region. The data plane API endpoints are specific to your AWS account and AWS Region. For more information about the AWS IoT Core for LoRaWAN endpoints, see [AWS IoT Core for LoRaWAN Endpoints](#) in the *AWS General Reference*.

For more secure communication between your devices and AWS IoT, you can connect your devices to AWS IoT Core for LoRaWAN through AWS PrivateLink in your virtual private cloud (VPC), instead of connecting over the public internet. For more information, see [AWS IoT Core for LoRaWAN and interface VPC endpoints \(AWS PrivateLink\)](#).

AWS IoT Core for LoRaWAN has quotas that apply to device data that is transmitted between the devices and the maximum TPS for the AWS IoT Wireless API operations. For more information, see [AWS IoT Core for LoRaWAN quotas](#) in the *AWS General Reference*.

AWS IoT Core for LoRaWAN pricing

If you're a new customer, when you sign up for AWS, you can get started with AWS IoT Core for LoRaWAN for free by using the [AWS Free Tier](#). With AWS IoT Core for LoRaWAN, you only pay for what you use. For more information about general product overview and pricing, see [AWS IoT Core pricing](#).

What is AWS IoT Core for LoRaWAN?

AWS IoT Core for LoRaWAN replaces a private LoRaWAN network server (LNS) by connecting your LoRaWAN devices and gateways to AWS. Using the AWS IoT rules engine, you can route messages received from LoRaWAN devices, where they can be formatted and sent to other AWS IoT services. To secure device communications with AWS IoT, AWS IoT Core for LoRaWAN uses X.509 certificates.

AWS IoT Core for LoRaWAN manages the service and device policies that AWS IoT Core requires to 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.

Features of AWS IoT Core for LoRaWAN

With AWS IoT Core for LoRaWAN, you can:

- Onboard and connect LoRaWAN devices and gateways to AWS IoT without the need to set up and manage a private LNS.
- Connect LoRaWAN devices that comply to 1.0.x or 1.1 LoRaWAN specifications standardized by LoRa Alliance. These devices can operate in class A, class B, or class C mode.
- Use LoRaWAN gateways that support LoRa Basics Station version 2.0.4 or later. All gateways that are qualified for AWS IoT Core for LoRaWAN run a compatible version of LoRa Basics Station.
- Connect your LoRaWAN devices to the cloud using publicly available LoRaWAN networks, which reduces the time to deployment, and eliminates the need for managing a private LoRaWAN network, thereby saving time and cost.
- Monitor signal strength, bandwidth, and spreading factor by using AWS IoT Core for LoRaWAN's adaptive data rate, and optimize the data rate if needed. You can also use network analyzer to monitor your LoRaWAN resources in real-time.
- Update LoRaWAN gateways' firmware using the CUPS service and the firmware of LoRaWAN devices using Firmware Updates Over-The-Air (FUOTA).

The following topics will provide more information about the LoRaWAN technology and AWS IoT Core for LoRaWAN.

Topics

- [What is LoRaWAN?](#)
- [How AWS IoT Core for LoRaWAN works](#)

What is 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."*

LoRa and LoRaWAN

The LoRaWAN protocol is a Low Power Wide Area Networking (LPWAN) communication protocol that functions on LoRa.

LoRaWAN has been recognized as an international standard for low power wide area networking. For more information, see [LoRAWAN formally recognized as ITU international standard](#). The LoRaWAN specification is open so anyone can set up and operate a LoRa network.

LoRa is a wireless audio frequency technology that operates in a license-free radio frequency spectrum. LoRa is a physical layer protocol that uses spread spectrum modulation and supports long-range communication at the cost of a narrow bandwidth. It uses a narrow band waveform with a central frequency to send data, which makes it robust to interference.

Characteristics of LoRaWAN technology

- Long range communication up to 10 miles in line of sight.
- Long battery duration of up to 10 years. For enhanced battery life, you can operate your devices in class A or class B mode, which requires increased downlink latency.
- Low cost for devices and maintenance.
- License-free radio spectrum but region-specific regulations apply.
- Low power but has a limited payload size of 51 bytes to 241 bytes depending on the data rate. The data rate can be 0,3 Kbit/s – 27 Kbit/s data rate with a 222 maximal payload size.

LoRaWAN protocol versions

LoRa Alliance specifies the LoRaWAN protocol using LoRaWAN specification documents. To account for the region-specific regulations, the LoRa Alliance also publishes regional parameter documents. For more information, see [LoRaWAN regional parameters and specifications](#).

The initial release of LoRaWAN is version 1.0. Additional versions released are 1.0.1, 1.0.2, 1.0.3, 1.0.4, and 1.1. Versions 1.0.1-1.0.4 are commonly referred to as 1.0.x.

Learn more about LoRaWAN

The following links contain helpful information about the LoRaWAN technology and about LoRa Basics Station, which is the software that runs on your LoRaWAN gateways for connecting end devices to AWS IoT Core for LoRaWAN.

- [LoRaWAN recognized as ITU International Standard](#)

LoRaWAN has been formally documented as an international standard by ITU for low power wide area networking. The standard is titled Recommendation ITU-T Y.4480 “Low power protocol for wide area wireless networks”.

- [The Things Fundamentals on LoRaWAN](#)

The Things Fundamentals on LoRaWAN contains an introductory video that covers the fundamentals of LoRaWAN and a series of chapters that'll help you learn about LoRa and LoRaWAN.

- [What is LoRaWAN](#)

LoRa Alliance provides a technical overview of LoRa and LoRaWAN, including a summary of the LoRaWAN specifications in different Regions.

- [LoRa Basics Station](#)

Semtech Corporation provides helpful concepts about LoRa basics for gateways and end nodes. LoRa Basics Station, an open source software that runs on your LoRaWAN gateway, is maintained and distributed through Semtech Corporation's [GitHub](#) repository. You can also learn about the LNS and CUPS protocols that describe how to exchange LoRaWAN data and perform configuration updates.

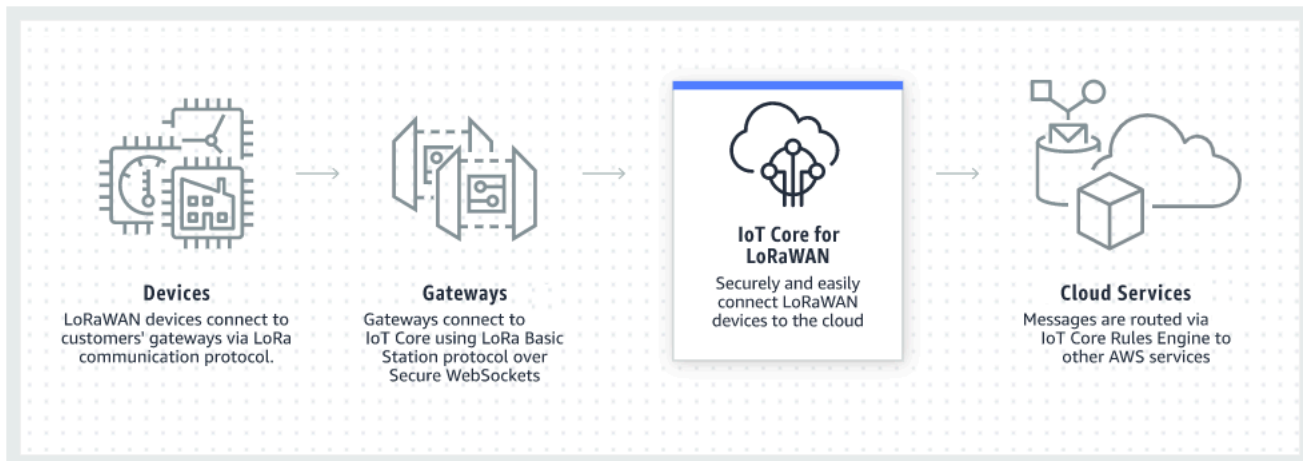
- [LoRaWAN regional parameters and specifications](#)

RP002-1.0.2 document includes support for all versions of the LoRaWAN Layer 2 specification. It includes information about the LoRaWAN specifications and regional parameters, and the different LoRaWAN versions.

How AWS IoT Core for LoRaWAN works

The LoRaWAN network architecture is deployed in a star of stars topology in which gateways relay information between end devices and the LoRaWAN network server (LNS). The following shows

how a LoRaWAN device interacts with AWS IoT Core for LoRaWAN. It also shows how AWS IoT Core for LoRaWAN acts as an LNS and communicates with other AWS services in the AWS Cloud.



LoRaWAN devices communicate with AWS IoT Core through LoRaWAN gateways. 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.

Get started using AWS IoT Core for LoRaWAN

The following steps show an overview of how you can get started using AWS IoT Core for LoRaWAN.

1. 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. For more information, see [Using qualified gateways from the AWS Partner Device Catalog](#).

2. Add your wireless devices and LoRaWAN gateways to AWS IoT Core for LoRaWAN.

[Connecting gateways and devices to AWS IoT Core for LoRaWAN](#) gives you information about how to describe your resources and 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.

3. Complete your AWS IoT Core for LoRaWAN solution.

Start with [our sample AWS IoT Core for LoRaWAN solution](#) and make it yours.

AWS IoT Core for LoRaWAN resources

The following resources will help you learn more about AWS IoT Core for LoRaWAN and how to get started.

- [Getting Started with AWS IoT Core for LoRaWAN](#)

The following video describes how AWS IoT Core for LoRaWAN works and walks you through the process of adding LoRaWAN gateways from the AWS Management Console.

- [AWS IoT Core for LoRaWAN workshop](#)

The workshop covers fundamentals of LoRaWAN technology and its implementation with AWS IoT Core for LoRaWAN. You can also use the workshop to walk through labs that show how to connect your gateway and device to AWS IoT Core for LoRaWAN for building a sample IoT solution.

- [Implementing Low-Power Wide-Area Network \(LPWAN\) Solutions with AWS IoT](#)

This paper provides you with a decision framework to help you decide if LPWAN is the right choice for your IoT use case, provides an overview of LPWAN connectivity technologies and their capabilities, and provides implementation guidelines.

Connecting gateways and devices to AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN helps you connect and manage wireless LoRaWAN (low-power long-range Wide Area Network) devices and replaces the need for you to develop and operate an LNS. Long range WAN (LoRaWAN) devices and gateways can connect to AWS IoT Core by using AWS IoT Core for LoRaWAN.

Naming conventions for your devices, gateways, profiles, and destinations

Before you get started with AWS IoT Core for LoRaWAN and creating the resources, consider the naming convention of your devices, gateways, and destination.

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 [Describing your AWS IoT Wireless resources](#).

Mapping of 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 onboard your device and gateway to AWS IoT Core for LoRaWAN

You can use the console interface or the API to add your LoRaWAN gateway and devices. If you're using AWS IoT Core for LoRaWAN for the first time, we recommend that you use the console. 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.

Note

If you're using a public network to connect your LoRaWAN devices to the cloud, you can skip onboarding your gateways. For more information, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

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 following topics describe how you can describe your AWS IoT Core for LoRaWAN resources and use the console or the API to add your gateways and devices.

Topics

- [Onboard your gateways to AWS IoT Core for LoRaWAN](#)
- [Onboard your devices to AWS IoT Core for LoRaWAN](#)

Onboard your gateways to AWS IoT Core for LoRaWAN

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 console.

Note

If you're using a public network to connect your LoRaWAN devices to the cloud, you can skip onboarding your gateways. For more information, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

Before onboarding your gateway

Before you onboard your gateway to AWS IoT Core for LoRaWAN, we recommend that you:

- Use gateways that are qualified for use with AWS IoT Core for LoRaWAN. These gateways connect to AWS IoT Core without any additional configuration settings and have a version 2.0.4 or later of the [LoRa Basics Station](#) software running on them. For more information, see [Managing gateways with AWS IoT Wireless](#).
- Consider the naming convention of the resources that you create so that you can more easily manage them. For more information, see [Describing your AWS IoT Wireless resources](#).

- Have the configuration parameters that are unique to each gateway ready to enter in advance, which makes entering the data into the console go more smoothly. The wireless gateway configuration parameters that AWS IoT requires to communicate with and manage the gateway include the gateway's EUI and its LoRa frequency band.

For onboarding your gateways to AWS IoT Core for LoRaWAN:

- [Consider frequency band selection and add necessary IAM role](#)
- [Add a gateway to AWS IoT Core for LoRaWAN](#)
- [Connect your LoRaWAN gateway and verify its connection status](#)

Consider frequency band selection and add necessary IAM role

Before you add your gateway to AWS IoT Core for LoRaWAN, we recommend that you consider the frequency band in which your gateway will be operating and add the necessary IAM role for connecting your gateway to AWS IoT Core for LoRaWAN.

Note

If you're adding your gateway using the console, click **Create role** in the console to create the necessary IAM role so you can then skip these steps. You need to perform these steps only if you're using the CLI to create the gateway.

Consider selection of LoRa frequency bands for your gateways and device connection

AWS IoT Core for LoRaWAN supports EU863-870, US902-928, AU915, and AS923-1 frequency bands, which you can use to connect your gateways and devices that are physically present in countries that support the frequency ranges and characteristics of these bands. The EU863-870 and US902-928 bands are commonly used in Europe and North America, respectively. The AS923-1 band is commonly used in Australia, New Zealand, Japan, and Singapore among other countries. The AU915 is used in Australia and Argentina among other countries. For more information about which frequency band to use in your region or country, see [LoRaWAN® Regional Parameters](#).

LoRa Alliance publishes LoRaWAN specifications and regional parameter documents that are available for download from the LoRa Alliance website. The LoRa Alliance regional parameters help companies decide which frequency band to use in their region or country. AWS IoT Core for LoRaWAN's frequency band implementation follows the recommendation in the regional

parameters specification document. These regional parameters are grouped into a set of radio parameters, along with a frequency allocation that is adapted to the Industrial, Scientific, and Medical (ISM) band. We recommend that you work with the compliance teams to ensure that you meet any applicable regulatory requirements.

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.

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 a gateway to AWS IoT Core for LoRaWAN

You can add your gateway to AWS IoT Core for LoRaWAN by using the console or the CLI.

Before adding your gateway, we recommend that you consider the factors mentioned in the **Before onboarding your gateway** section of [Onboard your gateways to AWS IoT Core for LoRaWAN](#).

If you're adding your gateway for the first time, we recommend that you use the console. If you want to add your gateway by using the CLI instead, you must have already created the necessary IAM role so that the gateway can connect with AWS IoT Core for LoRaWAN. For information about how to create the role, see [Add an IAM role to allow the Configuration and Update Server \(CUPS\) to manage gateway credentials](#).

Add a gateway using the console

Navigate to the [AWS IoT Core for LoRaWAN Intro](#) page of the AWS IoT console and choose **Get started**, and then choose **Add gateway**. If you've already added a gateway, choose **View gateway** to view the gateway that you added. If you would like to add more gateways, choose **Add gateway**.

1. Provide gateway details and frequency band information

Use the **Gateway details** section to provide information about the device configuration data such as the Gateway's EUI and the frequency band configuration.

- **Gateway's EUI**

The EUI (Extended Unique Identifier) of the individual gateway device. The EUI is a 16-digit alphanumeric code, such as `c0ee40ffff29df10`, that uniquely identifies a gateway in your LoRaWAN network. This information is specific to your gateway model and you can find it on your gateway device or in its user manual.

 **Note**

The Gateway's EUI is different from the Wi-Fi MAC address that you may see printed on your gateway device. The EUI follows a EUI-64 standard that uniquely identifies your gateway and therefore cannot be reused in other AWS accounts and regions.

- **Frequency band (RFRegion)**

The gateway's frequency band. You can choose from US915, EU868, AU915, or AS923-1, depending on what your gateway supports and which country or region the gateway is physically connecting from. For more information about the bands, see [Consider selection of LoRa frequency bands for your gateways and device connection](#).

2. Specify your wireless gateway configuration data (optional)

These fields are optional and you can use them to provide additional information about the gateway and its configuration.

- **Name, Description, and Tags for your gateway**

The information in these optional fields comes from how you organize and describe the elements in your wireless system. You can assign a **Name** to the gateway, use the **Description** field to provide information about the gateway, and use **Tags** to add key-value pairs of

metadata about the gateway. For more information on naming and describing your resources, see [Describing your AWS IoT Wireless resources](#).

- **LoRaWAN configuration using subbands and filters**

Optionally, you can also specify LoRaWAN configuration data such as the subbands that you want to use and filters that can control the flow of traffic. For this tutorial, you can skip these fields. For more information, see [Configure your gateway's subbands and filtering capabilities](#).

3. Associate an AWS IoT thing with the gateway

Specify whether to create an AWS IoT thing and associate it with the gateway. Things in AWS IoT can make it easier to search and manage your devices. Associating a thing with your gateway lets the gateway access other AWS IoT Core features.

4. Create and download the gateway certificate

To authenticate your gateway so that it can securely communicate with AWS IoT, your LoRaWAN gateway must present a private key and certificate to AWS IoT Core for LoRaWAN. Create a **Gateway certificate** so that AWS IoT can verify your gateway's identity by using the X.509 Standard.

Click the **Create certificate** button and download the certificate files. You'll use them later to configure your gateway.

5. Copy the CUPS and LNS endpoints and download certificates

Your LoRaWAN gateway must connect to a CUPS or LNS endpoint when establishing a connection to AWS IoT Core for LoRaWAN. We recommend that you use the CUPS endpoint as it also provides configuration management. To verify the authenticity of AWS IoT Core for LoRaWAN endpoints, your gateway will use a trust certificate for each of the CUPS and LNS endpoints,

Click the **Copy** button to copy the CUPS and LNS endpoints. You'll need this information later to configure your gateway. Then click the **Download server trust certificates** button to download the trust certificates for the CUPS and LNS endpoints.

6. Create the IAM role for the gateway permissions

You need to add an IAM role that allows the Configuration and Update Server (CUPS) to manage gateway credentials.

Note

In this step, you create the **IoTWirelessGatewayCertManager** role. If you have already created this role, you can skip this step. 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.

To create the **IoTWirelessGatewayCertManager** IAM role for your account, click the **Create role** button. If the role already exists, select it from the dropdown list.

Click **Submit** to complete the gateway creation.

Add a gateway by using the API

Note

If you're adding a gateway for the first time by using the API or CLI, you must add the **IoTWirelessGatewayCertManager** IAM role so that the gateway can connect with AWS IoT Core for LoRaWAN. For information about how to create the role, see the following section [Add an IAM role to allow the Configuration and Update Server \(CUPS\) to manage gateway credentials](#).

The following sections show how to add a gateway using the AWS IoT Wireless API operations or the AWS CLI. You first add your gateway and then associate a certificate with the gateway. You can also use the additional API operations, such as to update an existing gateway.

Topics

- [How to add your gateway](#)
- [Associate a certificate with your gateway](#)
- [Additional API operations](#)

How to add your gateway

You can use the AWS CLI to create a wireless gateway by using the [CreateWirelessGateway](#) API operation or the [create-wireless-gateway](#) CLI command to add your wireless gateway.

Note

If your gateway is communicating with class B LoRaWAN devices, you can also specify certain beaconing parameters when adding the gateway using the `CreateWirelessGateway` API or the `create-wireless-gateway` CLI command. For more information, see [Configuring your gateways to send beacons to class B devices](#).

The following example creates a wireless LoRaWAN device gateway. You can also provide an `input.json` file that will contain additional details such as the gateway certificate and provisioning credentials.

Note

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.

```
aws iotwireless create-wireless-gateway \  
  --lorawan GatewayEui="a1b2c3d4567890ab",RfRegion="US915" \  
  --name "myFirstLoRaWANGateway" \  
  --description "Using my first LoRaWAN gateway" \  
  --cli-input-json file://input.json
```

Associate a certificate with your gateway

After you add your gateway to AWS IoT Wireless, it must be associated with a certificate to connect to the CUPS endpoint. To connect to the endpoint, your gateway running LoRa Basics Station requires the following files:

- `cups.crt` - The gateway's CUPS certificate that it uses to connect to the CUPS endpoint.
- `cups.key` - Private key corresponding to the certificate.
- `cups.trust` - The trust certificate of the CUPS endpoint.
- `cups.uri` - The CUPS endpoint URI.

The following steps show you how to generate a certificate and associate it with your gateway.

Topics

- [Step 1: Generating a gateway certificate](#)
- [Step 2: Obtaining server trust certificate and CUPS endpoint](#)
- [Step 3: Associate the certificate with your gateway](#)

Step 1: Generating a gateway certificate

To generate a certificate for your gateway, use the AWS IoT API Reference API action, [CreateKeysAndCertificate](#), or the AWS CLI command, [create-keys-and-certificate](#) CLI command.

The following command shows an example of generating the certificate, `cups.crt`, and the private key, `cups.key`.

```
aws iot create-keys-and-certificate \  
  --set-as-active --certificate-pem-outfile "cups.crt" \  
  --private-key-outfile "cups.key"
```

Running this command generates the certificate and private key, and a certificate ID. The following example shows an output of running this command.

```
{  
  "certificateArn": "arn:aws:iot:us-  
east-1:123456789012:cert/  
abc1234d55ef32101a34434bb123cba2a011b2cdefa6bb5cee1a221b4567ab12",  
  "certificateId":  
  "abc1234d55ef32101a34434bb123cba2a011b2cdefa6bb5cee1a221b4567ab12",  
  "certificatePem": "-----BEGIN CERTIFICATE-----\n..\n-----END CERTIFICATE-----\n",  
  "KeyPair": {  
    "PublicKey": "-----BEGIN PUBLIC KEY -----\n..\n-----END PUBLIC KEY-----  
\n",  
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\n..\nEND RSA PRIVATE KEY-----  
\n"  
  }  
}
```

Store the certificate ID temporarily, as it will be used in the subsequent step to associate your certificate with the gateway.

Note

You must securely store the private key, `cups.key`. If you misplace the private key, rerun the `create-keys-and-certificate` command to generate another certificate.

Step 2: Obtaining server trust certificate and CUPS endpoint

Now that you've generated the certificate and private key, use the [GetServiceEndpoint](#) API action or the [get-service-endpoint](#) CLI command to obtain the server trust certificate, `cups.trust` and the endpoint URI, `cups.uri`.

The following command shows an example of obtaining the server trust certificate and the endpoint URI. When running the command, set the `service-type` parameter to CUPS.

```
aws iotwireless get-service-endpoint --service-type CUPS
```

The following shows an output of running the command.

```
{
  "ServiceType": "CUPS",
  "ServiceEndpoint": "https://ABCDEFGHIJKLMN.cups.lorawan.us-east-1.amazonaws.com:443",
  "ServerTrust": "-----BEGIN CERTIFICATE-----\n..\n-----END CERTIFICATE-----\n"
}
```

The `ServiceEndpoint` obtained from the response corresponds to the CUPS endpoint, `cups.uri`.

Note

Store the `ServerTrust` certificate in a `.pem` file with the `\n` replaced by new lines.

Step 3: Associate the certificate with your gateway

You must associate the gateway's certificate that you generated with the gateway that you added. AWS IoT Core for LoRaWAN will use this information to identify the certificate that the gateway will use to connect to the CUPS endpoint.

To associate the certificate with your gateway, use the [AssociateWirelessGatewaywithCertificate](#) API action or the [associate-wireless-gateway-with-certificate](#) CLI command.

The following command shows an example of associating a certificate with your gateway.

```
aws iotwireless associate-wireless-gateway-with-certificate \  
  --id <WirelessGatewayId> \  
  --iot-certificate-id <CertificateId>
```

Running this command returns the `IotCertificateId`, which is the ID of the certificate that you associated with the gateway. The following shows an output of running the command, where the `IotCertificateId` is the ID of the certificate, such as `abc1234d55ef32101a34434bb123cba2a011b2cdefa6bb5cee1a221b4567ab12`.

```
{  
  "IotCertificateId": "<CertificateId>"  
}
```

Additional API operations

You can use the following API actions to perform the tasks associated with adding, updating, or deleting a LoRaWAN gateway.

AWS IoT Wireless API actions for AWS IoT Core for LoRaWAN gateways

- [GetWirelessGateway](#)
- [ListWirelessGateways](#)
- [UpdateWirelessGateway](#)
- [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](#).

For information about the CLIs that you can use, see [AWS CLI reference](#).

Connect your LoRaWAN gateway and verify its connection status

Before you can check the gateway connection status, you must have already added your gateway and connected it to AWS IoT Core for LoRaWAN. For information about how to add your gateway, see [Add a gateway to AWS IoT Core for LoRaWAN](#).

Connect your gateway to AWS IoT Core for LoRaWAN

After you've added your gateway, connect to the configuration interface of your gateway to enter the configuration information and trust certificates.

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.

Gateways qualified for use with AWS IoT Core for LoRaWAN

For instructions on how to configure your LoRaWAN gateway, refer to the [configure gateway device](#) section of the AWS IoT Core for LoRaWAN workshop. Here, you'll find information about instructions for connecting gateways that are qualified for use with AWS IoT Core for LoRaWAN.

Gateways that support CUPS protocol

The following instructions show how you can connect your gateways that support the CUPS protocol.

1. Upload the following files that you obtained when adding your gateway.
 - Gateway device certificate and private key files.
 - Trust certificate file for CUPS endpoint, `cups.trust`.
2. Specify the CUPS endpoint URL that you obtained previously. The endpoint will be of the format `prefix.cups.lorawan.region.amazonaws.com:443`.

For details about how to obtain this information, see [Add a gateway to AWS IoT Core for LoRaWAN](#).

Gateways that support LNS protocol

The following instructions show how you can connect your gateways that support the LNS protocol.

1. Upload the following files that you obtained when adding your gateway.
 - Gateway device certificate and private key files.
 - Trust certificate file for LNS endpoint, `lns.trust`.
2. Specify the LNS endpoint URL that you obtained previously. The endpoint will be of the format `https://prefix.lns.lorawan.region.amazonaws.com:443`.

For details about how to obtain this information, see [Add a gateway to AWS IoT Core for LoRaWAN](#).

After that you've connected your gateway to AWS IoT Core for LoRaWAN, you can check the status of your connection and get information about when the last uplink was received by using the console or the API.

Check gateway connection status using the console

To check the connection status using the console, navigate to the [Gateways](#) page of the AWS IoT console and choose the gateway you've added. In the **LoRaWAN specific details** section of the Gateway details page, you'll see the connection status and the date and time the last uplink was received.

Check gateway connection status using the API

To check the connection status using the API, use the `GetWirelessGatewayStatistics` API. This API doesn't have a request body and only contains a response body that shows whether the gateway is connected and when the last uplink was received.

```
HTTP/1.1 200
Content-type: application/json

{
  "ConnectionStatus": "Connected",
  "LastUplinkReceivedAt": "2021-03-24T23:13:08.476015749Z",
  "WirelessGatewayId": "30cbdcf3-86de-4291-bfab-5bfa2b12bad5"
}
```

Onboard your devices to AWS IoT Core for LoRaWAN

After you have onboarded your gateway to AWS IoT Core for LoRaWAN and verified its connection status, you can onboard your wireless devices. For information about how to onboard your gateways, see [Onboard your gateways to AWS IoT Core for LoRaWAN](#).

LoRaWAN devices use a LoRaWAN protocol to exchange data with cloud-hosted applications. AWS IoT Core for LoRaWAN supports devices that comply to 1.0.x or 1.1 LoRaWAN specifications standardized by LoRa Alliance.

A LoRaWAN device typically contains one or more sensors and actors. The devices send uplink telemetry data through LoRaWAN gateways to AWS IoT Core for LoRaWAN. Cloud-hosted

applications can control the sensors by sending downlink commands to LoRaWAN devices through LoRaWAN gateways.

Before onboarding your wireless device

Before you onboard your wireless device to AWS IoT Core for LoRaWAN, you need to have the following information ready in advance:

- **LoRaWAN specification and wireless device configuration**

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. For the complete listing of its specifications and configuration parameters, see each device's documentation.

- **Device name and description (optional)**

The information in these optional fields comes from how you organize and describe the elements in your wireless system. For more information about naming and describing your resources, see [Describing your AWS IoT Wireless resources](#).

- **Device and service profiles**

Have some wireless device configuration parameters ready that are shared by many devices and can be stored in AWS IoT Core for LoRaWAN as device and service profiles. 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. For more information, see [Add profiles to AWS IoT Core for LoRaWAN](#).

- **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'll create to use with the device's messages and assign it to the device.

To onboard your wireless device to AWS IoT Core for LoRaWAN

- [Add your wireless device to AWS IoT Core for LoRaWAN](#)
- [Add profiles to AWS IoT Core for LoRaWAN](#)

- [Add destinations to AWS IoT Core for LoRaWAN](#)
- [Create rules to process LoRaWAN device messages](#)
- [Connect your LoRaWAN device and verify its connection status](#)

Add your wireless device to AWS IoT Core for LoRaWAN

If you're adding your wireless device for the first time, we recommend that you use the console. Navigate to the [AWS IoT Core for LoRaWAN Intro](#) page of the AWS IoT console, choose **Get started**, and then choose **Add device**. If you've already added a device, choose **View device** to view the gateway that you added. If you would like to add more devices, choose **Add device**.

Alternatively, you can also add wireless devices from the [Devices](#) page of the AWS IoT console.

Add your wireless device specification to AWS IoT Core for LoRaWAN using the console

Choose a **Wireless device specification** based on your activation method and the LoRaWAN version. Once selected, your data is encrypted with a key that AWS owns and manages for you.

OTAA and ABP activation modes

Before your LoRaWAN device can send uplink data, you must complete a process called *activation* or *join procedure*. To activate your device, you can either use OTAA (Over the air activation) or ABP (Activation by personalization).

ABP doesn't require a join procedure and uses static keys. When you use OTAA, your LoRaWAN device sends a join request and the Network Server can allow the request. We recommend that you use OTAA to activate your device because new session keys are generated for each activation, which makes it more secure.

LoRaWAN version

When you use OTAA, your LoRaWAN device and cloud-hosted applications share the root keys. These root keys depend on whether you're using version v1.0.x or v1.1. v1.0.x has only one root key, **AppKey** (Application Key) whereas v1.1 has two root keys, **AppKey** (Application Key) and **NwkKey** (Network Key). The session keys are derived based on the root keys for each activation. Both the **NwkKey** and **AppKey** are 32-digit hexadecimal values that your wireless vendor provided.

Wireless Device EUIs

After you select the **Wireless device specification**, you see the EUI (Extended Unique Identifier) parameters for the wireless device displayed on the console. You can find this information from the documentation for the device or the wireless vendor.

- **DevEUI:** 16-digit hexadecimal value that is unique to your device and found on the device label or its documentation.
- **AppEUI:** 16-digit hexadecimal value that is unique to the join server and found in the device documentation. In LoRaWAN version v1.1, the **AppEUI** is called as **JoinEUI**.

For more information about the unique identifiers, session keys, and root keys, refer to the [LoRa Alliance](#) documentation.

Add your wireless device specification to AWS IoT Core for LoRaWAN by using the API

If you're adding a wireless device using the API, you must create your device profile and service profile first before creating the wireless device. You'll use the device profile and service profile ID when creating the wireless device. For information about how to create these profiles using the API, see [Add a device profile by using the API](#).

The following lists describe the API actions that perform the tasks associated with adding, updating, or deleting a service profile.

AWS IoT Wireless API actions for service profiles

- [CreateWirelessDevice](#)
- [GetWirelessDevice](#)
- [ListWirelessDevices](#)
- [UpdateWirelessDevice](#)
- [DeleteWirelessDevice](#)

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](#).

How to use the AWS CLI to create a wireless device

You can use the AWS CLI to create a wireless device by using the [create-wireless-device](#) command. The following example creates a wireless device by using an input.json file to input the parameters.

Note

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.

Contents of input.json

```
{
  "Description": "My LoRaWAN wireless device",
  "DestinationName": "IoTWirelessDestination",
  "LoRaWAN": {
    "DeviceProfileId": "ab0c23d3-b001-45ef-6a01-2bc3de4f5333",
    "ServiceProfileId": "fe98dc76-cd12-001e-2d34-5550432da100",
    "OtaaV1_1": {
      "AppKey": "3f4ca100e2fc675ea123f4eb12c4a012",
      "JoinEui": "b4c231a359bc2e3d",
      "NwkKey": "01c3f004a2d6efffe32c4eda14bcd2b4"
    },
    "DevEui": "ac12efc654d23fc2"
  },
  "Name": "SampleIoTWirelessThing",
  "Type": LoRaWAN
}
```

You can provide this file as input to the `create-wireless-device` command.

```
aws iotwireless create-wireless-device \
  --cli-input-json file://input.json
```

For information about the CLIs that you can use, see [AWS CLI reference](#)

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.

Add device profiles

Device profiles define the device capabilities and boot parameters that the network server uses to set the LoRaWAN radio access service. It includes selection of parameters such as LoRa frequency band, LoRa regional parameters version, and MAC version of the device. To learn about the different frequency bands, see [Consider selection of LoRa frequency bands for your gateways and device connection](#).

Add a device profile by using the console

If you're adding a wireless device by using the console as described in [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#), after you've added the wireless device specification, you can add your device profile. Alternatively, you can also add wireless devices from the [Profiles](#) page of the AWS IoT console on the **LoRaWAN** tab.

You can choose from default device profiles or create a new device profile. We recommend that you use the default device profiles. If your application requires you to create a device profile, provide a **Device profile name**, select the **Frequency band (RfRegion)** that you're using for the device and gateway, and keep the other settings to the default values, unless specified otherwise in the device documentation.

Add a device profile by using the API

If you're adding a wireless device by using the API, you must create your device profile before creating the wireless device.

The following lists describe the API actions that perform the tasks associated with adding, updating, or deleting a service profile.

AWS IoT Wireless API actions for service profiles

- [CreateDeviceProfile](#)
- [GetDeviceProfile](#)
- [ListDeviceProfiles](#)
- [UpdateDeviceProfile](#)
- [DeleteDeviceProfile](#)

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](#).

How to use the AWS CLI to create a device profile

You can use the AWS CLI to create a device profile by using the [create-device-profile](#) command. The following example creates a device profile.

```
aws iotwireless create-device-profile
```

Running this command automatically creates a device profile with an ID that you can use when creating the wireless device. You can now create the service profile using the following API and then create the wireless device by using the device and service profiles.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

For information about the CLIs that you can use, see [AWS CLI reference](#)

Add service profiles

Service profiles describe the communication parameters the device needs to communicate with the application server.

Note

When creating a service profile, you can specify that you want to use the public network instead of your own private LoRaWAN gateway. For more information, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

Add a service profile using the console

If you're adding a wireless device using the console as described in [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#), after you've added the device profile, you can add your service profile. Alternatively, you can also add wireless devices from the [Profiles](#) page of the AWS IoT console on the **LoRaWAN** tab.

We recommend that you leave the setting **AddGWMetaData** enabled so that you'll receive additional gateway metadata for each payload, such as RSSI and SNR for the data transmission.

Add a service profile using the API

If you're adding a wireless device using the API, you must first create your service profile before creating the wireless device.

The following lists describe the API actions that perform the tasks associated with adding, updating, or deleting a service profile.

AWS IoT Wireless API actions for service profiles

- [CreateServiceProfile](#)
- [GetServiceProfile](#)
- [ListServiceProfiles](#)
- [UpdateServiceProfile](#)
- [DeleteServiceProfile](#)

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](#).

How to use the AWS CLI to create a service profile

You can use the AWS CLI to create a service by using the [create-service-profile](#) command. The following example creates a service profile.

```
aws iotwireless create-service-profile
```

Running this command automatically creates a service profile with an ID that you can use when creating the wireless device. You can now create the wireless device by using the device and service profiles.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ServiceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

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.

If you're adding your destination for the first time, we recommend that you use the console.

Add a destination using the console

If you're adding a wireless device using the console as described in [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#), after you've already added the wireless device specification and profiles to AWS IoT Core for LoRaWAN as described previously, you can go ahead and add a destination.

Alternatively, you can also add an AWS IoT Core for LoRaWAN destination from the [Destinations](#) page of the AWS IoT console.

To process a device's data, specify the following fields when creating an AWS IoT Core for LoRaWAN destination, and then choose **Add destination**.

- **Destination details**

Enter a **Destination name** and an optional description for your destination.

- **Rule name**

The AWS IoT rule that is configured to evaluate messages sent by your device and process the device's data. The rule name will be mapped to your destination. The destination requires the rule to process the messages that it receives. You can choose for the messages to be processed by either invoking an AWS IoT rule or by publishing to the AWS IoT message broker.

- If you choose **Enter a rule name**, enter a name, and then choose **Copy** to copy the rule name that you'll enter when creating the AWS IoT rule. You can either choose **Create rule** to create the rule now or navigate to the [Rules](#) Hub of the AWS IoT console and create a rule with that name.

You can also enter a rule and use the **Advanced** setting to specify a topic name. The topic name is provided during rule invocation and is accessed by using the topic expression inside the rule. For more information about AWS IoT rules, see <https://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html>.

- If you choose **Publish to AWS IoT message broker**, enter a topic name. You can then copy the MQTT topic name and multiple subscribers can subscribe to this topic to receive messages

published to that topic. For more information, see <https://docs.aws.amazon.com/iot/latest/developerguide/topics.html>.

For more information about AWS IoT rules for destinations, see [Create rules to process LoRaWAN device messages](#).

- **Role name**

The IAM role that grants the device's data permission to access the rule named in **Rule name**. In the console, you can create a new service role or select an existing service role. If you're creating a new service role, you can either enter a role name (for example, **IoTWirelessDestinationRole**), or leave it blank for AWS IoT Core for LoRaWAN to generate a new role name. AWS IoT Core for LoRaWAN will then automatically create the IAM role with the appropriate permissions on your behalf.

For more information about IAM roles, see [Using IAM roles](#).

Add a destination by using the API

If you want to add a destination using the CLI instead, you must have already created the rule and IAM role for your destination. For more information about the details that a destination requires in the role, see [Create an IAM role for your destinations](#).

The following list contains the API actions that perform the tasks associated with adding, updating, or deleting a destination.

AWS IoT Wireless API actions for destinations

- [CreateDestination](#)
- [GetDestination](#)
- [ListDestinations](#)
- [UpdateDestination](#)
- [DeleteDestination](#)

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](#).

How to use the AWS CLI to add a destination

You can use the AWS CLI to add a destination by using the [create-destination](#) command. The following example shows how to create a destination by entering a rule name by using `RuleName` as the value for the `expression-type` parameter. If you want to specify a topic name for publishing or subscribing to the message broker, change the `expression-type` parameter's value to `MqttTopicId`.

```
aws iotwireless create-destination \  
  --name IoTWirelessDestination \  
  --expression-type RuleName \  
  --expression IoTWirelessRule \  
  --role-arn arn:aws:iam::123456789012:role/IoTWirelessDestinationRole
```

Running this command creates a destination with the specified destination name, rule name, and role name. For information about rule and role names for destinations, see [Create rules to process LoRaWAN device messages](#) and [Create an IAM role for your destinations](#).

For information about the CLIs that you can use, see [AWS CLI reference](#).

Create an IAM role 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 must define it so that it will appear in the list of roles.

When you use the console to add a destination, AWS IoT Core for LoRaWAN automatically creates an IAM role for you, as described previously in this topic. When you add a destination using the API or CLI, you must create the IAM role for your destination.

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

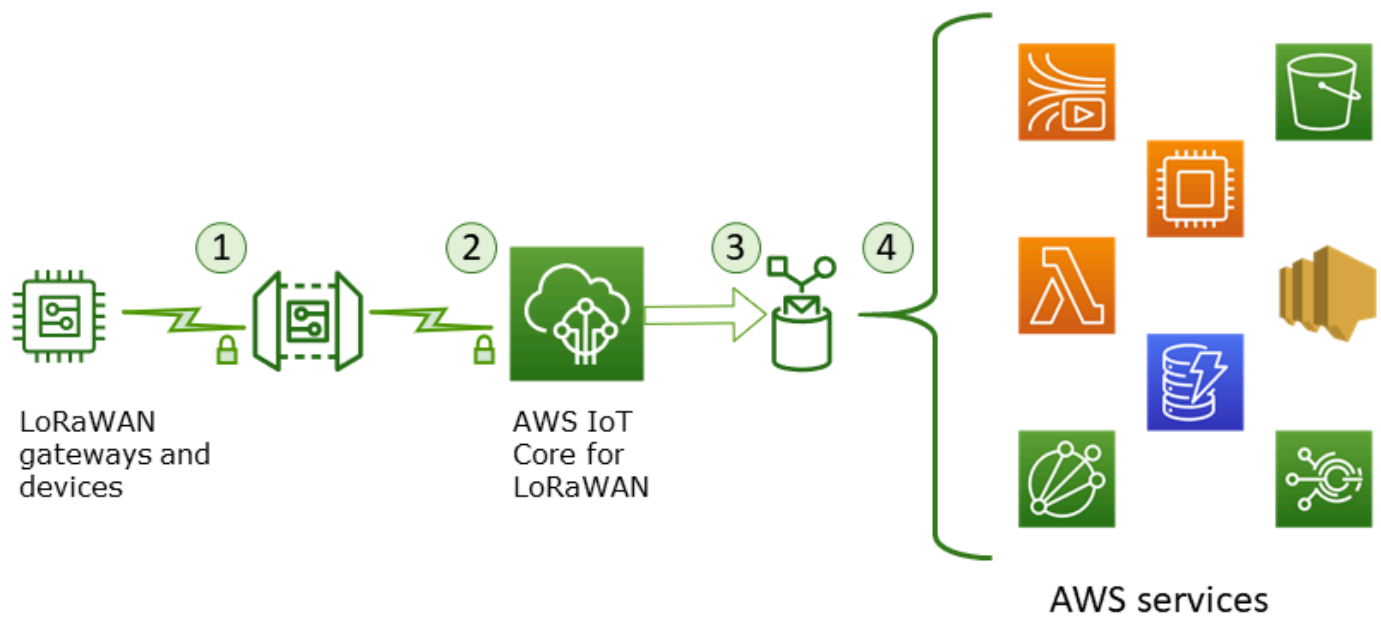
AWS IoT rules send device messages to other services. AWS IoT rules can also process the binary messages received from a LoRaWAN device to convert the messages to other formats that can make them easier for other services to use.

[AWS IoT Core for LoRaWAN destinations](#) 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](#) 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 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 devices

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
    }
) as decodingoutput
```

For more information on using the SELECT AND WHERE clauses, see [AWS IoT SQL reference](#).

For information about AWS IoT rules and how to create and use them, see [AWS IoT rules](#) and [AWS IoT rules tutorials](#).

For information about creating and using AWS IoT Core for LoRaWAN destinations, see [Add destinations to AWS IoT Core for LoRaWAN](#).

For information about using binary message payloads in a rule, see [Binary payloads](#).

For more information about the data security and encryption used to protect the message payload on its journey, see [Data protection in AWS IoT Wireless](#).

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](#).

Connect your LoRaWAN device and verify its connection status

Before you can check the device connection status, you must have already added your device and connected it to AWS IoT Core for LoRaWAN. For information about how to add your device, see [Add your wireless device to AWS IoT Core for LoRaWAN](#).

After you've added your device, refer to your device's user manual to learn how to initiate sending an uplink message from your LoRaWAN device.

Wireless device destination payload

The following code shows the payload received at the destination for your wireless device. It shows a sample payload when using your own private LoRaWAN gateway, and when you use a public network. It also shows the payload format if you exclude the gateway metadata information. The following shows a sample payload.

```
HTTP/1.1 200
Content-type: application/json

{
  "LastUplinkReceivedAt": "2021-03-24T23:13:08.476015749Z",
```

```

"LoRaWAN": {
  "DataRate": 5,
  "DevEui": "647fda0000006420",
  "Frequency": 868100000,
  "Gateways": [
    {
      "GatewayEui": "c0ee40ffff29df10",
      "Rssi": -67,
      "Snr": 9.75
    }
  ],
  "WirelessDeviceId": "30cbdcf3-86de-4291-bfab-5bfa2b12bad5"
}

```

Payload example with private LoRaWAN gateway

This example uses a private LoRaWAN gateway to show the gateway metadata information in the uplink message. The metadata consists of the gateway EUI, SNR (signal to noise ratio), and RSSI (Received signal to strength indicator). These values can help you determine the strength of your gateway channel and whether to switch to a stronger channel.

```

{
  "MessageId": "d8374454-f361-4907-9f3f-ca53233bb281",
  "WirelessDeviceId": "d7c96c47-6058-46d6-a033-c67d28c2243c",
  "PayloadData": "w0r7P9SI8tsIgm10=",
  "WirelessMetadata":
  {
    "LoRaWAN":
    {
      "ADR": false,
      "Bandwidth": 125,
      "ClassB": false,
      "CodeRate": "4/5",
      "DataRate": "0",
      "DevAddr": "725dd3eb",
      "DevEui": "ac1f09fffe081943",
      "FCnt": 5,
      "FOptLen": 0,
      "FPort": 1,
      "Frequency": "868300000",
      "Gateways": [
        {
          "GatewayEui": "2cf7f11053100080",

```



```

        "Rssi": -34,
        "Snr": 9.5
    }
],
"MIC": "9eb0337c",
"MType": "UnconfirmedDataUp",
"Major": "LoRaWANR1",
"Modulation": "LORA",
"PolarizationInversion": false,
"SpreadingFactor": 12,
"Timestamp": "2023-12-01T16:16:11Z"
}
}
}

```

Payload example with public network

You can also connect to the public network instead of your own private LoRaWAN gateway. The public network is provided and operated as a service directly by Everynet. The following example shows the public LoRaWAN network metadata in the message. The metadata consists of the ID of the gateway and the network provider (Everynet), whether downlink is allowed, and the SNR and RSSI values. For more information about the public network, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

Note

The uplink message will mention `PublicGateways` to indicate that it's received from the public network and not a private LoRaWAN gateway.

```

{
  "MessageId": "d8374454-f361-4907-9f3f-ca53233bb281",
  "WirelessDeviceId": "d7c96c47-6058-46d6-a033-c67d28c2243c",
  "PayloadData": "w0r7P9SI8tsIgMl0=",
  "WirelessMetadata":
  {
    "LoRaWAN":
    {
      "ADR": false,
      "Bandwidth": 125,
      "ClassB": false,

```

```

    "CodeRate": "4/5",
    "DataRate": "0",
    "DevAddr": "725dd3eb",
    "DevEui": "ac1f09fffe081943",
    "FCnt": 5,
    "FOptLen": 0,
    "FPort": 1,
    "Frequency": "868300000",
    "PublicGateways": [
      {
        "DLAllowed": true,
        "Id": "3abe094",
        "ProviderNetId": "0x0000b",
        "RfRegion": "US915",
        "Rssi": -12,
        "Snr": 6.75
      }
    ],
    "MIC": "9eb0337c",
    "MType": "UnconfirmedDataUp",
    "Major": "LoRaWANR1",
    "Modulation": "LORA",
    "PolarizationInversion": false,
    "SpreadingFactor": 12,
    "Timestamp": "2023-12-01T16:16:11Z"
  }
}

```

Payload example without gateway metadata

If you want to exclude the gateway metadata information from your uplink metadata, disable the **AddGwMetadata** parameter when you create the service profile. For information about disabling this parameter, see [Add service profiles](#).

In this case, you won't see the Gateways section in the uplink metadata, as illustrated in the following example.

```

{
  "MessageId": "d8374454-f361-4907-9f3f-ca53233bb281",
  "WirelessDeviceId": "d7c96c47-6058-46d6-a033-c67d28c2243c",
  "PayloadData": "w0r7P9SI8tsIgm10=",
  "WirelessMetadata":

```

```
{
  "LoRaWAN":
  {
    "ADR": false,
    "Bandwidth": 125,
    "ClassB": false,
    "CodeRate": "4/5",
    "DataRate": "0",
    "DevAddr": "725dd3eb",
    "DevEui": "ac1f09fffe081943",
    "FCnt": 5,
    "FOptLen": 0,
    "FPort": 1,
    "Frequency": "868300000",
    "MIC": "9eb0337c",
    "MType": "UnconfirmedDataUp",
    "Major": "LoRaWANR1",
    "Modulation": "LORA",
    "PolarizationInversion": false,
    "SpreadingFactor": 12,
    "Timestamp": "2023-12-01T16:16:11Z"
  }
}
```

Check device connection status

The following sections show you how to check the connection status using the AWS Management Console and the AWS CLI.

Check device connection status using the console

To check the connection status using the console, navigate to the [Devices](#) page of the AWS IoT console and choose the device you've added. In the **Details** section of the Wireless devices details page, you'll see the date and time the last uplink was received.

Check device connection status using the API

To check the connection status using the API, use the [GetWirelessDeviceStatistics API](#). This API doesn't have a request body and only contains a response body that shows when the last uplink was received. The response from the API also indicates whether it's received from a public network or a private LoRaWAN gateway.

Next steps

Now that you have connected your device and verified the connection status, you can observe the format of the uplink metadata received from the device by using the [MQTT test client](#) on the **Test** page of the AWS IoT console. For more information, see [View format of uplink messages sent from LoRaWAN devices](#).

Configuring position of wireless resources with AWS IoT Core for LoRaWAN

Before using this feature, note that the chosen third party provider for resolving position information for LoRaWAN devices relies on data feeds and data sets provided or maintained by the International GNSS Service (IGS), EarthData via NASA, or other third-parties. These data feeds and data sets are Third-Party Content (as defined in the Customer Agreement) and provided on an as-is basis. For more information, see [AWS Service Terms](#).

You can use AWS IoT Core for LoRaWAN to specify your static position data, or activate positioning to identify the position of your device in real time using third-party solvers. You can add or update the position information for either LoRaWAN devices or gateways, or both.

You specify the position information either when adding your device or gateway to AWS IoT Core for LoRaWAN, or when editing the configuration details of your device or gateway. The position information is specified as a [GeoJSON](#) payload. The GeoJSON format is a format that's used to encode geographic data structures. The payload contains the latitude and longitude co-ordinates of your device location, that are based on the [World Geodetic System coordinate system \(WGS84\)](#).

After the solvers compute the position of your resource, if you have Amazon Location Service, you can activate an Amazon Location map where the position of your resource will be displayed. Using the position data, you can:

- Activate positioning to identify and obtain the position of your LoRaWAN devices.
- Track and monitor the position of your gateways and devices.
- Define AWS IoT rules that process any updates to the position data and routes it to other AWS services. For a list of rule actions, see [AWS IoT rule actions](#) in the *AWS IoT developer guide*.
- Create alerts and receive notifications to devices in case of any unusual activity by using the position data and Amazon SNS.

How positioning works for LoRaWAN devices

You can activate positioning to identify the position of your devices using third-party Wi-Fi and GNSS solvers. This information can be used to track and monitor your device. The following steps show you how to activate positioning and view the position information for LoRaWAN devices.

Note

The third-party solvers can only be used with LoRaWAN devices that have the [LoRa Edge](#) chip. It can't be used with LoRaWAN gateways. For gateways, you can still specify the static position information and identify the location on an Amazon Location map.

1. Add your device

Before you activate positioning, first add your device to AWS IoT Core for LoRaWAN. The LoRaWAN device must have the LoRa Edge chipset, which is an ultra-low power platform that integrates a long range LoRa transceiver, multi-constellation GNSS scanner, and passive Wi-Fi MAC scanner targeting geolocation applications.

2. Activate positioning

To obtain the real-time position of your devices, activate positioning. When your LoRaWAN device sends an uplink message, the Wi-Fi and GNSS scan data contained in the message is sent to AWS IoT Core for LoRaWAN using the geolocation frame port.

3. Retrieve position information

Retrieve the estimated device position from the solvers computed based on the scan results from the transceivers. If the position information was computed using both Wi-Fi and GNSS scan results, AWS IoT Core for LoRaWAN selects the estimated position that has the higher accuracy.

4. View position information

After the solver computes the position information, it will also provide the accuracy information which indicates the difference between the position computed by the solvers and the static position information that you entered. You can also view the device location on an Amazon Location map.

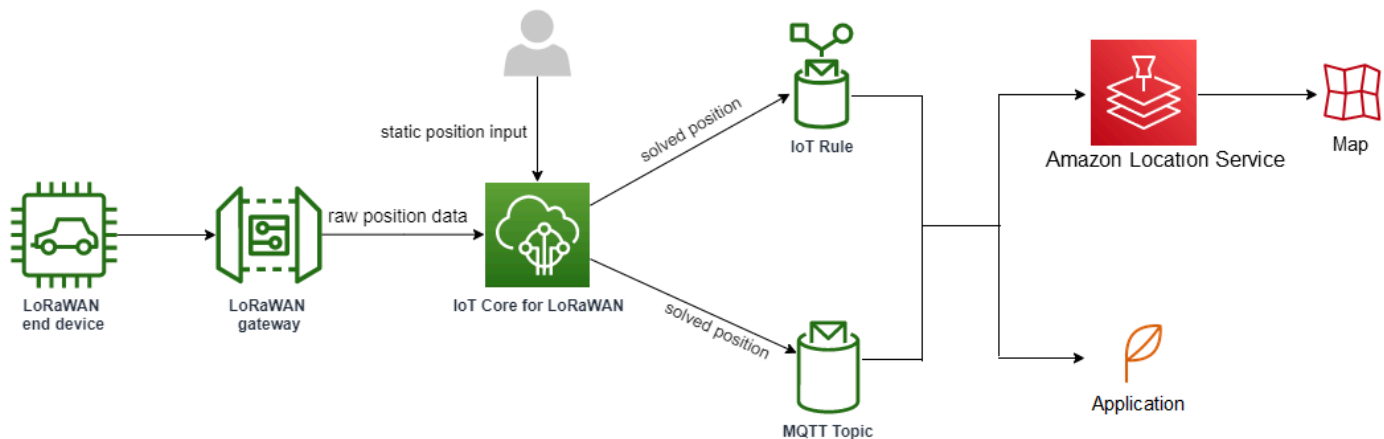
Note

As solvers can't be used for LoRaWAN gateways, the accuracy information will be reported as 0.0 .

For more information about the uplink message format and the frequency ports that are used for the positioning solver, see [Uplink message from AWS IoT Core for LoRaWAN to rules engine](#).

Overview of positioning workflow

The following diagram shows how AWS IoT Core for LoRaWAN stores and updates the position information of your devices and gateways.



1. Specify static position of your resource

Specify the static position information of your device or gateway as a GeoJSON payload, using the latitude and longitude coordinates. You can also specify an optional altitude coordinate.

These coordinates are based on the WGS84 coordinate system. For more information, see [World Geodetic System \(WGS84\)](#).

2. Activate positioning for devices

If you're using LoRaWAN devices that have the LoRa Edge chip, you can optionally activate positioning to track your device position in real time. When your device sends an uplink message, the GNSS and Wi-Fi scan data is sent to AWS IoT Core for LoRaWAN using the geolocation frame port. The solvers then use this information to resolve the device position.

3. Add a destination to route position data

You can add a destination that describes the IoT rule for processing the device data and route the updated position information to AWS IoT Core for LoRaWAN. You can also view the last known position of your resource on an Amazon Location map.

Configuring your resource position

You can configure the position of your resource using the AWS Management Console, the AWS IoT Wireless API, or the AWS CLI.

If your devices have the LoRa Edge chip, you can activate positioning to compute the real-time position information. For your gateways, you can still enter the static position coordinates and use Amazon Location to track the gateway position on an Amazon Location map.

Topics

- [Configuring the position of LoRaWAN gateways](#)
- [Configuring position of LoRaWAN devices](#)

Configuring the position of LoRaWAN gateways

When you add your gateway to AWS IoT Core for LoRaWAN, you can specify the static position data. If you've activated Amazon Location Service maps, the position data is displayed on an Amazon Location map.

Note

The third-party solvers can't be used with LoRaWAN gateways. For gateways, you can still specify the static position coordinates. When solvers aren't used to compute the position, such as in the case of gateways, the accuracy information will be reported as 0.0.

You can configure the gateway position using the AWS Management Console, the AWS IoT Wireless API, or the AWS CLI.

Configuring position of your gateway using the console

To configure the position of your gateway resources by using the AWS Management Console, first sign in to the console and then go to the [Gateways](#) hub page of the AWS IoT console.

Add position information

To add a position configuration for your gateway

1. In the **Gateways** hub page, choose **Add gateway**.
2. Enter the gateway's EUI, frequency band (RFRegion), and any additional gateway details and LoRaWAN configuration information. For more information, see [Add a gateway using the console](#).
3. Go to the **Position information - Optional** section, and enter the position information for your gateway using the latitude and longitude coordinates, and an optional altitude coordinate. The position information is based on the WGS84 coordinate system.

View gateway's position

After you've configured your gateway's position, AWS IoT Core for LoRaWAN creates an Amazon Location map called `iotwireless.map`. You can see this map in the details page of your gateway on the **Position** tab. Based on the position coordinates that you specified, the position of your gateway will be displayed as a marker on the map. You can zoom in or zoom out to clearly view the position of your gateway on the map. On the **Position** tab, you'll also see the accuracy information and the timestamp at which your gateway's position was determined.

Note

If you don't have Amazon Location Service maps installed, you'll see a message indicating that you must use Amazon Location Service to access the map and view the gateway position. Using Amazon Location Service maps may incur additional charges to your AWS account. For more information, see [AWS IoT Core pricing](#).

The map, `iotwireless.map`, acts as a source of map data which is accessed using Get API operations, such as [GetMapTile](#). For information about Get APIs used with maps, see [Amazon Location Service API reference](#).

To get additional details about this map, go to the Amazon Location Service console, choose **maps**, and then choose [iotwireless.map](#). For more information, see [Maps](#) in the *Amazon Location Service developer guide*.

Update gateway's position configuration

To change the gateway's position configuration, in the gateway details page, choose **Edit** and then update the position information and the destination.

Note

Information about historical position data isn't available. When you update the gateway's position coordinates, it overwrites the previously reported position data. After you've updated the position, in the **Position** tab of the gateway details, you'll see the new position information. The change in timestamp indicates that it corresponds to the last known position of the gateway.

Configure position of your gateway using the API

You can specify the position information and configure the gateway position using the AWS IoT Wireless API or the AWS CLI.

Important

The API actions [UpdatePosition](#), [GetPosition](#), [PutPositionConfiguration](#), [GetPositionConfiguration](#), and [ListPositionConfigurations](#) are no longer supported. Calls to update and retrieve the position information should use the [GetResourcePosition](#) and [UpdateResourcePosition](#) API operations instead.

Add position information

To add the static position information for a given wireless gateway, specify the coordinates using the [UpdateResourcePosition](#) API operation or the [update-resource-position](#) CLI command. Specify `WirelessGateway` as the `ResourceType`, the ID of the wireless gateway to be updated as the `ResourceIdentifier`, and the position information as a GeoJSON payload.

```
aws iotwireless update-resource-position \  
  --resource-type WirelessGateway \  
  --resource-id "12345678-a1b2-3c45-67d8-e90fa1b2c34d" \  
  --cli-input-json file://gatewayposition.json
```

The following shows the contents of the `gatewayposition.json` file.

Contents of gatewayposition.json

```
{
  "type": "Point",
  "coordinates": [33.3318, -22.2155, 13.123],
  "properties": {
    "timestamp": "2018-11-30T18:35:24Z"
  }
}
```

Running this command doesn't produce any output. To see the position information that you specified, use the `GetResourcePosition` API operation.

Get position information

To get the position information for a given wireless gateway, use the [GetResourcePosition](#) API operation or the [get-resource-position](#) CLI command. Specify `WirelessGateway` as the `resourceType` and provide the ID of the wireless gateway as the `resourceIdentifier`.

```
aws iotwireless get-resource-position \
  --resource-type WirelessGateway \
  --resource-id "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
```

Running this command displays the position information of your wireless gateway as a GeoJSON payload. You'll see information about the position coordinates, the type of position information, and additional properties, such as the timestamp which corresponds to the last known position of the gateway.

```
{
  {
    "type": "Point",
    "coordinates": [33.3318, -22.2155, 13.123],
    "properties": {
      "timestamp": "2018-11-30T18:35:24Z"
    }
  }
}
```

Configuring position of LoRaWAN devices

When you add your device to AWS IoT Core for LoRaWAN, you can specify the static position information, optionally activate positioning, and specify a destination. The destination describes

the IoT rule that processes the device's position information and routes the updated position to Amazon Location Service. After you configure your device position, the position data is displayed on an Amazon Location map with the accuracy information, and the destination that you specified.

You can configure the position of your device using the AWS Management Console, the AWS IoT Wireless API, or the AWS CLI.

Frame ports and format of uplink messages

If you activate positioning, you must specify the geolocation frame port for communicating the Wi-Fi and GNSS scan data from the device to AWS IoT Core for LoRaWAN. The position information is communicated to AWS IoT Core for LoRaWAN using this frame port.

The LoRaWAN specification provides a data delivery field (FRMPayload) and a Port field (FPort) to distinguish between different types of messages. To communicate the position information, you can specify a value anywhere between 1 and 223 for the frame port. FPort 0 is reserved for MAC messages, FPort 224 is reserved for MAC compliance testing, and ports 225-255 are reserved for future standardized application extensions.

Uplink message from AWS IoT Core for LoRaWAN to rules engine

When you add a destination, it creates an AWS IoT rule to route the data to Amazon Location Service using the rules engine. The updated position information is then displayed on an Amazon Location map. If you haven't activated positioning, the destination routes the position data when you update the static position coordinates of your device.

The following code shows the format of the uplink message sent from AWS IoT Core for LoRaWAN with the position information, accuracy, solver configuration, and the wireless metadata. The fields highlighted below are optional. If there's no vertical accuracy information, the value is null.

```
{
  // Position configuration parameters for given wireless device
  "WirelessDeviceId": "5b58245e-146c-4c30-9703-0ca942e3ff35",

  // Position information for a device in GeoJSON format. Altitude
  // is optional. If no vertical accuracy information is available
  // or positioning isn't activated, the value is set to null.
  // The position information coordinates are listed in the order
  // [longitude, latitude, altitude].
  "coordinates": [33.33000183105469, -22.219999313354492, 99.0],
  "type": "Point",
  "properties": {
```

```
    "horizontalAccuracy": number,
    "verticalAccuracy": number",
    "timestamp": "2022-08-19T03:08:35.061Z"
  },

  //Parameters controlled by AWS IoT Core for LoRaWAN
  "WirelessMetadata":
  {
    "LoRaWAN":
    {
      "ADR": false,
      "Bandwidth": 125,
      "ClassB": false,
      "CodeRate": "4/5",
      "DataRate": "0",
      "DevAddr": "00b96cd4",
      "DevEui": "58a0cb000202c99",
      "FOptLen": 2,
      "FCnt": 1,
      "Fport": 136,
      "Frequency": "868100000",
      "Gateways": [
        {
          "GatewayEui": "80029cffffe5cf1cc",
          "Snr": -29,
          "Rssi": 9.75
        }
      ],
      "MIC": "7255cb07",
      "MType": "UnconfirmedDataUp",
      "Major": "LoRaWANR1",
      "Modulation": "LORA",
      "PolarizationInversion": false,
      "SpreadingFactor": 12,
      "Timestamp": "2021-05-03T03:24:29Z"
    }
  }
}
```

Configuring position of your devices using the console

To configure and manage the position of your devices by using the AWS Management Console, first sign in to the console and then go to the [Devices](#) hub page of the AWS IoT console.

Add position information

To add position information for your device:

1. In the **Devices** hub page, choose **Add wireless device**.
2. Enter the wireless device specification, device and service profiles, and the destination that defines the IoT rule for routing the data to other AWS services. For more information, see [Onboard your devices to AWS IoT Core for LoRaWAN](#).
3. Enter the position information, optionally activate geolocation, and specify a position data destination that you want to use for routing messages.

- **Position information**

Specify the position data for your device using the latitude and longitude coordinates, and an optional altitude coordinate. The position information is based on the WGS84 coordinate system.

- **Geolocation**

Activate positioning if you want AWS IoT Core for LoRaWAN to use geolocation for computing the device position. It uses third-party GNSS and Wi-Fi solvers to identify the position of your device in real time.

To enter the geolocation information, choose **Activate positioning**, and enter the geolocation frame port for communicating the GNSS and Wi-Fi scan data to AWS IoT Core for LoRaWAN. You'll see default FPorts populated for your reference. However, you can choose a different value anywhere between 1 and 223.

- **Position data destination**

Choose a destination to describe the AWS IoT rule that processes the device's position data and forwards it to AWS IoT Core for LoRaWAN. Use this destination only to route position data. It must be different from the destination that you use for routing device data to other AWS services.

View device's position configuration

After you've configured your device's position, AWS IoT Core for LoRaWAN creates an Amazon Location map called `iotwireless.map`. You can see this map in the details page of your device on the **Position** tab. Based on the position coordinates that you specified or the position computed by the third-party solvers, the position of your device will be displayed as a marker on the map. You can zoom in or zoom out to clearly view the position of your device on the map. In the device's details page, on the **Position** tab, you'll also see the accuracy information, the timestamp at which your device's position was determined, and the position data destination that you specified.

Note

If you haven't activated Amazon Location Service maps, you'll see a message indicating that you'll have to use Amazon Location Service to access the map and view the position. Using Amazon Location Service maps may incur additional charges to your AWS account. For more information, see [AWS IoT Core pricing](#).

The map, `iotwireless.map`, acts as a source of map data which is accessed using Get API operations, such as [GetMapTile](#). For information about Get APIs used with maps, see [Amazon Location Service API reference](#).

To get additional details about this map, go to the Amazon Location Service console, choose **maps**, and then choose [iotwireless.map](#). For more information, see [Maps](#) in the *Amazon Location Service developer guide*.

Update device's position configuration

To change the device's position configuration, in the device details page, choose **Edit** and then update the position information, any geolocation settings, and the destination.

Note

Information about historical position data isn't available. When you update the device's position coordinates, it overwrites the previously reported position data. After you've updated the position, in the **Position** tab of the device details, you'll see the new position information. The change in timestamp indicates that it corresponds to the last known position of the device.

Configure device position using the API

You can specify the position information, configure the device position, and activate optional geolocation using the AWS IoT Wireless API or the AWS CLI.

Important

The API actions [UpdatePosition](#), [GetPosition](#), [PutPositionConfiguration](#), [GetPositionConfiguration](#), and [ListPositionConfigurations](#) are no longer supported. Calls to update and retrieve the position information should use the [GetResourcePosition](#) and [UpdateResourcePosition](#) API operations instead.

Add position information and configuration

To add the position information for a given wireless device, specify the coordinates using the [UpdateResourcePosition](#) API operation or the [update-resource-position](#) CLI command. Specify `WirelessDevice` as the `ResourceType`, the ID of the wireless device to be updated as the `ResourceIdentifier`, and the position information.

```
aws iotwireless update-resource-position \  
  --resource-type WirelessDevice \  
  --resource-id "1ffd32c8-8130-4194-96df-622f072a315f" \  
  --position [33.33, -33.33, 10.0]
```

The following shows the contents of the `deviceposition.json` file. To specify the FPort values for sending the geolocation data, use the [Positioning](#) object with the [CreateWirelessDevice](#) and [UpdateWirelessDevice](#) API operations.

Contents of deviceposition.json

```
{  
  "type": "Point",  
  "coordinates": [33.3318, -22.2155, 13.123],  
  "properties": {  
    "verticalAccuracy": 707,  
    "horizontalAccuracy":  
    "timestamp": "2018-11-30T18:35:24Z"  
  }  
}
```

Running this command doesn't produce any output. To see the position information that you specified, use the `GetResourcePosition` API operation.

Get position information and configuration

To get the position information for a given wireless device, use the [GetResourcePosition](#) API or the [get-resource-position](#) CLI command. Specify `WirelessDevice` as the `resourceType` and provide the ID of the wireless device as the `resourceIdentifier`.

```
aws iotwireless get-resource-position \  
  --resource-type WirelessDevice \  
  --resource-id "1ffd32c8-8130-4194-96df-622f072a315f"
```

Running this command displays the position information of your wireless device as a GeoJSON payload. You'll see information about the position coordinates, the location type, and properties which can include the accuracy information and the timestamp which corresponds to the last known position of the device.

```
{  
  "type": "Point",  
  "coordinates": [33.3318, -22.2155, 13.123],  
  "properties": {  
    "verticalAccuracy": 707,  
    "horizontalAccuracy": 389,  
    "horizontalConfidenceLevel": 0.68,  
    "verticalConfidenceLevel": 0.68,  
    "timestamp": "2018-11-30T18:35:24Z"  
  }  
}
```

Managing gateways with AWS IoT Wireless

Following are some important considerations when using your gateways with AWS IoT Core for LoRaWAN. For information about how to add your gateway to AWS IoT Core for LoRaWAN, see [Onboard your gateways to AWS IoT Core for LoRaWAN](#).

LoRa Basics Station software requirement

To connect to AWS IoT Core for LoRaWAN, your LoRaWAN gateway must have software called [LoRa Basics Station](#) running on it. LoRa Basics Station is an open source software that is maintained

by Semtech Corporation and distributed by their [GitHub](#) repository. AWS IoT Core for LoRaWAN supports LoRa Basics Station version 2.0.4 and later. The latest version is 2.0.6.

Using qualified gateways from the AWS Partner Device Catalog

The [AWS Partner Device Catalog](#) contains gateways and developer kits that are qualified for use with AWS IoT Core for LoRaWAN. We recommend that you use these qualified gateways because you don't have to modify the embedding software for connecting the gateways to AWS IoT Core. These gateways already have a version of the BasicStation software compatible with AWS IoT Core for LoRaWAN.

Note

If you have a gateway that is not listed in the Partner Catalog as a qualified gateway with AWS IoT Core for LoRaWAN, you might still be able to use it if the gateway is running LoRa Basics Station software with version 2.0.4 and later. Make sure that you use **TLS Server and Client Authentication** for authenticating your LoRaWAN gateway.

Using CUPS and LNS protocols

LoRa Basics Station software contains two sub protocols for connecting gateways to network servers, LoRaWAN Network Server (LNS) and Configuration and Update Server (CUPS) protocols.

The LNS protocol establishes a data connection between a LoRa Basics Station compatible gateway and a network server. LoRa uplink and downlink messages are exchanged through this data connection over secure WebSockets.

The CUPS protocol enables credentials management, and remote configuration and firmware update of gateways. AWS IoT Core for LoRaWAN provides both LNS and CUPS endpoints for LoRaWAN data ingestion and remote gateway management respectively.

For more information, see [LNS protocol](#) and [CUPS protocol](#).

Topics

- [Configure beaconing and filtering capabilities of your LoRaWAN gateways](#)
- [Update gateway firmware using CUPS service with AWS IoT Core for LoRaWAN](#)
- [Choosing gateways to receive the LoRaWAN downlink data traffic](#)

Configure beaconing and filtering capabilities of your LoRaWAN gateways

When working with LoRaWAN devices, you can configure certain optional parameters for your LoRaWAN gateways. The parameters include:

- **Beaconing**

You can configure beaconing parameters for your LoRaWAN gateways that are acting as a bridge for your class B LoRaWAN devices. These devices receive a downlink message at scheduled time slots, so you must configure the beaconing parameters for your gateways to transmit these time-synchronized beacons.

- **Filtering**

You can configure the NetID and JoinEUI parameters for your LoRaWAN gateways to filter the device data traffic. Filtering the traffic helps conserve bandwidth usage and reduces the traffic flow between the gateways and LNS.

- **Sub-bands**

You can configure the sub-bands for your gateway to specify the particular sub-band that you want to use. For wireless devices that can't hop between the various sub-bands, you can use this capability to communicate with the devices using only the frequency channels in that particular sub-band.

The following topics contain more information about these parameters and how to configure them. The beaconing parameters aren't available in the AWS Management Console and can only be specified using the AWS IoT Wireless API or the AWS CLI.

Topics

- [Configuring your gateways to send beacons to class B devices](#)
- [Configure your gateway's subbands and filtering capabilities](#)

Configuring your gateways to send beacons to class B devices

If you onboard class B wireless devices to AWS IoT Core for LoRaWAN, the devices receive downlink messages in scheduled time slots. The devices open these slots based on time-synchronized beacons that are transmitted by the gateway. For your gateways to transmit these time-

synchronous beacons, you can use AWS IoT Core for LoRaWAN to configure certain beaconing-related parameters for the gateways.

To configure these beaconing parameters, your gateway must be running LoRa Basics Station software version 2.0.6. See [Using qualified gateways from the AWS Partner Device Catalog](#).

How to configure the beaconing parameters

Note

You only need to configure the beaconing parameters for your gateway if it's communicating with a class B wireless device.

You configure the beaconing parameters when adding your gateway to AWS IoT Core for LoRaWAN using the [CreateWirelessGateway](#) API operation. When you invoke the API operation, specify the following parameters using the `Beaconing` object for your gateways. After you configure the parameters, the gateways will send the beacons to your devices at a 128-second interval.

- `DataRate`: The data rate for the gateways that are transmitting the beacons.
- `Frequencies`: The list of frequencies for the gateways to transmit the beacons.

The following example shows how you configure these parameters for the gateway. The `input.json` file will contain additional details, such as the gateway certificate and provisioning credentials. For more information about adding your gateway to AWS IoT Core for LoRaWAN using the `CreateWirelessGateway` API operation, see [Add a gateway by using the API](#).

Note

The beaconing parameters aren't available when you add your gateway to AWS IoT Core for LoRaWAN using the AWS IoT console.

```
aws iotwireless create-wireless-gateway \  
  --name "myLoRaWANGateway" \  
  --cli-input-json file://input.json
```

The following shows the contents of the `input.json` file.

Contents of input.json

```
{
  "Description": "My LoRaWAN gateway",
  "LoRaWAN": {
    "Beaconing": {
      "DataRate": 8,
      "Frequencies": [923300000, 923900000]
    },
    "GatewayEui": "a1b2c3d4567890ab",
    "RfRegion": US915,
    "JoinEuiFilters": [
      ["0000000000000001", "00000000000000ff"],
      ["000000000000ff00", "000000000000ffff"]
    ],
    "NetIdFilters": ["000000", "000001"],
    "RfRegion": "US915",
    "SubBands": [2]
  }
}
```

The following code shows a sample output of running this command.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:400232685877aa:WirelessGateway/a01b2c34-
d44e-567f-abcd-0123e445663a",
  "Id": "a01b2c34-d44e-567f-abcd-0123e445663a"
}
```

Get information about the beaconing parameters

You can get information about the beaconing parameters for your gateway using the [GetWirelessGateway](#) API operation.

Note

If a gateway has already been onboarded, you can't use the `UpdateWirelessGateway` API operation to configure the beaconing parameters. To configure the parameters, you must delete the gateway and then specify the parameters when adding your gateway using the `CreateWirelessGateway` API operation.

```
aws iotwireless get-wireless-gateway \  
  --identifier "12345678-a1b2-3c45-67d8-e90fa1b2c34d" \  
  --identifier-type WirelessGatewayId
```

Running this command returns information about your gateway and the beaconing parameters.

Configure your gateway's subbands and filtering capabilities

LoRaWAN gateways run a [LoRa Basics Station](#) software that enables the gateways to connect to AWS IoT Core for LoRaWAN. To connect to AWS IoT Core for LoRaWAN, your LoRa gateway first queries the CUPS server for the LNS endpoint, and then establishes a WebSockets data connection with that endpoint. After the connection is established, uplink and downlink frames can be exchanged through that connection.

Filtering of LoRa data frames received by gateway

After your LoRaWAN gateway establishes a connection to the endpoint, AWS IoT Core for LoRaWAN responds with a `router_config` message that specifies a set of parameters for the LoRa gateway's configuration, including filtering parameters `NetID` and `JoinEui`. For more information about `router_config` and how a connection is established with the LoRaWAN Network Server (LNS), see [LNS protocol](#).

```
{  
  "msgtype"      : "router_config"  
  "NetID"       : [ INT, .. ]  
  "JoinEui"     : [ [INT,INT], .. ] // ranges: beg,end inclusive  
  "region"      : STRING           // e.g. "EU863", "US902", ..  
  "hwspec"     : STRING  
  "freq_range" : [ INT, INT ]      // min, max (hz)  
  "DRs"        : [ [INT,INT,INT], .. ] // sf,bw,dnonly  
  "sx1301_conf": [ SX1301CONF, .. ]  
  "nocca"      : BOOL  
  "nodc"       : BOOL  
  "nodwell"    : BOOL  
}
```

The gateways carry LoRaWAN device data to and from LNS usually over high-bandwidth networks like Wi-Fi, Ethernet, or Cellular. The gateways usually pick up all messages and pass through the traffic that comes to it to AWS IoT Core for LoRaWAN. However, you can configure the gateways to

filter some of the device data traffic, which helps conserve bandwidth usage and reduces the traffic flow between the gateway and LNS.

To configure your LoRa gateway to filter the data frames, you can use the parameters `NetID` and `JoinEui` in the `router_config` message. `NetID` is a list of `NetID` values that are accepted. Any LoRa data frame carrying a data frame other than those listed will be dropped. `JoinEui` is a list of pairs of integer values encoding ranges of `JoinEUI` values. Join request frames will be dropped by the gateway unless the field `JoinEui` in the message is within the range `[BegEui,EndEui]`.

Frequency channels and subbands

For US915 and AU915 RF regions, wireless devices have choices of 64 125KHz and 8 500KHz uplink channels to access the LoRaWAN networks using the LoRa gateways. The uplink frequency channels are divided into 8 subbands, each with 8 125KHz channels and one 500KHz channel. For each regular gateway in AU915 region, one or more subbands will be supported.

Some wireless devices can't hop between subbands and use the frequency channels in only one subband when connected to AWS IoT Core for LoRaWAN. For the uplink packets from those devices to be transmitted, configure the LoRa gateways to use that particular subband. For gateways in other RF regions, such as EU868, this configuration is not required.

Configure your gateway to use filtering and subbands using the console

You can configure your gateway to use a particular subband and also enable the capability to filter the LoRa data frames. To specify these parameters using the console:

1. Navigate to the [AWS IoT Core for LoRaWAN Gateways](#) page of the AWS IoT console and choose **Add gateway**.
2. Specify the gateway details such as the **Gateway's Eui**, **Frequency band (RFRegion)** and an optional **Name** and **Description**, and choose whether to associate an AWS IoT thing to your gateway. For more information about how to add a gateway, see [Add a gateway using the console](#).
3. In the **LoRaWAN configuration** section, you can specify the subbands and filtering information.
 - **SubBands**: To add a subband, choose **Add SubBand** and specify a list of integer values that indicate which subbands are supported by the gateway. The `SubBands` parameter can only be configured in the `RfRegion` US915 and AU915 and must have values in the range `[1, 8]` within one of these supported regions.

- **NetIdFilters**: To filter uplink frames, choose **Add NetId** and specify a list of string values that the gateway uses. The NetID of the incoming uplink frame from the wireless device must match at least one of the listed values, otherwise the frame is dropped.
 - **JoinEuiFilters**: Choose **Add JoinEui range** and specify a list of pairs of string values that a gateway uses to filter LoRa frames. The JoinEUI value specified as part of the join request from the wireless device must be within the range of at least one of the JoinEuiRange values, each listed as a pair of [BegEui, EndEui], otherwise the frame is dropped.
4. You can then continue to configure your gateway by following the instructions described in [Add a gateway using the console](#).

After you've added a gateway, in the [AWS IoT Core for LoRaWAN Gateways](#) page of the AWS IoT console, if you select the gateway that you've added, you can see the SubBands and filters **NetIdFilters** and **JoinEuiFilters** in the **LoRaWAN specific details** section of the Gateway details page.

Configure your gateway to use filtering and subbands using the API

You can use the [CreateWirelessGateway](#) API that you use to create a gateway to configure the subbands you want to use and enable the filtering capability. Using the [CreateWirelessGateway](#) API, you can specify the subbands and filters as part of the gateway configuration information that you provide using the **LoRaWAN** field. The following shows the request token that includes this information.

```
POST /wireless-gateways HTTP/1.1
Content-type: application/json

{
  "Arn": "arn:aws:iotwireless:us-east-1:400232685877aa:WirelessGateway/
    a11e3d21-e44c-471c-afca-6716c228336a",
  "Description": "Using my first LoRaWAN gateway",
  "LoRaWAN": {
    "GatewayEui": "a1b2c3d4567890ab",
    "JoinEuiFilters": [
      ["0000000000000001", "00000000000000ff"],
      ["000000000000ff00", "000000000000ffff"]
    ],
    "NetIdFilters": ["000000", "000001"],
    "RfRegion": "US915",
    "SubBands": [2]
  }
}
```

```
},  
  "Name": "myFirstLoRaWANGateway"  
  "ThingArn": null,  
  "ThingName": null  
}
```

You can also use the [UpdateWirelessGateway](#) API to update the filters but not the subbands. If the `JoinEuiFilters` and `NetIdFilters` values are null, it means there is no update for the fields. If the values aren't null and empty lists are included, then the update is applied. To get the values of the fields that you specified, use the [GetWirelessGateway](#) API.

Update gateway firmware using CUPS service with AWS IoT Core for LoRaWAN

The [LoRa Basics Station](#) software that runs on your gateway provides credential management and firmware update interface using the Configuration and Update Server (CUPS) protocol. The CUPS protocol provides secure firmware update delivery with ECDSA signatures.

You'll have to frequently update your gateway's firmware. You can use the CUPS service with AWS IoT Core for LoRaWAN to provide firmware updates to the gateway where the updates can also be signed. To update the gateway's firmware, you can use the SDK or CLI but not the console.

The update process takes about 45 minutes to complete. It can take longer if you're setting up your gateway for the first time to connect to AWS IoT Core for LoRaWAN. Gateway manufacturers usually provide their own firmware update files and signatures so you can use that instead and proceed to [Upload the firmware file to an S3 bucket and add an IAM role](#).

If you don't have the firmware update files, see [Generate the firmware update file and signature](#) for an example that you can use to adapt to your application.

To perform your gateway's firmware update:

- [Generate the firmware update file and signature](#)
- [Upload the firmware file to an S3 bucket and add an IAM role](#)
- [Schedule and run the firmware update by using a task definition](#)

Generate the firmware update file and signature

The steps in this procedure are optional and depend on the gateway you're using. Gateway manufacturers provide their own firmware update in the form of an update file or a script and

Basics Station runs this script in the background. In this case, you'll most likely find the firmware update file in the release notes of the gateway you're using. You can then use that update file or script instead and proceed to [Upload the firmware file to an S3 bucket and add an IAM role](#).

If you don't have this script, following shows the commands to run for generating the firmware update file. The updates can also be signed to ensure that the code was not altered or corrupted and devices run code published only by trusted authors.

In this procedure, you'll:

- [Generate the firmware update file](#)
- [Generate signature for the firmware update](#)
- [Review the next steps](#)

Generate the firmware update file

The LoRa Basics Station software running on the gateway is capable of receiving firmware updates in the CUPS response. If you don't have a script provided by the manufacturer, refer to the following firmware update script that is written for the Raspberry Pi based RAKWireless Gateway. We have a base script and the new station binary, version file, and `station.conf` are attached to it.

Note

The script is specific to the RAKWireless Gateway, so you'll have to adapt it to your application depending on the gateway you're using.

Base script

Following shows a sample base script for the Raspberry Pi based RAKWireless Gateway. You can save the following commands in a file `base.sh` and then run the script in the terminal on the Raspberry Pi's web browser.

```
#!/bin/bash*
execution_folder=/home/pi/Documents/basicstation/examples/aws_lorawan
station_path="$execution_folder/station"
version_path="$execution_folder/version.txt"
station_conf_path="$execution_folder/station_conf"
```

```
# Function to find the Basics Station binary at the end of this script
# and store it in the station path
function prepare_station()
{
  match=$(grep --text --line-number '^STATION:$' $0 | cut -d ':' -f 1)
  payload_start=$((match + 1))
  match_end=$(grep --text --line-number '^END_STATION:$' $0 | cut -d ':' -f 1)
  payload_end=$((match_end - 1))
  lines=$((payload_end - payload_start + 1))
  head -n $payload_end $0 | tail -n $lines > $station_path
}

# Function to find the version.txt at the end of this script
# and store it in the location for version.txt
function prepare_version()
{
  match=$(grep --text --line-number '^VERSION:$' $0 | cut -d ':' -f 1)
  payload_start=$((match + 1))
  match_end=$(grep --text --line-number '^END_VERSION:$' $0 | cut -d ':' -f 1)
  payload_end=$((match_end - 1))
  lines=$((payload_end - payload_start + 1))
  head -n $payload_end $0 | tail -n $lines > $version_path
}

# Function to find the version.txt at the end of this script
# and store it in the location for version.txt
function prepare_station_conf()
{
  match=$(grep --text --line-number '^CONF:$' $0 | cut -d ':' -f 1)
  payload_start=$((match + 1))
  match_end=$(grep --text --line-number '^END_CONF:$' $0 | cut -d ':' -f 1)
  payload_end=$((match_end - 1))
  lines=$((payload_end - payload_start + 1))
  head -n $payload_end $0 | tail -n $lines > $station_conf_path
}

# Stop the currently running Basics station so that it can be overwritten
# by the new one
killall station

# Store the different files
prepare_station
prepare_versionp
prepare_station_conf
```

```
# Provide execute permission for Basics station binary
chmod +x $station_path

# Remove update.bin so that it is not read again next time Basics station starts
rm -f /tmp/update.bin

# Exit so that rest of this script which has binaries attached does not get executed
exit 0
```

Add payload script

To the base script, we append the Basics Station binary, the version.txt that identifies the version to update to, and station.conf in a script called `addpayload.sh`. Then, run this script.

```
*#!/bin/bash
*
base.sh > fwstation

# Add station
echo "STATION:" >> fwstation
cat $1 >> fwstation
echo "" >> fwstation
echo "END_STATION:" >> fwstation

# Add version.txt
echo "VERSION:" >> fwstation
cat $2 >> fwstation
echo "" >> fwstation
echo "END_VERSION:" >> fwstation

# Add station.conf
echo "CONF:" >> fwstation
cat $3 >> fwstation
echo "END_CONF:" >> fwstation

# executable
chmod +x fwstation
```

After you've run these scripts, you can run the following command in the terminal to generate the firmware update file, `fwstation`.

```
$ ./addpayload.sh station version.txt station.conf
```

Generate signature for the firmware update

The LoRa Basics Station software provides signed firmware updates with ECDSA signatures. To support signed updates, you'll need:

- A signature that must be generated by an ECDSA private key and less than 128 bytes.
- The private key that is used for the signature and must be stored in the gateway with file name of the format `sig-%d.key`. We recommend using the file name `sig-0.key`.
- A 32-bit CRC over the private key.

The signature and CRC will be passed to the AWS IoT Core for LoRaWAN APIs. To generate the previous files, you can use the following script `gen.sh` that is inspired by the [basicstation](#) example in the GitHub repository.

```
*#!/bin/bash

*function ecdsaKey() {
    # Key not password protected for simplicity
    openssl ecparam -name prime256v1 -genkey | openssl ec -out $1
}

# Generate ECDSA key
ecdsaKey sig-0.prime256v1.pem

# Generate public key
openssl ec -in sig-0.prime256v1.pem -pubout -out sig-0.prime256v1.pub

# Generate signature private key
openssl ec -in sig-0.prime256v1.pub -inform PEM -outform DER -pubin | tail -c 64 >
sig-0.key

# Generate signature
openssl dgst -sha512 -sign sig-0.prime256v1.pem $1 > sig-0.signature

# Convert signature to base64
openssl enc -base64 -in sig-0.signature -out sig-0.signature.base64

# Print the crc
```

```
crc_res=$(crc32 sig-0.key)printf "The crc for the private key=%d\n" $((16#$crc_res))

# Remove the generated files which won't be needed later
rm -rf sig-0.prime256v1.pem sig-0.signature sig-0.prime256v1.pub
```

The private key generated by the script should be saved into the gateway. The key file is in binary format.

```
./gen_sig.sh fwstation
read EC key
writing EC key
read EC key
writing EC key
read EC key
writing EC key
The crc for the private key=3434210794

$ cat sig-0.signature.base64
MEQCIDPY/p2s5gXIPNC0gZr+NzeTLpX+WfBo5tYWbh5pQWN3AiBR0en+XlIdMScv
AsfVfU/ZScJCaIkVNZh4esyS8mNIgA==

$ ls sig-0.key
sig-0.key

$ scp sig-0.key pi@192.168.1.11:/home/pi/Documents/basicstation/examples/iotwireless
```

Review the next steps

Now that you have generated the firmware and signature, go to the next topic to upload the firmware file, `fwstation`, to an Amazon S3 bucket. The bucket is a container that will store the firmware update file as an object. You can add an IAM role that will give the CUPS server permission to read the firmware update file in the S3 bucket.

Upload the firmware file to an S3 bucket and add an IAM role

You can use Amazon S3 to create a *bucket*, which is a container that can store your firmware update file. You can upload your file to the S3 bucket and add an IAM role that allows the CUPS server to read your update file from the bucket. For more information about Amazon S3, see [Getting started with Amazon S3](#).

The firmware update file that you want to upload depends on the gateway you're using. If you followed a procedure similar to the one described in [Generate the firmware update file and signature](#), you'll upload the `fwstation` file generated by running the scripts.

This procedure takes about 20 minutes to complete.

To upload your firmware file:

- [Create an Amazon S3 bucket and upload the update file](#)
- [Create an IAM role with permissions to read the S3 bucket](#)
- [Review the next steps](#)

Create an Amazon S3 bucket and upload the update file

You'll create an Amazon S3 bucket by using the AWS Management Console and then upload your firmware update file into the bucket.

Create an S3 bucket

To create an S3 bucket, open the [Amazon S3 console](#). Sign in if you haven't already and then perform the following steps:

1. Choose **Create bucket**.
2. Enter a unique and meaningful name for the **Bucket name**, (for example, `iotwirelessfwupdate`). For recommended naming convention for your bucket, see <https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html>.
3. Make sure you selected the AWS Region selected as the one you used to create your LoRaWAN gateway and device, and the **Block all public access** setting is selected so that your bucket uses the default permissions.
4. Choose **Enable** for **Bucket versioning** which will help you keep multiple versions of the firmware update file in the same bucket.
5. Confirm **Server-side encryption** is set to **Disable** and choose **Create bucket**.

Upload your firmware update file

You can now see your bucket in the list of Buckets displayed in the AWS Management Console. Choose your bucket and complete the following steps to upload your file.

1. Choose your bucket and then choose **Upload**.

2. Choose **Add file** and then upload the firmware update file. If you followed the procedure described in [Generate the firmware update file and signature](#), you'll upload the `fwstation` file, otherwise upload the file provided by your gateway manufacturer.
3. Make sure all settings are set to their default. Make sure that **Predefined ACLs** is set to **private** and choose **Upload** to upload your file.
4. Copy the S3 URI of the file you uploaded. Choose your bucket and you'll see the file you uploaded displayed in the list of **Objects**. Choose your file and then choose **Copy S3 URI**. The URI will be something like: `s3://iotwirelessfwupdate/fwstation` if you named your bucket similar to the example described previously (`fwstation`). You'll use the S3 URI when creating the IAM role.

Create an IAM role with permissions to read the S3 bucket

You'll now create an IAM role and policy that will give CUPS the permission to read your firmware update file from the S3 bucket.

Create an IAM policy for your role

To create an IAM policy for your AWS IoT Core for LoRaWAN destination role, open the [Policies hub of the IAM console](#) and then complete the following steps:

1. Choose **Create policy**, and choose the **JSON** tab.
2. Delete any content from the editor and paste this policy document. The policy provides permissions to access the `iotwireless` bucket and the firmware update file, `fwstation`, stored inside an object.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::iotwirelessfwupdate/fwstation",
```

```
        "arn:aws:s3:::iotwirelessfwupdate"
      ]
    }
  ]
}
```

3. Choose **Review policy**, and in **Name**, enter a name for this policy (for example, `IoTWirelessFwUpdatePolicy`). You'll need this name to use in the next procedure.
4. Choose **Create policy**.

Create an IAM role with the attached policy

You'll now create an IAM role and attach the policy created previously for accessing the S3 bucket. Open the [Roles hub of the IAM console](#) and complete the following steps:

1. 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. Check the IAM policy (for example, `IoTWirelessFwUpdatePolicy`) you created earlier in the search results and choose it.
5. Choose **Next: Tags**, and then choose **Next: Review**.
6. In **Role name**, enter the name of this role (for example, `IoTWirelessFwUpdateRole`), and then choose **Create role**.

Edit trust relationship of the IAM role

In the confirmation message displayed after you ran the previous step, choose the name of the role you created to edit it. You'll edit the role to add the following trust relationship.

1. In the **Summary** section of the role you created, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
2. 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": {}
    }
  ]
}
```

3. To save your changes and exit, choose **Update Trust Policy**.
4. Obtain the ARN for your role. Choose your IAM role and in the Summary section, you'll see a **Role ARN**, such as `arn:aws:iam::123456789012:role/IoTWirelessFwUpdateRole`. Copy this **Role ARN**.

Review the next steps

Now that you have created the S3 bucket and an IAM role that allows the CUPS server to read the S3 bucket, go to the next topic to schedule and run the firmware update. Keep the **S3 URI** and **Role ARN** that you copied previously so that you can enter them to create a task definition that will be run to perform the firmware update.

Schedule and run the firmware update by using a task definition

You can use a task definition to include details about the firmware update and define the update. AWS IoT Core for LoRaWAN provides a firmware update based on information from the following three fields associated with the gateway.

- **Station**

The version and build time of the Basics Station software. To identify this information, you can also generate it by using the Basics Station software that is being run by your gateway (for example, `2.0.5(rpi/std) 2021-03-09 03:45:09`).

- **PackageVersion**

The firmware version, specified by the file `version.txt` in the gateway. While this information might not be present in the gateway, we recommend it as a way to define your firmware version (for example, `1.0.0`).

- **Model**

The platform or model that is being used by the gateway (for example, Linux).

This procedure takes 20 minutes to complete.

To complete this procedure:

- [Get the current version running on your gateway](#)
- [Create a wireless gateway task definition](#)
- [Run the firmware update task and track progress](#)

Get the current version running on your gateway

To determine your gateway's eligibility for a firmware update, the CUPS server checks all three fields, `Station`, `PackageVersion`, and `Model`, for a match when the gateway presents them during a CUPS request. When you use a task definition, these fields are stored as part of the `CurrentVersion` field.

You can use the AWS IoT Core for LoRaWAN API or AWS CLI to get the `CurrentVersion` for your gateway. Following commands show how to get this information using the CLI.

1. If you've already provisioned a gateway, you can get information about the gateway using the [get-wireless-gateway](#) command.

```
aws iotwireless get-wireless-gateway \
  --identifier 5a11b0a85a11b0a8 \
  --identifier-type GatewayEui
```

Following shows a sample output for the command.

```
{
  "Name": "Raspberry pi",
  "Id": "1352172b-0602-4b40-896f-54da9ed16b57",
```

```

    "Description": "Raspberry pi",
    "LoRaWAN": {
      "GatewayEui": "5a11b0a85a11b0a8",
      "RfRegion": "US915"
    },
    "Arn": "arn:aws:iotwireless:us-
east-1:231894231068:WirelessGateway/1352172b-0602-4b40-896f-54da9ed16b57"
  }

```

2. Using the wireless gateway ID reported by the `get-wireless-gateway` command, you can use the [get-wireless-gateway-firmware-information](#) command to get the `CurrentVersion`.

```

aws iotwireless get-wireless-gateway-firmware-information \
  --id "3039b406-5cc9-4307-925b-9948c63da25b"

```

Following shows a sample output for the command, with information from all three fields displayed by the `CurrentVersion`.

```

{
  "LoRaWAN": {
    "CurrentVersion": {
      "PackageVersion": "1.0.0",
      "Model": "rpi",
      "Station": "2.0.5(rpi/std) 2021-03-09 03:45:09"
    }
  }
}

```

Create a wireless gateway task definition

When you create the task definition, we recommend that you specify automatic creation of tasks by using the `AutoCreateTasks` parameter. `AutoCreateTasks` applies to any gateway that has a match for all three parameters mentioned previously. If this parameter is disabled, the parameters have to be manually assigned to the gateway.

You can create the wireless gateway task definition by using the AWS IoT Core for LoRaWAN API or AWS CLI. Following commands show how to create the task definition using the CLI.

1. Create a file, `input.json`, that'll contain the information to pass to the `CreateWirelessGatewayTaskDefinition` API. In the `input.json` file, provide the following information that you obtained earlier:

- **UpdateDataSource**

Provide the link to your object containing the firmware update file that you uploaded to the S3 bucket. (for example, `s3://iotwirelessfwupdate/fwstation`.)

- **UpdateDataRole**

Provide the link to the Role ARN for the IAM role that you created, which provides permissions to read the S3 bucket. (for example, `arn:aws:iam::123456789012:role/IoTWirelessFwUpdateRole`.)

- **SigKeyCRC and UpdateSignature**

This information might be provided by your gateway manufacturer, but if you followed the procedure described in [Generate the firmware update file and signature](#), you'll find this information when generating the signature.

- **CurrentVersion**

Provide the `CurrentVersion` output that you obtained previously by running the `get-wireless-gateway-firmware-information` command.

```
cat input.json
```

Following shows the contents of the `input.json` file.

```
{
  "AutoCreateTasks": true,
  "Name": "FirmwareUpdate",
  "Update":
  {
    "UpdateDataSource" : "s3://iotwirelessfwupdate/fwstation",
    "UpdateDataRole" : "arn:aws:iam::123456789012:role/
IoTWirelessFwUpdateRole",
    "LoRaWAN" :
    {
      "SigKeyCrc": 3434210794,
      "UpdateSignature": "MEQCIDPY/p2ssgXIPNC0gZr+NzeTLpX
+WfBo5tYWbh5pQWN3AiBR0en+X1IdMScvAsfvU/ZScJCa1kVNZh4esyS8mNIgA==",

```

```
        "CurrentVersion" :
        {
            "PackageVersion": "1.0.0",
            "Model": "rpi",
            "Station": "2.0.5(rpi/std) 2021-03-09 03:45:09"
        }
    }
}
```

2. Pass the `input.json` file to the [create-wireless-gateway-task-definition](#) command to create the task definition.

```
aws iotwireless create-wireless-gateway-task-definition \
  --cli-input-json file://input.json
```

Following shows the output of the command.

```
{
  "Id": "4ac46ff4-efc5-44fd-9def-e8517077bb12",
  "Arn": "arn:aws:iotwireless:us-
east-1:231894231068:WirelessGatewayTaskDefinition/4ac46ff4-efc5-44fd-9def-
e8517077bb12"
}
```

Run the firmware update task and track progress

The gateway is ready to receive the firmware update and, once powered on, it connects to the CUPS server. When the CUPS server finds a match in the version of the gateway, it schedules a firmware update.

A task is a task definition in process. As you specified automatic task creation by setting `AutoCreateTasks` to `True`, the firmware update task starts as soon as a matching gateway is found.

You can track the progress of the task by using the `GetWirelessGatewayTask` API. When you run the [get-wireless-gateway-task](#) command the first time, it will show the task status as `IN_PROGRESS`.

```
aws iotwireless get-wireless-gateway-task \
```

```
--id 1352172b-0602-4b40-896f-54da9ed16b57
```

Following shows the output of the command.

```
{
  "WirelessGatewayId": "1352172b-0602-4b40-896f-54da9ed16b57",
  "WirelessGatewayTaskDefinitionId": "ec11f9e7-b037-4fcc-aa60-a43b839f5de3",
  "LastUplinkReceivedAt": "2021-03-12T09:56:12.047Z",
  "TaskCreatedAt": "2021-03-12T09:56:12.047Z",
  "Status": "IN_PROGRESS"
}
```

When you run the command the next time, if the firmware update takes effect, it will show the updated fields, `Package`, `Version`, and `Model` and the task status changes to `COMPLETED`.

```
aws iotwireless get-wireless-gateway-task \
  --id 1352172b-0602-4b40-896f-54da9ed16b57
```

Following shows the output of the command.

```
{
  "WirelessGatewayId": "1352172b-0602-4b40-896f-54da9ed16b57",
  "WirelessGatewayTaskDefinitionId": "ec11f9e7-b037-4fcc-aa60-a43b839f5de3",
  "LastUplinkReceivedAt": "2021-03-12T09:56:12.047Z",
  "TaskCreatedAt": "2021-03-12T09:56:12.047Z",
  "Status": "COMPLETED"
}
```

In this example, we showed you the firmware update using the Raspberry Pi based RAKWireless gateway. The firmware update script stops the running `BasicStation` to store the updated `Package`, `Version`, and `Model` fields so `BasicStation` will have to be restarted.

```
2021-03-12 09:56:13.108 [CUP:INFO] CUPS provided update.bin
2021-03-12 09:56:13.108 [CUP:INFO] CUPS provided signature len=70 keycrc=37316C36
2021-03-12 09:56:13.148 [CUP:INFO] ECDSA key#0 -> VERIFIED
2021-03-12 09:56:13.148 [CUP:INFO] Running update.bin as background process
2021-03-12 09:56:13.149 [SYS:VERB] /tmp/update.bin: Forked, waiting...
2021-03-12 09:56:13.151 [SYS:INFO] Process /tmp/update.bin (pid=6873) completed
2021-03-12 09:56:13.152 [CUP:INFO] Interaction with CUPS done - next regular check in
10s
```

If the firmware update fails, you see a status of `FIRST_RETRY` from the CUPS server, and the gateway sends the same request. If the CUPS server is unable to connect to the gateway after a `SECOND_RETRY`, it will show a status of `FAILED`.

After the previous task was `COMPLETED` or `FAILED`, delete the old task by using the [delete-wireless-gateway-task](#) command before starting a new one.

```
aws iotwireless delete-wireless-gateway-task \  
  --id 1352172b-0602-4b40-896f-54da9ed16b57
```

Choosing gateways to receive the LoRaWAN downlink data traffic

When you send a downlink message from AWS IoT Core for LoRaWAN to your device, you can choose the gateways you want to use for the downlink data traffic. You can specify an individual gateway or choose from a list of gateways to receive the downlink traffic.

How to specify the gateway list

You can specify an individual gateway or the list of gateways to use when sending a downlink message from AWS IoT Core for LoRaWAN to your device using the [SendDataToWirelessDevice](#) API operation. When you invoke the API operation, specify the following parameters using the `ParticipatingGateways` object for your gateways.

Note

The list of gateways you want to use isn't available in the AWS IoT console. You can specify this list of gateways to use only when using the `SendDataToWirelessDevice` API operation or the CLI.

- `DownlinkMode`: Indicates whether to send the downlink message in sequential mode or concurrent mode. For class A devices, specify `UsingUplinkGateway` to use only the chosen gateways from the previous uplink message transmission.
- `GatewayList`: The list of gateways that you want to use for sending the downlink data traffic. The downlink payload will be sent to the specified gateways with the specified frequency. This is indicated using a list of `GatewayListItem` objects, that consists of `GatewayId` and `DownlinkFrequency` pairs.

- **TransmissionInterval**: The duration of time for which AWS IoT Core for LoRaWAN will wait before transmitting the payload to the next gateway.

Note

You can specify this list of gateways to use only when sending the downlink message to a class B or a class C wireless device. If you use a class A device, the gateway that you chose when sending the uplink message will be used when a downlink message is sent to the device.

The following example shows how you specify these parameters for the gateway. The `input.json` file will contain additional details. For more information about sending a downlink message using the `SendDataToWirelessDevice` API operation, see [Perform downlink queue operations by using the API](#).

Note

The parameters for specifying the list of participating gateways aren't available when you send a downlink message from AWS IoT Core for LoRaWAN using the AWS IoT console.

```
aws iotwireless send-data-to-wireless-device \  
  --id "11aa5eae-2f56-4b8e-a023-b28d98494e49" \  
  --transmit-mode "1" \  
  --payload-data "SGVsbG8gVG8gRGV2c2lt" \  
  --cli-input-json file://input.json
```

The following shows the contents of the `input.json` file.

Contents of `input.json`

```
{  
  "WirelessMetadata": {  
    "LoRaWAN": {  
      "FPort": "1",  
      "ParticipatingGateways": {  
        "DownlinkMode": "SEQUENTIAL",
```



```

    "TransmissionInterval": 1200,
    "GatewayList": [
      {
        "DownlinkFrequency": 100000000,
        "GatewayID": a01b2c34-d44e-567f-abcd-0123e445663a
      },
      {
        "DownlinkFrequency": 100000101,
        "GatewayID": 12345678-a1b2-3c45-67d8-e90fa1b2c34d
      }
    ]
  }
}
}
}
}

```

The output of running this command generates a MessageId for the downlink message. In some cases, even if you receive the MessageId, packets can get dropped. For more information about how you can resolve the error, see [Troubleshoot downlink message queue errors](#).

```

{
  MessageId: "6011dd36-0043d6eb-0072-0008"
}

```

Get information about the list of participating gateways

You can get information about the list of gateways that are participating in receiving the downlink message by listing messages in the downlink queue. To list messages, use the [ListQueuedMessages](#) API.

```

aws iotwireless list-queued-messages \
  --wireless-device-type "LoRaWAN"

```

Running this command returns information about the messages in the queue and their parameters.

Managing devices with AWS IoT Core for LoRaWAN

Following are some important considerations when using your devices with AWS IoT Core for LoRaWAN. For information about how to add your device to AWS IoT Core for LoRaWAN, see [Onboard your devices to AWS IoT Core for LoRaWAN](#).

Device considerations

When selecting a device that you want to use for communicating with AWS IoT Core for LoRaWAN, consider the following.

- Available sensors
- Battery capacity
- Energy consumption
- Cost
- Antenna type and transmission range

Using devices with gateways qualified for AWS IoT Core for LoRaWAN

The devices that you use can be paired with wireless gateways that are qualified for use with AWS IoT Core for LoRaWAN. You can find these gateways and developer kits in the [AWS Partner Device Catalog](#). We also recommend that you consider proximity of these devices to your gateways. For more information, see [Using qualified gateways from the AWS Partner Device Catalog](#).

LoRaWAN version

AWS IoT Core for LoRaWAN supports all devices that comply to 1.0.x or 1.1 LoRaWAN specifications standardized by LoRa Alliance.

Activation modes

Before your LoRaWAN device can send uplink data, you must complete a process called *activation* or *join* procedure. To activate your device, you can either use OTAA (Over the air activation) or ABP (Activation by personalization). We recommend that you use OTAA to activate your device because new session keys are generated for each activation, which makes it more secure.

Your wireless device specification is based on the LoRaWAN version and activation mode, which determines the root keys and session keys generated for each activation. For more information, see [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#).

Device classes

LoRaWAN devices can send uplink messages at any time. Listening to downlink messages consumes battery capacity and reduces battery duration. The LoRaWAN protocol specifies three classes of LoRaWAN devices.

- Class A devices sleep most of the time and listen for downlink messages only for a short period of time. These devices are mostly battery-powered sensors with a battery lifetime of up to 10 years.
- Class B devices can receive messages in scheduled downlink slots. These devices are mostly battery-powered actuators.
- Class C devices never sleep and continuously listen to incoming messages and so there isn't much delay in receiving the messages. These devices are mostly mains-powered actuators.

For more information about these wireless device considerations, refer to the resources mentioned in [Learn more about LoRaWAN](#).

Topics

- [Performing adaptive data rate \(ADR\) with AWS IoT Core for LoRaWAN](#)
- [Managing communication between your LoRaWAN devices and AWS IoT](#)
- [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#)

Performing adaptive data rate (ADR) with AWS IoT Core for LoRaWAN

To optimize the device transmission power consumption while making sure that messages from the end devices are received at the gateways, AWS IoT Core for LoRaWAN uses adaptive data rate. Adaptive data rate instructs the end devices to optimize the data rate, transmission power, and the number of retransmissions while attempting to reduce the error rate of the packets received at the gateways. For example, if your end device is located close to the gateways, adaptive data rate reduces the transmission power and increases the data rate.

Topics

- [How adaptive data rate \(ADR\) works](#)
- [Configure data rate limits \(CLI\)](#)

How adaptive data rate (ADR) works

To enable ADR, your device must set the ADR bit in the frame header. Once the ADR bit is set, AWS IoT Core for LoRaWAN sends the `LinkADRReq` MAC command and your devices respond with the `LinkADRRans` command which includes the ACK status of the ADR command. Once your devices ACK the ADR command, it will then follow the ADR instructions from AWS IoT Core for LoRaWAN and adjust the transmission parameter values for optimal data rate.

The AWS IoT Core for LoRaWAN ADR algorithm uses the SINR information in the uplink metadata history to determine the optimal transmission power and data rate to use for the devices. The algorithm uses the 20 most recent uplink messages that start once the ADR bit is set in the frame header. To determine the number of retransmissions, it uses the packet error rate (PER), which is a percentage of the total number of packets that are lost. When you use this algorithm, you can only control the range of data rates, that is, the minimum and maximum limits for the data rates.

Configure data rate limits (CLI)

By default, AWS IoT Core for LoRaWAN will perform ADR when you set the ADR bit in the frame header of your LoRaWAN device. You can control the minimum and maximum limits for the data rate when creating a service profile for your LoRaWAN devices using the AWS IoT Wireless API operation [CreateServiceProfile](#), or the AWS CLI command, [create-service-profile](#).

Note

You cannot specify the maximum and minimum data rate limits when creating a service profile from the AWS Management Console. It can only be specified using the AWS IoT Wireless API or the AWS CLI.

To specify the minimum and maximum limits for the data rate, use the `DrMin` and `DrMax` parameters with the `CreateServiceProfile` API operation. The default minimum and maximum data rate limits are 0 and 15. For example, the following CLI command sets a minimum data rate limit of 3 and a maximum limit of 12.

```
aws iotwireless create-service-profile \  
  --lorawan DrMin=3,DrMax=12
```

Running this command generates an ID and an Amazon Resource Name (ARN) for the service profile.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ServiceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

You can get the values of the parameters specified using the AWS IoT Wireless API operation [GetServiceProfile](#), or the AWS CLI command, [get-service-profile](#).

```
aws iotwireless get-service-profile --id "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
```

Running this command generates the values for the service profile parameters.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:651419225604:ServiceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
  "LoRaWAN": {
    "UlRate": 60,
    "UlBucketSize": 4096,
    "DlRate": 60,
    "DlBucketSize": 4096,
    "AddGwMetadata": false,
    "DevStatusReqFreq": 24,
    "ReportDevStatusBattery": false,
    "ReportDevStatusMargin": false,
    "DrMin": 3,
    "DrMax": 12,
    "PrAllowed": false,
    "HrAllowed": false,
    "RaAllowed": false,
    "NwkGeoLoc": false,
    "TargetPer": 5,
    "MinGwDiversity": 1
  }
}
```

If you've created multiple profiles, you can use the API operation, [ListServiceProfiles](#), or the AWS CLI command, [list-service-profiles](#) to list the service profiles in your AWS account, and then use the `GetServiceProfile` API or the `get-service-profile` CLI command to retrieve the service profile for which you customized the data rate limits.

Managing communication between your LoRaWAN devices and AWS IoT

After you've connected your LoRaWAN device to AWS IoT Core for LoRaWAN, your devices can start sending messages to the cloud. Uplink messages are messages that are sent from your device and received by AWS IoT Core for LoRaWAN. Your LoRaWAN devices can send uplink messages at any time, which are then forwarded to other AWS services and cloud-hosted applications. Messages that are sent from AWS IoT Core for LoRaWAN and other AWS services and applications to your devices are called downlink messages.

The following shows how you can view and manage uplink and downlink messages that are sent between your devices and the Cloud. You can maintain a queue of downlink messages and send these messages to your devices in the order in which they were added to the queue.

Topics

- [View format of uplink messages sent from LoRaWAN devices](#)
- [Queue downlink messages to send to LoRaWAN devices](#)

View format of uplink messages sent from LoRaWAN devices

After you've connected your LoRaWAN device to AWS IoT Core for LoRaWAN, you can observe the format of the uplink message that you'll receive from your wireless device.

Before you can observe the uplink messages

You must have onboarded your wireless device and connected your device to AWS IoT so that it can transmit and receive data. For information about onboarding your device to AWS IoT Core for LoRaWAN, see [Onboard your devices to AWS IoT Core for LoRaWAN](#).

What do the uplink messages contain?

LoRaWAN devices connect to AWS IoT Core for LoRaWAN by using LoRaWAN gateways. The uplink message that you receive from the device will contain the following information.

- Payload data that corresponds to the encrypted payload message that is sent from the wireless device.
- Wireless metadata that includes:

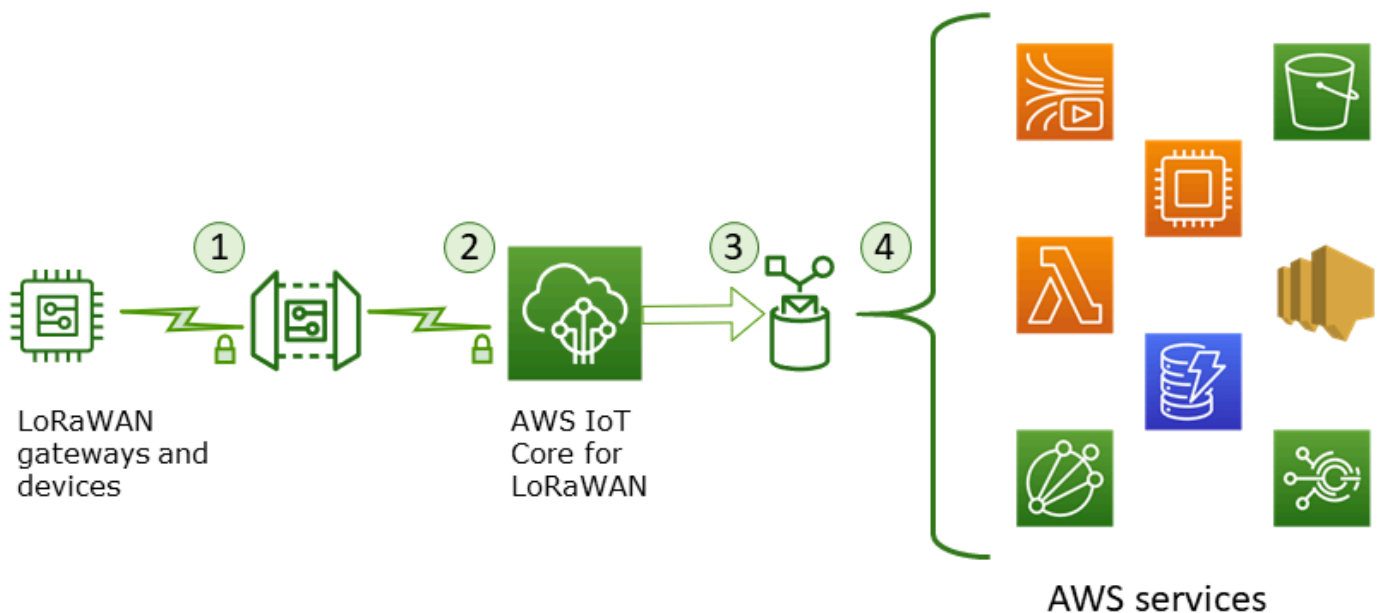
- Device information such as DevEui, the data rate, and the frequency channel in which the device is operating.
- Optional additional parameters and the gateway information for gateways that are connected to the device. The gateway parameters include the gateway's EUI, the SNR, and RSSI.

By using the wireless metadata, you can obtain useful information about the wireless device and the data that is transmitted between your device and AWS IoT. For example, you can use the `AckedMessageId` parameter to check whether the last confirmed downlink message has been received by the device. Optionally, if you choose to include the gateway information, you can identify whether you want to switch to a stronger gateway channel that's closer to your device.

How to observe the uplink messages?

After you've onboarded your device, you can use the [MQTT test client](#) on the **Test** page of the AWS IoT console to subscribe to the topic that you specified when creating your destination. You'll start to see messages after your device is connected and starts sending payload data.

This diagram identifies the key elements in a LoRaWAN system connected to AWS IoT Core for LoRaWAN, which shows the primary data plane and how data flows through the system.



When the wireless device starts sending uplink data, AWS IoT Core for LoRaWAN wraps the wireless metadata information with the payload and then sends it to your AWS applications.

Uplink message examples

The following examples show the format of the uplink message received from your device. The format includes the gateway metadata that varies depending on whether you're using the public network, or your own private LoRaWAN gateway that you onboarded to AWS IoT Core for LoRaWAN.

Uplink message example with private LoRaWAN gateway

This example uses a private LoRaWAN gateway to show the gateway metadata information in the uplink message. The metadata consists of the gateway EUI, SNR (signal to noise ratio), and RSSI (Received signal to strength indicator). These values can help you determine the strength of your gateway channel and whether to switch to a stronger channel.

```
{
  "WirelessDeviceId": "5b58245e-146c-4c30-9703-0ca942e3ff35",
  "PayloadData": "Cc48AAAAAAAAAAAA=",
  "WirelessMetadata":
  {
    "LoRaWAN":
    {
      "ADR": false,
      "Bandwidth": 125,
      "ClassB": false,
      "CodeRate": "4/5",
      "DataRate": "0",
      "DevAddr": "00b96cd4",
      "DevEui": "58a0cb000202c99",
      "FOptLen": 2,
      "FCnt": 1,
      "Fport": 136,
      "Frequency": "868100000",
      "Gateways": [
        {
          "GatewayEui": "80029cffffe5cf1cc",
          "Snr": -29,
          "Rssi": 9.75
        }
      ],
      "MIC": "7255cb07",
      "MType": "UnconfirmedDataUp",
      "Major": "LoRaWANR1",
      "Modulation": "LORA",

```



```

        "PolarizationInversion": false,
        "SpreadingFactor": 12,
        "Timestamp": "2021-05-03T03:24:29Z"
    }
}
}

```

Uplink message example with public network

You can also connect to the public network instead of your own private LoRaWAN gateway. The public network is provided and operated as a service directly by Everynet. The following example shows the public LoRaWAN network metadata in the uplink message. The metadata consists of the ID of the gateway and the network provider (Everynet), whether downlink is allowed, and the SNR and RSSI values. For more information about the public network, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

Note

The uplink message will mention `PublicGateways` to indicate that it's received from the public network and not a private LoRaWAN gateway.

```

{
  "WirelessDeviceId": "5b58245e-146c-4c30-9703-0ca942e3ff35",
  "PayloadData": "Cc48AAAAAAAAAAAA=",
  "WirelessMetadata":
  {
    "LoRaWAN":
    {
      "ADR": false,
      "Bandwidth": 125,
      "ClassB": false,
      "CodeRate": "4/5",
      "DataRate": "0",
      "DevAddr": "00b96cd4",
      "DevEui": "58a0cb000202c99",
      "FOptLen": 2,
      "FCnt": 1,
      "Fport": 136,
      "Frequency": "868100000",

```

```

    "PublicGateways": [
      {
        "DLAllowed": true,
        "Id": "0x3abe094",
        "ProviderNetId": "0x0000b",
        "RfRegion": "US915",
        "Rssi": -12,
        "Snr": 6.75
      }
    ],
    "MIC": "7255cb07",
    "MType": "UnconfirmedDataUp",
    "Major": "LoRaWANR1",
    "Modulation": "LORA",
    "PolarizationInversion": false,
    "SpreadingFactor": 12,
    "Timestamp": "2021-05-03T03:24:29Z"
  }
}

```

Uplink message example without gateway metadata

If you want to exclude the gateway metadata information from your uplink metadata, disable the **AddGwMetadata** parameter when you create the service profile. For information about disabling this parameter, see [Add service profiles](#).

In this case, you won't see the Gateways section in the uplink metadata, as illustrated in the following example.

```

{
  "WirelessDeviceId": "0d9a439b-e77a-4573-a791-49d5c0f4db95",
  "PayloadData": "AAAAAAAAA8=",
  "WirelessMetadata": {
    "LoRaWAN": {
      "ClassB": false,
      "CodeRate": "4/5",
      "DataRate": "1",
      "DevAddr": "01920f27",
      "DevEui": "ffffffff10000163b0",
      "FCnt": 1,
      "FPort": 5,

```

```
    "Timestamp": "2021-04-29T05:19:43.646Z"  
  }  
}  
}
```

Queue downlink messages to send to LoRaWAN devices

Cloud-hosted applications and other AWS services can send downlink messages to your wireless devices. Downlink messages are messages that are sent from AWS IoT Core for LoRaWAN to your wireless device. You can schedule and send downlink messages for each device that you've onboarded to AWS IoT Core for LoRaWAN.

If you have multiple devices for which you want to send a downlink message, you can use a multicast group. Devices in a multicast group share the same multicast address, which is then distributed to an entire group of recipient devices. For more information, see [Create multicast groups to send a downlink payload to multiple devices](#).

How a downlink message queue works

The device class of your LoRaWAN device determines how the messages in your queue are sent to the device. Class A devices send an uplink message to AWS IoT Core for LoRaWAN to indicate that the device is available to receive downlink messages. Class B devices can receive messages at regular downlink slots. Class C devices can receive downlink messages at any time. For more information about device classes, see [Device classes](#).

The following shows how messages are queued and sent to your class A devices.

1. AWS IoT Core for LoRaWAN buffers the downlink message that you added to the queue with the frame port, payload data, and the acknowledge mode parameters that you specified by using the AWS IoT console or the AWS IoT Wireless API.
2. Your LoRaWAN device sends an uplink message to indicate that it's online and can start receiving downlink messages.
3. If you added more than one downlink message to the queue, AWS IoT Core for LoRaWAN sends the first downlink message in the queue to your device with the acknowledge (ACK) flag set.
4. Your device either sends an uplink message to AWS IoT Core for LoRaWAN immediately, or it sleeps until the next uplink message and includes the ACK flag in the message.
5. When AWS IoT Core for LoRaWAN receives the uplink message with the ACK flag, it clears the downlink message from the queue, indicating that your device has successfully received the

downlink message. If the ACK flag is missing from the uplink message after checking three times, the message is discarded.

Perform downlink queue operations by using the console

You can use the AWS Management Console to queue downlink messages and clear individual messages, or the entire queue, as needed. For class A devices, after an uplink is received from the device to indicate that it's online, the queued messages are then sent to the device. After the message is sent, it's automatically cleared from the queue.

Queue downlink messages

To create a downlink message queue

1. Go to the [Devices hub of the AWS IoT console](#) and choose the device for which you want to queue downlink messages.
2. In the **Downlink messages** section of the device details page, choose **Queue downlink messages**.
3. Specify the following parameters to configure your downlink message:
 - **FPort**: Choose the frame port for the device to communicate with AWS IoT Core for LoRaWAN.
 - **Payload**: Specify the payload message that you want to send to your device. The maximum payload size is 242 bytes. If adaptive data rate (ADR) is enabled, AWS IoT Core for LoRaWAN uses it to choose the optimal data rate for your payload size. You can further optimize the data rate as needed.
 - **Acknowledge mode**: Confirm whether your device has received the downlink message. If a message requires this mode, you'll see an uplink message with the ACK flag in your data stream, and the message will be cleared from the queue.
4. To add your downlink message to the queue, choose **Submit**.

Your downlink message has now been added to the queue. If you don't see your message or you receive an error, you can troubleshoot the error as described in [Troubleshoot downlink message queue errors](#).

Note

After your downlink message has been added to the queue, you can no longer edit the parameters **FPort**, **Payload**, and **Acknowledge mode**. To send a downlink message with

different values for these parameters, you can delete this message and queue a new downlink message with the updated parameter values.

The queue lists the downlink messages you've added. To see the payload for the uplink and downlink messages that are exchanged between your devices and AWS IoT Core for LoRaWAN, you can use network analyzer. For more information, see [Monitoring your wireless resource fleet in real time using network analyzer](#).

List downlink message queue

The downlink message that you created is added to the queue. Each subsequent downlink message is added to the queue after this message. You can see a list of downlink messages in the **Downlink messages** section of the device details page. After an uplink is received, the messages are sent to the device. After a downlink message has been received by your device, it will be removed from the queue. The next message then moves up in the queue to be sent to your device.

Delete individual downlink messages or clear entire queue

Each downlink message is cleared from the queue automatically after it's sent to your device. You can also delete individual messages or clear the entire downlink queue. These actions can't be undone.

- If you find messages in the queue that you don't want to send, choose the messages and choose **Delete**.
- If you don't want to send any messages from the queue to your device, you can clear the entire queue by choosing **Clear downlink queue**.

Perform downlink queue operations by using the API

You can use the AWS IoT Wireless API to queue downlink messages and clear individual messages, or the entire queue, as needed.

Queue downlink messages

To create a downlink message queue, use the [SendDataToWirelessDevice](#) API operation or the [send-data-to-wireless-device](#) CLI command.

```
aws iotwireless send-data-to-wireless-device \
```

```
--id "11aa5eae-2f56-4b8e-a023-b28d98494e49" \  
--transmit-mode "1" \  
--payload-data "SGVsbG8gVG8gRGV2c21t" \  
--wireless-metadata LoRaWAN={FPort=1}
```

The output of running this command generates a `MessageId` for the downlink message. In some cases, even if you receive the `MessageId`, packets can get dropped. For more information about how you can resolve the error, see [Troubleshoot downlink message queue errors](#).

```
{  
  MessageId: "6011dd36-0043d6eb-0072-0008"  
}
```

List downlink messages in the queue

To list all downlink messages in the queue, use the [ListQueuedMessages](#) API operation or the [list-queued-messages](#) CLI command.

```
aws iotwireless list-queued-messages
```

By default, a maximum of 10 downlink messages are displayed when running this command.

Remove individual downlink messages or clear entire queue

To remove individual messages from the queue or to clear the entire queue, use the [DeleteQueuedMessages](#) API operation or the [delete-queued-messages](#) CLI command.

- To remove individual messages, provide the `messageID` for messages you want to remove for your wireless device, specified by the `wirelessDeviceId`.
- To clear the entire downlink queue, specify `messageID` as `*` for your wireless device, specified by the `wirelessDeviceId`.

Troubleshoot downlink message queue errors

Here are some things to check if you're not seeing the expected results:

- **Downlink messages don't appear in the AWS IoT console**

If you don't see your downlink message in the queue after adding it as described in [Perform downlink queue operations by using the console](#), it might be because your device hasn't

completed a process called *activation* or *join procedure*. This procedure is completed when your device onboards with AWS IoT Core for LoRaWAN. For more information, see [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#).

After onboarding your device to AWS IoT Core for LoRaWAN, you can monitor your device to check whether join and rejoin succeeded by using the network analyzer or Amazon CloudWatch. For more information, see [Monitoring tools](#).

- **Missing downlink message packets when using the API**

When you use the `SendDataToWirelessDevice` API operation, the API returns a unique `MessageId`. However, it can't confirm whether your LoRaWAN device has received the downlink message. The downlink packets can get dropped in cases such as when your device hasn't completed the join procedure. For more information about how to resolve this error, see the previous section.

- **Missing ARN error when sending downlink message**

When sending a downlink message to your device from the queue, you can receive a missing Amazon Resource Name (ARN) error. This error might occur because the destination hasn't been specified correctly for the device that's receiving the downlink message. To resolve this error, check the destination details for your device.

Managing LoRaWAN traffic from public LoRaWAN devices networks (Everynet)

You can connect your LoRaWAN devices to the cloud in minutes by using publicly available LoRaWAN networks. AWS IoT Core for LoRaWAN now supports Everynet's network coverage in the US, UK, Ireland, and Spain. When using the public network, you'll be charged a public network connectivity charge for each device every month. The pricing applies to all AWS Regions where public network connectivity is offered. For information about pricing for this feature, see the [AWS IoT Core pricing page](#).

Important

The public network is operated and provided as a service directly by Everynet. Before using this feature, see the applicable [AWS Service Terms](#). In addition, if you use a public network through AWS IoT Core for LoRaWAN, certain LoRaWAN device information such as

DevEUI and JoinEUI will be replicated across regions where AWS IoT Core for LoRaWAN is available.

AWS IoT Core for LoRaWAN supports the public LoRaWAN network according to the LoRa Alliance specification for roaming, as described in [LoRaWAN Backend Interfaces 1.0 Specification](#). The public network capability can be used to connect your end devices that are outside the home network. To support this capability, AWS IoT Core for LoRaWAN partners with Everynet to offer extended radio coverage.

Benefits of using a public LoRaWAN network

Your LoRaWAN devices can use a public network to connect to the cloud, which reduces the time to deployment, and reduces the time and cost that are required to maintain a private LoRaWAN network.

By using a public LoRaWAN network, you'll receive benefits such as coverage extension, running core without radio network, and coverage densification. This feature can be used to:

- Provide coverage to devices when they move out of their home network, such as *Device A* in figure shown in the [Public LoRaWAN network support architecture](#) section.
- Extend coverage to devices that don't have a LoRa gateway to connect to, such as *Device B* in figure shown in the [Public LoRaWAN network support architecture](#) section. The device can then use the gateway provided by the partner to connect to the home network.

Your LoRaWAN devices can use a public network to connect to the cloud using the roaming feature, which reduces the time to deployment, and reduces the time and cost that are required to maintain a private LoRaWAN network.

The following sections describe the public network support architecture, how public LoRaWAN network support works, and how to use this feature.

Topics

- [How LoRaWAN public network support works](#)
- [How to use the public network support](#)

How LoRaWAN public network support works

AWS IoT Core for LoRaWAN supports the passive roaming feature, according to the LoRa Alliance specification. With passive roaming, the roaming process is entirely transparent to the end device. End devices that roam outside the home network can connect to gateways in that network and exchange uplink and downlink data using the application server. The devices stay connected to the home network throughout the entire roaming process.

Note

AWS IoT Core for LoRaWAN supports only the stateless feature of passive roaming. Handover roaming is not supported. In handover roaming, your device will switch to a different carrier when it travels outside the home network.

Topics

- [Public LoRaWAN network concepts](#)
- [Public LoRaWAN network support architecture](#)

Public LoRaWAN network concepts

The following concepts are used by the public network feature supported by AWS IoT Core for LoRaWAN.

LoRaWAN network server (LNS)

An LNS is a standalone private server that can run on your premises or can be a cloud-based service. AWS IoT Core for LoRaWAN is an LNS that offers services on the cloud.

Home network server (hNS)

The home network is the network that the device belongs to. The home network server (hNS) is an LNS where AWS IoT Core for LoRaWAN stores the provisioning data of the device, such as the DevEUI, AppEUI, and session keys.

Visited network server (vNS)

The visited network is the network that the device gets coverage from when it leaves the home network. The visited network server (vNS) is an LNS that has a business and technical

agreement with the hNS for being able to serve the end device. AWS partner, Everynet, acts as the visited network to provide coverage.

Serving network server (sNS)

The serving network server (sNS) is an LNS that handles the MAC commands for the device. There can be only one sNS for one LoRa session.

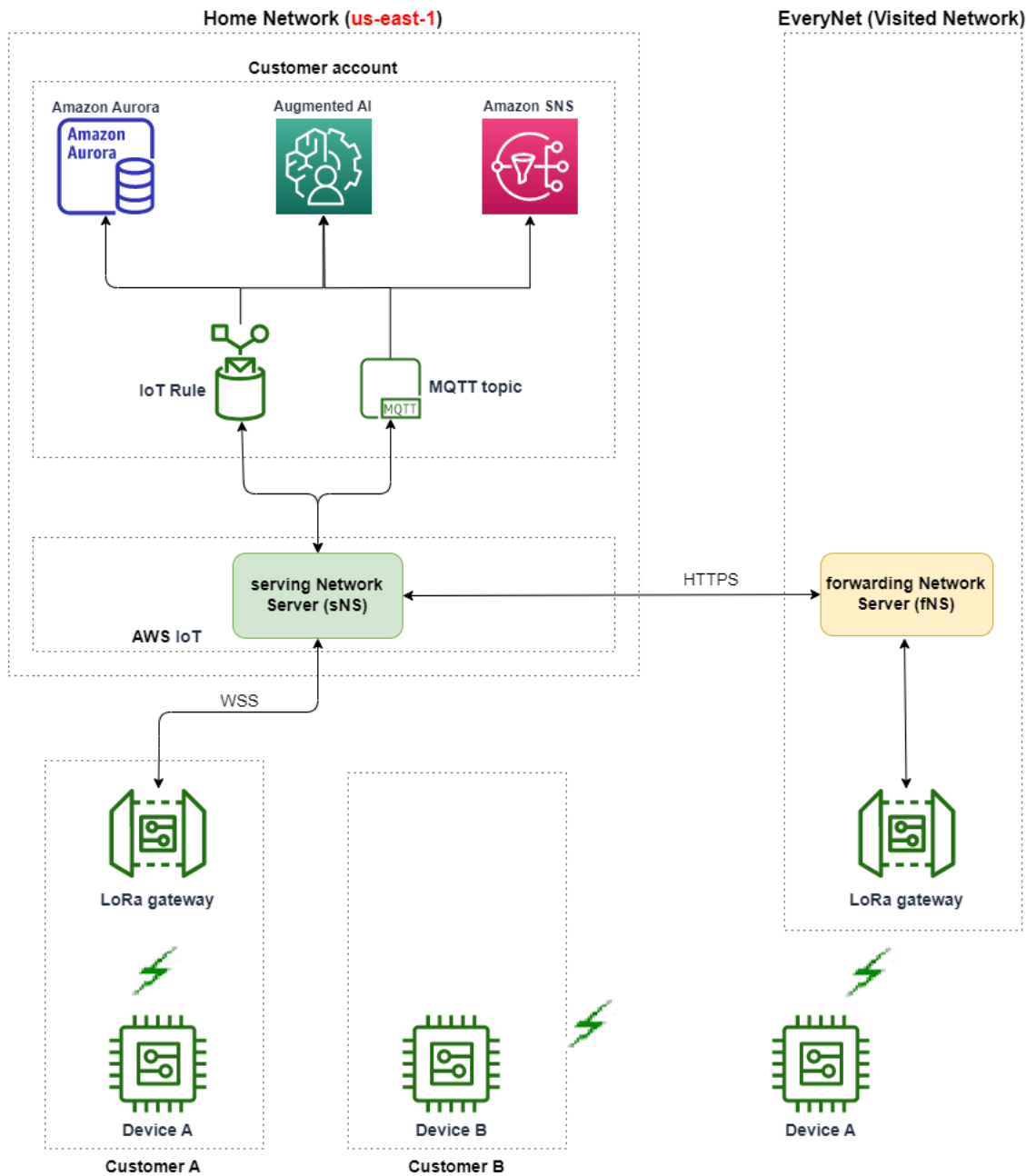
Forwarding network server (fNS)

The forwarding network server (fNS) is an LNS that manages the radio gateways. There can be zero or more fNS involved in one LoRa session. This network server manages the forwarding of data packets that are received from the device to the home network.

Public LoRaWAN network support architecture

The following architecture diagram shows how AWS IoT Core for LoRaWAN partners with Everynet to provide public network connectivity. In this case, *Device A* is connected to the hNS (home network server) provided by AWS IoT Core for LoRaWAN through a LoRa gateway. When *Device A* moves out of the home network, it enters a visited network, and is covered by the visited network server (vNS) provided by Everynet. The vNS also extends coverage to *Device B* which doesn't have a LoRa gateway to connect to.

You can view the public network coverage information in the AWS IoT console as described in the following section.



AWS IoT Core for LoRaWAN uses a roaming hub functionality, in accordance with the [LoRa Alliance LoRaWAN Roaming Hub Technical Recommendation](#). The roaming hub provides an endpoint for Everynet to route the traffic received from the end device. In this case, Everynet acts as a forwarding network server (fNS) to forward the traffic received from the device. It uses an HTTP RESTful API, as defined by the LoRa Alliance specification.

Note

If your device moves from its home network and enters a location where both your home network and Everynet can offer coverage, it uses first-come-first-serve policy to determine whether to connect to your LoRa gateway, or to Everynet's gateway.

When visiting a public network, the hNS and serving network server (sNS) are separated. Uplink and downlink packets are then exchanged between the sNS and hNS.

How to use the public network support

To enable Everynet's public network support, you enable certain roaming parameters when creating the service profile. In this beta release, these parameters are available when you use the AWS IoT Wireless API, or the AWS CLI. The following sections show the parameters you must enable, and how to enable public network using the AWS CLI.

Note

You can enable public network support only when creating a new service profile. You can't update an existing profile to enable public network using these parameters.

Topics

- [Roaming parameters](#)
- [Enable public network support for devices](#)

Roaming parameters

Specify the following parameters when creating a service profile for your device. Specify these parameters when adding a service profile from the [Profiles](#) hub of the AWS IoT console, or using the AWS IoT Wireless API operation, [CreateServiceProfile](#), or the AWS CLI command, [create-service-profile](#).

Note

AWS IoT Core for LoRaWAN does not support handover roaming. When creating the service profile, you can't enable the `HiAllowed` parameter that specifies whether to use handover roaming.

- **Roaming activation allowed (`RaAllowed`):** This parameter specifies whether to enable roaming activation. Roaming activation enables an end device to activate under the coverage of a vNS. When using the roaming feature, `RaAllowed` must be set to `true`.
- **Passive roaming allowed (`PrAllowed`):** This parameter specifies whether to enable passive roaming. When using the roaming feature, `PrAllowed` must be set to `true`.

Enable public network support for devices

To enable public LoRaWAN network support on your devices, run the following procedure.

Note

You can enable the public network capability only for OTAA devices. This feature is not supported for devices that use ABP as the activation method.

1. Create service profile with roaming parameters

Create a service profile by enabling the roaming parameters.

Note

When you create a device profile for the device that you'll associate with this service profile, we recommend that you specify a large value for the `RxDelay1` parameter, at least greater than 2s.

- **Using the AWS IoT console**

Go to the [Profiles](#) hub of the AWS IoT console and choose **Add service profile**. When creating the profile, choose **Enable public network**.

- **Using the AWS IoT Wireless API**

To enable roaming when creating a service profile, use the [CreateServiceProfile](#) API operation or the [create-service-profile](#) CLI command, as shown in example below.

```
aws iotwireless create-service-profile \  
  --region us-east-1 \  
  --name roamingprofile1 \  
  --lorawan '{"AddGwMetadata":true,"PrAllowed":true,"RaAllowed":true}'
```

Running this command returns the ARN and ID of the service profile as output.

```
{  
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ServiceProfile/12345678-  
a1b2-3c45-67d8-e90fa1b2c34d",  
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"  
}
```

2. Check roaming parameters in service profile

To check the roaming parameters that you specified, you can view the service profile in the console, or using the `get-service-profile` CLI command, as shown in example below.

- **Using the AWS IoT console**

Go to the [Profiles](#) hub of the AWS IoT console and choose the profile that you created. In the **Profile configuration** tab of the details page, you'll see **RaAllowed** and **PrAllowed** set to `true`.

- **Using the AWS IoT Wireless API**

To view the roaming parameters that you enabled, use the [GetServiceProfile](#) API operation or the [get-service-profile](#) CLI command, as shown in example below.

```
aws iotwireless get-service-profile \  
  --region us-east-1 \  
  --id 12345678-a1b2-3c45-67d8-e90fa1b2c34d
```

Running this command returns the service profile details as output, including the values for roaming parameters, `RaAllowed` and `PrAllowed`.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ServiceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
  "Name": "roamingprofile1"
  "LoRaWAN": {
    "UlRate": 60,
    "UlBucketSize": 4096,
    "DlRate": 60,
    "DlBucketSize": 4096,
    "AddGwMetadata": true,
    "DevStatusReqFreq": 24,
    "ReportDevStatusBattery": false,
    "ReportDevStatusMargin": false,
    "DrMin": 0,
    "DrMax": 15,
    "PrAllowed": true,
    "RaAllowed": true,
    "NwkGeoLoc": false,
    "TargetPer": 5,
    "MinGwDiversity": 1
  }
}
```

3. Attach service profile to devices

Attach the service profile that you created with the roaming parameters to your end devices. You can also create a device profile and add a destination for your wireless devices. You'll use this destination to route uplink messages that are sent from your device. For more information about creating device profiles and a destination, see [Add device profiles](#) and [Add destinations to AWS IoT Core for LoRaWAN](#).

- **Onboarding new devices**

If you haven't already onboarded your devices, you specify this service profile to be used when adding your device to AWS IoT Core for LoRaWAN. The following command shows how you can use the `create-wireless-device` CLI command to add a device using the ID of the service profile that you created. For information about adding the service profile using the console, see [Add your wireless device specification to AWS IoT Core for LoRaWAN using the console](#).

```
aws iotwireless create-wireless-device --cli-input-json file://createdevice.json
```

The following shows the contents of the file *createdevice.json*.

Contents of createdevice.json

```
{
  "Name": "DeviceA",
  "Type": LoRaWAN,
  "DestinationName": "RoamingDestination1",
  "LoRaWAN": {
    "DeviceProfileId": "ab0c23d3-b001-45ef-6a01-2bc3de4f5333",
    "ServiceProfileId": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
    "OtaaV1_1": {
      "AppKey": "3f4ca100e2fc675ea123f4eb12c4a012",
      "JoinEui": "b4c231a359bc2e3d",
      "NwkKey": "01c3f004a2d6efffe32c4eda14bcd2b4"
    },
    "DevEui": "ac12efc654d23fc2"
  },
}
```

The output of running this command produces the ARN and ID of the wireless device as output.

```
{
  "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:WirelessDevice/1ffd32c8-8130-4194-96df-622f072a315f",
  "Id": "1ffd32c8-8130-4194-96df-622f072a315f"
}
```

- **Updating existing devices**

If you have already onboarded your devices, you can update your existing wireless devices to use this service profile. The following command shows how you can use the `update-wireless-device` CLI command to update a device using the ID of the service profile that you created.

```
aws iotwireless update-wireless-device \
  --id "1ffd32c8-8130-4194-96df-622f072a315f" \
```



```
--service-profile-id "12345678-a1b2-3c45-67d8-e90fa1b2c34d" \  
--description "Using roaming service profile A"
```

This command doesn't produce any output. You can use the `GetWirelessDevice` API or the `get-wireless-device` CLI command to get the updated information.

4. Connect device to cloud using Everynet

As roaming has been enabled, your device must now perform a join to obtain a new `DevAddr`. If you're using OTAA, your LoRaWAN device sends a join request and the Network Server can allow the request. It can then connect to the AWS Cloud using the network coverage provided by Everynet. For instructions on how to perform the activation procedure or join for your device, see the device documentation.

Note

- You can enable the roaming capability and connect to the public network only for devices that use OTAA as the activation method. ABP devices aren't supported. For instructions on how to perform the activation procedure or join for your device, see the device documentation. See [Activation modes](#).
- To disable the roaming capability for your devices, you can disassociate the devices from this service profile, and then associate them with another service profile that has the roaming parameters set to `false`. After switching to this service profile, your devices must perform another join so that they don't continue running on the public network.

5. Exchange uplink and downlink messages

After your device has joined to AWS IoT Core for LoRaWAN, you can start exchanging messages between your device and the Cloud.

- **View uplink messages**

When you send uplink messages from your devices, AWS IoT Core for LoRaWAN delivers these messages to your AWS account using the destination that you configured earlier. These messages will be sent from your device to the Cloud over Everynet's network.

You can use either view the messages using the AWS IoT rule name or use the MQTT client to subscribe to the MQTT topic that was specified when creating the destination. For more

information about the rule name and other destination details that you specify, see [Add a destination using the console](#).

For more information about viewing uplink message and the format, see [View format of uplink messages sent from LoRaWAN devices](#).

- **Send downlink messages**

You can queue and send downlink messages to your devices from the console, or by using the AWS IoT Wireless API command, `SendDataToWirelessDevice`, or the AWS CLI command, `send-data-to-wireless-device`. For information about queuing and sending downlink messages, see [Queue downlink messages to send to LoRaWAN devices](#).

The following code shows an example of how you can send a downlink message using the `send-data-to-wireless-device` CLI command. You specify the ID of the wireless device to receive the data, the payload, whether to use the acknowledge mode, and the wireless metadata.

```
aws iotwireless send-data-to-wireless-device \  
  --id "1ffd32c8-8130-4194-96df-622f072a315f" \  
  --transmit-mode "1" \  
  --payload-data "SGVsbG8gVG8gRGV2c2lt" \  
  --wireless-metadata LoRaWAN={FPort=1}
```

The output of running this command generates a `MessageId` for the downlink message.


 **Note**

In some cases, even if you receive the `MessageId`, packets can get dropped. For information about troubleshooting such scenarios and resolving them, see [Troubleshoot downlink message queue errors](#).

```
{  
  MessageId: "6011dd36-0043d6eb-0072-0008"  
}
```

- **View coverage information**

After you've enabled the public network, you can view the network coverage information in the AWS IoT console. Go to the [Coverage](#) hub of the AWS IoT console and then search for locations to see the coverage information of your devices on the map.

 **Note**

This feature uses the Amazon Location Service to display the coverage information of your devices on an Amazon Location map. Before using Amazon Location maps, review the Terms and Conditions for Amazon Location Service. Note that AWS may transmit your API queries to your chosen third party data provider, which may be outside of the AWS Region that you are currently using. For more information, see [AWS Service Terms](#).

Perform firmware update over-the-air (FUOTA) for LoRaWAN devices and multicast groups

You can perform firmware update over-the-air to update the device firmware of a single LoRaWAN device or a group of devices. To update the device firmware or to send a downlink payload to multiple devices, create a multicast group. Using multicast, a source can send data to a single multicast group, which is then distributed to a group of recipient devices.

AWS IoT Core for LoRaWAN's support for FUOTA and multicast groups is based on the [LoRa Alliance's](#) following specifications:

- LoRaWAN Remote Multicast Setup Specification, TS005-2.0.0
- LoRaWAN Fragmented Data Block Transportation Specification, TS004-2.0.0
- LoRaWAN Application Layer Clock Synchronization Specification, TS003-2.0.0

 **Note**

AWS IoT Core for LoRaWAN automatically performs the clock synchronization according to the LoRa Alliance specification. It uses the function `AppTimeReq` to reply the server-side time to the devices that request it using `ClockSync` signaling.

The following topics show how to create multicast groups and perform FUOTA.

Topics

- [Prepare devices for multicast and FUOTA configuration](#)
- [Create multicast groups to send a downlink payload to multiple devices](#)
- [Firmware update over-the-air \(FUOTA\) for AWS IoT Core for LoRaWAN devices](#)

Prepare devices for multicast and FUOTA configuration

When you add your wireless device to AWS IoT Core for LoRaWAN, you can prepare your wireless device for multicast setup and FUOTA configuration by using the console or the CLI. If you're performing this configuration for the first time, we recommend that you use the console. To manage your multicast group and add or remove a number of devices from your group, we recommend using the CLI to manage a large number of resources.

GenAppKey and FPorts

When you add your wireless device, before you can add your devices to multicast groups or perform FUOTA, configure the following parameters. Before you configure these parameters, make sure that your devices support FUOTA and multicast and your wireless device specification is either OTAA v1.1 or OTAAv1.0.x.

- **GenAppKey:** For devices that support the LoRaWAN version 1.0.x and to use multicast groups, the GenAppKey is the device-specific root key from which the session keys for your multicast group are derived.

Note

For LoRaWAN devices that use the wireless specification OTAA v1.1, the AppKey is used for the same purpose as the GenAppKey.

To set up the parameters to initiate the data transfer, AWS IoT Core for LoRaWAN distributes session keys with the end devices. For more information about LoRaWAN versions, see [LoRaWAN version](#).

Note

AWS IoT Core for LoRaWAN stores the GenAppKey information that you provide in an encrypted format.

- **FPorts:** According to the LoRaWAN specifications for FUOTA and multicast groups, AWS IoT Core for LoRaWAN assigns the default values for the following fields of the `FPorts` parameter. If you have already assigned any of the following `FPort` values, then you can choose a different value that is available, from 1 to 223.
 - **Multicast:** 200

This `FPort` value is used for multicast groups.
 - **FUOTA:** 201

This `FPort` value is used for FUOTA.
 - **ClockSync:** 202

This `FPort` value is used for the clock synchronization.

Device profiles for multicast and FUOTA

At the start of a multicast session, a class B or class C distribution window is used to send the downlink message to the devices in your group. The devices that you add for multicast and FUOTA must support class B or class C modes of operation. Depending on the device class that your device supports, choose a device profile for your device that has either or both class B or class C modes enabled.

For information about device profiles, see [Add profiles to AWS IoT Core for LoRaWAN](#).

Prepare devices for multicast and FUOTA by using the console

To specify the `FPorts` and `GenAppKey` parameters for multicast setup and FUOTA by using the console:

1. Navigate to the [Devices hub of the AWS IoT console](#) and choose **Add wireless device**.
2. Choose the **Wireless device specification**. Your device must use OTAA for device activation. When you choose OTAA v1.0.x or OTAA v1.1, a **FUOTA configuration-Optional** section appears.

3. Enter the EUI (Extended Unique Identifier) parameters for your wireless device.
4. Expand the **FUOTA configuration-Optional** section and then choose **This device supports firmware updates over the air (FUOTA)**. You can now enter the **FPort** values for multicast, FUOTA, and clock sync. If you chose OTAA v1.0.x for the wireless device specification, enter the **GenAppKey**.
5. Add your device to AWS IoT Core for LoRaWAN by choosing your profiles and a destination for routing messages. For the device profile linked to the device, make sure you select one or both **Supports Class B** or **Supports Class C** modes.

Note

To specify the FUOTA configuration parameters, you must use the [Devices hub of the AWS IoT console](#). These parameters don't appear if you onboard your devices by using the **Intro** page of the AWS IoT console.

For more information about the wireless device specification and onboarding your device, see [Add your wireless device to AWS IoT Core for LoRaWAN](#).

Note

You can specify these parameters only when you create the wireless device. You can't change or specify parameters when you update an existing device.

Prepare devices for multicast and FUOTA by using the API operation

To use multicast groups or to perform FUOTA, configure these parameters by using the [CreateWirelessDevice](#) API operation or the [create-wireless-device](#) CLI command. In addition to specifying the application key and FPorts parameters, make sure that the device profile that's linked to the device supports one or both class B or class C modes.

You can provide an `input.json` file as input to the `create-wireless-device` command.

```
aws iotwireless create-wireless-device \  
  --cli-input-json file://input.json
```

where:

Contents of input.json

```
{
  "Description": "My LoRaWAN wireless device"
  "DestinationName": "IoTWirelessDestination"
  "LoRaWAN": {
    "DeviceProfileId": "ab0c23d3-b001-45ef-6a01-2bc3de4f5333",
    "ServiceProfileId": "fe98dc76-cd12-001e-2d34-5550432da100",
    "FPorts": {
      "ClockSync": 202,
      "Fuota": 201,
      "Multicast": 200
    },
    "OtaaV1_0_x": {
      "AppKey": "3f4ca100e2fc675ea123f4eb12c4a012",
      "AppEui": "b4c231a359bc2e3d",
      "GenAppKey": "01c3f004a2d6efffe32c4eda14bcd2b4"
    },
    "DevEui": "ac12efc654d23fc2"
  },
  "Name": "SampleIoTWirelessThing"
  "Type": LoRaWAN
}
```

For information about the CLI commands that you can use, see [AWS CLI reference](#).

Note

After you specify the values of these parameters, you can't update them by using the `UpdateWirelessDevice` API operation. Instead, you can create a new device with the values for the parameters `GenAppKey` and `FPorts`.

To get information about the values specified for these parameters, you can use the [GetWirelessDevice](#) API operation or the [get-wireless-device](#) CLI command.

Next steps

After you've configured the parameters, you can create multicast groups and FUOTA tasks to send downlink payload or update the firmware of your LoRaWAN devices.

- For information about creating multicast groups, see [Create multicast groups and add devices to the group](#).
- For information about creating FUOTA tasks, see [Create FUOTA task and provide firmware image](#).

Create multicast groups to send a downlink payload to multiple devices

To send a downlink payload to multiple devices, create a multicast group. Using multicast, a source can send data to a single multicast address, which is then distributed to an entire group of recipient devices.

Devices in a multicast group share the same multicast address, session keys, and frame counter. By using the same session keys, devices in a multicast group can decrypt the message when a downlink transmission is initiated. A multicast group only supports downlink. It doesn't confirm whether the downlink payload has been received by the devices.

With AWS IoT Core for LoRaWAN's multicast groups, you can:

- Filter your list of devices by using the device profile, RFRegion, or device class, and then add these devices to a multicast group.
- Schedule and send one or more downlink payload messages to devices in a multicast group, within a 48-hour distribution window.
- Have devices temporarily switch to Class B or class C mode at the start of your multicast session for receiving the downlink message.
- Monitor your multicast group setup and the state of its devices, and also troubleshoot any issues.
- Use Firmware Updates-Over-The-Air (FUOTA) to securely deploy firmware updates to devices in a multicast group.

The following video describes how AWS IoT Core for LoRaWAN multicast groups can be created and walks you through the process of adding a device to the group and schedule a downlink message to the group.

The following shows how to create your multicast group and schedule a downlink message.

Topics

- [Create multicast groups and add devices to the group](#)
- [Monitor and troubleshoot status of your multicast group and devices in the group](#)

- [Schedule a downlink message to send to devices in your multicast group](#)

Create multicast groups and add devices to the group

You can create multicast groups by using the console or the CLI. If you're creating your multicast group for the first time, we recommend that you use the console to add your multicast group. When you want to manage your multicast group and add or remove devices from your group, you can use the CLI.

After exchanging signaling with the end devices you added, AWS IoT Core for LoRaWAN establishes the shared keys with the end devices and sets up the parameters for the data transfer.

Prerequisites

Before you can create multicast groups and add devices to the group:

- Prepare your devices for multicast and FUOTA setup by specifying the FUOTA configuration parameters `GenAppKey` and `FPorts`. For more information, see [Prepare devices for multicast and FUOTA configuration](#).
- Check whether the devices support class B or class C modes of operation. Depending on the device class that your device supports, choose a device profile that has one or both **Supports Class B** or **Supports Class C** modes enabled. For information about device profiles, see [Add profiles to AWS IoT Core for LoRaWAN](#).

At the start of the multicast session, a class B or class C distribution window is used to send downlink messages to the devices in your group.

Create multicast groups by using the console

To create multicast groups by using the console, go to the [Multicast groups](#) page of the AWS IoT console and choose **Create multicast group**.

1. Create a multicast group

To create your multicast group, specify the multicast properties and tags for your group.

1. Specify multicast properties

To specify multicast properties, enter the following information for your multicast group.

- **Name:** Enter a unique name for your multicast group. The name must contain only letters, numbers, hyphens, and underscores. It can't contain spaces.
- **Description:** You can provide an optional description for your multicast group. The description length can be up to 2,048 characters.

2. Tags for multicast group

You can optionally provide any key-value pairs as **Tags** for your multicast group. To continue creating your multicast group, choose **Next**.

2. Add devices to a multicast group

You can add individual devices or a group of devices to your multicast group. To add devices:

1. Specify RFRegion

Specify the **RFRegion** or frequency band for your multicast group. The **RFRegion** for your multicast group must match the **RFRegion** of devices that you add to the multicast group. For more information about the **RFRegion**, see [Consider selection of LoRa frequency bands for your gateways and device connection](#).

2. Select a multicast device class

Choose whether you want devices in the multicast group to switch to a class B or class C mode at the start of the multicast session. A class B session can receive downlink messages at regular downlink slots and a class C session can receive downlink messages at anytime.

3. Choose the devices you want to add to the group

Choose whether you want to add devices individually or in bulk to the multicast group.

- To add devices individually, enter the wireless device ID of each device that you want to add to your group.
- To add devices in bulk, you can filter the devices you want to add by device profile or tags. For device profile, you can add devices with a profile that supports class B, class C, or both device classes.

4. To create your multicast group, choose **Create** .

The multicast group details and the devices you added appear in the group. For information about the status of the multicast group and your devices and for troubleshooting any issues, see [Monitor and troubleshoot status of your multicast group and devices in the group](#).

After creating a multicast group, you can choose **Action** to edit, delete, or add devices to the multicast group. After you've added the devices, you can schedule a session for the downlink payload to be sent to the devices in your group.

Create multicast groups by using the API

To create multicast groups and add devices to the group by using the API:

1. Create a multicast group

To create your multicast group, use the [CreateMulticastGroup](#) API operation or the [create-multicast-group](#) CLI command. You can provide an `input.json` file as input to the `create-multicast-group` command.

```
aws iotwireless create-multicast-group \  
  --cli-input-json file://input.json
```

where:

Contents of `input.json`

```
{  
  "Description": "Multicast group to send downlink payload and perform FUOTA.",  
  "LoRaWAN": {  
    "DlClass": "ClassB",  
    "RfRegion": "US915"  
  },  
  "Name": "MC_group_FUOTA"  
}
```

After you create your multicast group, you can use the following API operations or CLI commands to update, delete, or get information about your multicast groups.

- [UpdateMulticastGroup](#) or [update-multicast-group](#)
- [GetMulticastGroup](#) or [get-multicast-group](#)
- [ListMulticastGroups](#) or [list-multicast-groups](#)
- [DeleteMulticastGroup](#) or [delete-multicast-group](#)

2. Add devices to a multicast group

You can add devices to your multicast group individually or in bulk.

- To add devices in bulk to your multicast group, use the [StartBulkAssociateWirelessDeviceWithMulticastGroup](#) API operation or the [start-bulk-associate-wireless-device-with-multicast-group](#) CLI command. To filter the devices you want to associate in bulk to your multicast group, provide a query string. The following shows how you can add a group of devices that has a device profile with the specified ID linked to it.

```
aws iotwireless start-bulk-associate-wireless-device-with-multicast-group \
  --id "12abd34e-5f67-89c2-9293-593b1bd862e0" \
  --cli-input-json file://input.json
```

where:

Contents of input.json

```
{
  "QueryString": "DeviceProfileName: MyWirelessDevice AND DeviceProfileId:
d6d8ef8e-7045-496d-b3f4-ebcaa1d564bf",
  "Tags": [
    {
      "Key": "Multicast",
      "Value": "ClassB"
    }
  ]
}
```


Here, `multicast-groups/d6d8ef8e-7045-496d-b3f4-ebcaa1d564bf/bulk` is the URL that's used to associate devices with the group.

- To add devices individually to your multicast group, use the [AssociateWirelessDeviceWithMulticastGroup](#) API operation or the [associate-wireless-device-with-multicast-group](#) CLI. Provide the wireless device ID for each device you want to add to your group.

```
aws iotwireless associate-wireless-device-with-multicast-group \
  --id "12abd34e-5f67-89c2-9293-593b1bd862e0" \
  --wireless-device-id "ab0c23d3-b001-45ef-6a01-2bc3de4f5333"
```

After you create your multicast group, you can use the following API operations or CLI commands to get information about your multicast group or to disassociate devices.

- [DisassociateWirelessDeviceFromMulticastGroup](#) or [disassociate-wireless-device-from-multicast-group](#)
- [StartBulkDisassociateWirelessDeviceFromMulticastGroup](#) or [start-bulk-disassociate-wireless-device-from-multicast-group](#)
- [ListWirelessDevices](#) or [list-wireless-devices](#)

 **Note**

The `ListWirelessDevices` API operation can be used to list wireless devices in general, and wireless devices that are associated with a multicast group or a FUOTA task.

- To list wireless devices associated with a multicast group, use the `ListWirelessDevices` API operation with `MulticastGroupID` as the filter.
- To list wireless devices associated with a FUOTA task, use the `ListWirelessDevices` API operation with `FuotaTaskID` as the filter.

Next steps

After you've created a multicast group and added devices, you can continue adding devices and monitor the status of the multicast group and your devices. If your devices have been added successfully to the group, you can configure and schedule a downlink message to be sent to the devices. Before you can send a downlink message, your devices' status must be **Multicast setup ready**. After you schedule a downlink message, the status changes to **Session attempting**. For more information, see [Schedule a downlink message to send to devices in your multicast group](#).

If you want to update the firmware of the devices in the multicast group, you can perform Firmware Updates Over-The-Air (FUOTA) with AWS IoT Core for LoRaWAN. For more information, see [Firmware update over-the-air \(FUOTA\) for AWS IoT Core for LoRaWAN devices](#).

If your devices weren't added or if you see an error in the multicast group or device statuses, you can hover over the error to get more information and resolve it. If you still see an error, for

information about how to troubleshoot and resolve the issue, see [Monitor and troubleshoot status of your multicast group and devices in the group](#).

Monitor and troubleshoot status of your multicast group and devices in the group

After you've added devices and created your multicast group, open the AWS Management Console. Navigate to the [Multicast groups](#) page of the AWS IoT console and choose the multicast group you created to view its details. You'll see information about the multicast group, the number of devices that have been added, and device status details. You can use the status information to track progress of your multicast session and troubleshoot any errors.

Multicast group status

Your multicast group can have one of the following status messages displayed in the AWS Management Console.

- **Pending**

This status indicates that you've created a multicast group but it doesn't yet have a multicast session. You'll see this status message displayed when your group has been created. During this time, you can update your multicast group, and associate or disassociate devices with your group. After the status changes from **Pending**, additional devices can't be added to the group.

- **Session attempting**

After your devices have been added successfully to the multicast group, when your group has a scheduled multicast session, you'll see this status message displayed. During this time, you can't update or add devices to your multicast group. If you cancel your multicast session, the group status changes to **Pending**.

- **In session**

When it's the earliest session time for your multicast session, you'll see this status message displayed. A multicast group also continues to be in this state when it's associated with a FUOTA task that has an ongoing firmware update session.

If you don't have an associated FUOTA task in session, and if the multicast session is canceled because the session time exceeded the time out or you canceled your multicast session, the group status changes to **Pending**.

- **Delete waiting**

If you delete your multicast group, its group status changes to **Delete waiting**. Deletions are permanent and can't be undone. This action can take time and the group status will be **Delete_Waiting** until the multicast group has been deleted. After your multicast group enters this state, it can't transition to one of the other states.

Status of devices in multicast group

The devices in your multicast group can have one of the following status messages displayed in the AWS Management Console. You can hover over each status message to get more information about what it indicates.

- **Package attempting**

After your devices have been associated with the multicast group, the device status is **Package attempting**. This status indicates that AWS IoT Core for LoRaWAN has not yet confirmed whether the device supports multicast setup and operation.

- **Package unsupported**

After your devices have been associated with the multicast group, AWS IoT Core for LoRaWAN checks whether your device's firmware is capable of multicast setup and operation. If your device doesn't have the supported multicast package, its status is **Package unsupported**. To resolve the error, check whether your device's firmware is capable of multicast setup and operation.

- **Multicast setup attempting**

If the devices associated with your multicast group are capable of multicast setup and operation, the status is **Multicast setup attempting**. This status indicates that the device hasn't completed the multicast setup yet.

- **Multicast setup ready**

Your device has completed the multicast setup and has been added to the multicast group. This status indicates that the devices are ready for a multicast session and a downlink message can be sent to those devices. The status also indicates when you can use FUOTA to update the firmware of the devices in the group.

- **Session attempting**

A multicast session has been scheduled for the devices in your multicast group. At the start of a multicast group session, the device status is **Session attempting**, and requests are sent for

whether a class B or class C distribution window can be initiated for the session. If the time it takes to set up the multicast session exceeds the timeout or if you cancel the multicast session, the status changes to **Multicast setup done**.

- **In session**

This status indicates that a class B or class C distribution window has been initiated and your device has an ongoing multicast session. During this time, downlink messages can be sent from AWS IoT Core for LoRaWAN to devices in the multicast group. If you update your session time, it overrides the current session and the status changes to **Session attempting**. When the session time ends or if you cancel the multicast session, the status changes to **Multicast setup ready**.

Next steps

Now that you've learned the different statuses of your multicast group and the devices in your group, and how you can troubleshoot any issues such as when a device is not capable of multicast setup, you can schedule a downlink message to be sent to the devices and your multicast group will be in **In session**. For information about scheduling a downlink message, see [Schedule a downlink message to send to devices in your multicast group](#).

Schedule a downlink message to send to devices in your multicast group

After you've successfully added devices to your multicast group, you can start a multicast session and configure a downlink message to be sent to those devices. The downlink message must be scheduled within 48 hours and the start time for the multicast must be at least 30 minutes later than the current time.

Note

Devices in a multicast group can't acknowledge when a downlink message has been received.

Prerequisites

Before you can send a downlink message, you must have created a multicast group and successfully added devices to the group for which you want to send a downlink message. You can't add more devices after a start time has been scheduled for your multicast session. For more information, see [Create multicast groups and add devices to the group](#).

If any of the devices weren't added successfully, the multicast group and device status will contain information to help you resolve the errors. If the errors still persist, for information about troubleshooting these errors, see [Monitor and troubleshoot status of your multicast group and devices in the group](#).

Schedule a downlink message by using the console

To send a downlink message by using the console, go to the [Multicast groups](#) page of the AWS IoT console and choose the multicast group you created. In the multicast group details page, choose **Schedule downlink message** and then choose **Schedule downlink session**.

1. Schedule downlink message window

You can set up a time window for a downlink message to be sent to the devices in your multicast group. The downlink message must be scheduled within 48 hours.

To schedule your multicast session, specify the following parameters:

- **Start date** and **Start time**: The start date and time must be at least 30 minutes after and 48 hours before the current time.

Note

The time that you specify is in UTC so consider checking the time difference with your time zone when scheduling the downlink window.

- **Session timeout**: The time after which you want the multicast session to timeout if no downlink message has been received. The minimum timeout allowed is 60 seconds. The maximum timeout value is 2 days for class B multicast groups and 18 hours for class C multicast groups.

2. Configure your downlink message

To configure your downlink message, specify the following parameters:

- **Data rate**: Choose a data rate for your downlink message. The data rate depends on RFRegion and payload size. The default data rate is 8 for the US915 region and 0 for the EU868 region.
- **Frequency**: Choose a frequency for sending your downlink message. To avoid messaging conflicts, choose an available frequency depending on the RFRegion.

- **FPort:** Choose an available frequency port for sending the downlink message to your devices.
- **Payload:** Specify the maximum size of your payload depending on the data rate. Using the default data rate, you can have a maximum payload size of 33 bytes in the US915 RfRegion and 51 bytes in the EU868 RfRegion. Using larger data rates, you can transfer up to a maximum payload size of 242 bytes.

To schedule your downlink message, choose **Schedule**.

Schedule a downlink message by using the API

To schedule a downlink message by using the API, use the [StartMulticastGroupSession](#) API operation or the [start-multicast-group-session](#) CLI command.

You can use the following API operations or CLI commands to get information about a multicast group and to delete a multicast group.

- [GetMulticastGroupSession](#) or [get-multicast-group-session](#)
- [DeleteMulticastGroupSession](#) or [delete-multicast-group-session](#)

To send data to a multicast group after the session has been started, use the [SendDataToMulticastGroup](#) API operation or the [send-data-to-multicast-group](#) CLI command.

Next steps

After you've configured a downlink message to be sent to the devices, the message is sent at the start of the session. The devices in a multicast group can't confirm whether the message has been received.

Configure additional downlink messages

You can also configure additional downlink messages to be sent to the devices in your multicast group:

- To configure additional downlink messages from the console:
 1. Go to the [Multicast groups](#) page of the AWS IoT console and choose the multicast group you created.

2. In the multicast group details page, choose **Schedule downlink message** and then choose **Configure additional downlink message**.
 3. Specify the parameters **Data rate**, **Frequency**, **FPort**, and **Payload**, similar to how you configured these parameters for your first downlink message.
- To configure additional downlink messages using the API or CLI, call the [SendDataToMulticastGroup](#) API operation or the [send-data-to-multicast-group](#) CLI command for each additional downlink message.

Update session schedule

You can also update the session schedule to use a new start date and time for your multicast session. The new session schedule will override the previously scheduled session.

Note

Update your multicast session only when required. These updates can cause a group of devices to wake up for a long duration and drain the battery.

- To update the session schedule from the console:
 1. Go to the [Multicast groups](#) page of the AWS IoT console and choose the multicast group you created.
 2. In the multicast group details page, choose **Schedule downlink message** and then choose **Update session schedule**.
 3. Specify the parameters **State date**, **Start time**, and **Session timeout**, similar to how you specified these parameters for your first downlink message.
- To update the session schedule from the API or CLI, use the [StartMulticastGroupSession](#) API operation or the [start-multicast-group-session](#) CLI command.

Firmware update over-the-air (FUOTA) for AWS IoT Core for LoRaWAN devices

Use Firmware Updates Over-The-Air (FUOTA) to deploy firmware updates to AWS IoT Core for LoRaWAN devices.

Using FUOTA, you can send firmware updates to individual devices or to a group of devices. You can also send firmware updates to multiple devices by creating a multicast group. First add your devices to the multicast group, and then send your firmware update image to all those devices. We recommend that you digitally sign the firmware images so that devices receiving the images can verify that they're coming from the right source.

With AWS IoT Core for LoRaWAN's FUOTA, you can:

- Deploy new firmware images or delta images to a single device or a group of devices.
- Verify the authenticity and integrity of new firmware after it's deployed to devices.
- Monitor the progress of a deployment and debug issues in case of a failed deployment.

AWS IoT Core for LoRaWAN's support for FUOTA and multicast groups is based on the [LoRa Alliance's](#) following specifications:

- LoRaWAN Remote Multicast Setup Specification, TS005-2.0.0
- LoRaWAN Fragmented Data Block Transportation Specification, TS004-1.0.0
- LoRaWAN Application Layer Clock Synchronization Specification, TS003-1.0.0

 **Note**

AWS IoT Core for LoRaWAN automatically performs the clock synchronization according to the LoRa Alliance specification. It uses the function `AppTimeReq` to reply the server-side time to the devices that request it using `ClockSync` signaling.

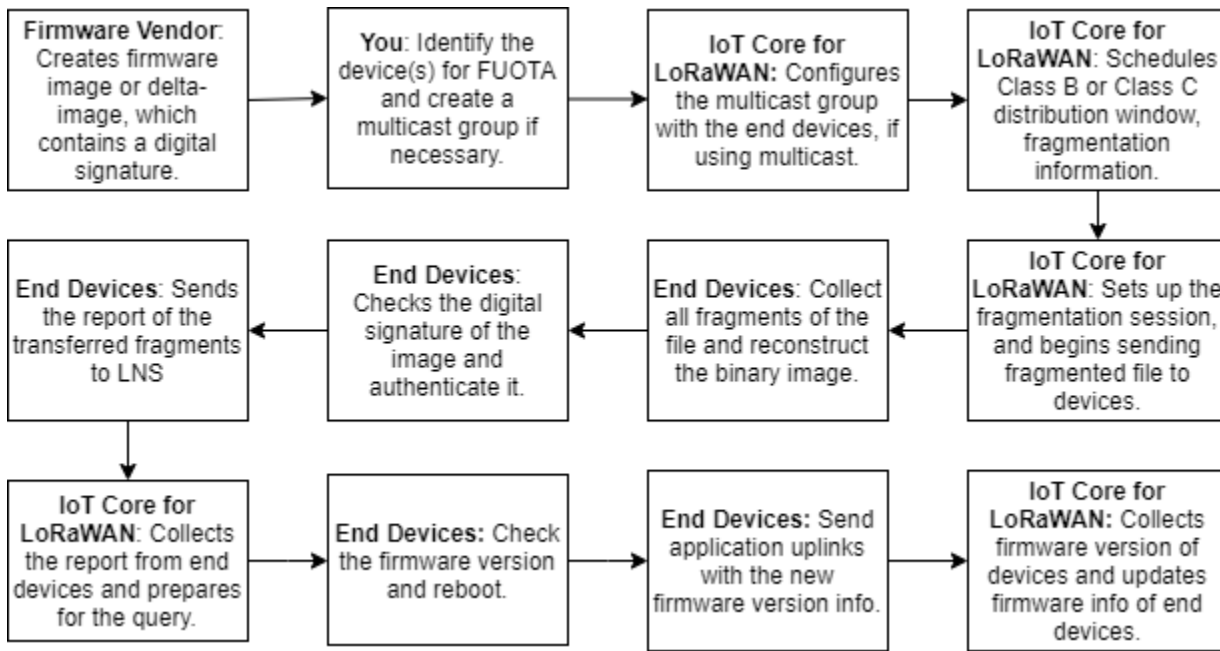
The following video describes how AWS IoT Core for LoRaWAN FUOTA tasks can be created and walks you through the process of adding devices to the task and schedule a FUOTA task.

The following topics show how to perform FUOTA.

- [FUOTA process overview](#)
- [Create FUOTA task and provide firmware image](#)
- [Add devices and multicast groups to a FUOTA task and schedule a FUOTA session](#)
- [Monitor and troubleshoot the status of your FUOTA task and devices added to the task](#)

FUOTA process overview

The following diagram shows how AWS IoT Core for LoRaWAN performs the FUOTA process for your end devices. If you're adding individual devices to your FUOTA session, you can skip the steps for creating and configuring your multicast group. You can add your devices directly to a FUOTA session, and AWS IoT Core for LoRaWAN will then start the firmware update process.



To perform FUOTA for your devices, first create your digitally signed firmware image and configure the devices and multicast groups that you want to add to your FUOTA task. After you start a FUOTA session, your end devices collect all fragments, reconstruct the image from the fragments, report the status to AWS IoT Core for LoRaWAN, and then apply the new firmware image.

The following illustrates the different steps in the FUOTA process:

1. Create a firmware image or delta image with a digital signature

For AWS IoT Core for LoRaWAN to perform FUOTA for your LoRaWAN devices, we recommend that you digitally sign the firmware image or the delta image when sending firmware updates over the air. The devices that receive the images can then verify that it's coming from the right source.

Your firmware image must not be larger than 1 megabyte in size. The larger your firmware size, the longer it can take for your update process to complete. For faster data transfer or if your new image is larger than 1 Megabyte, use a delta image, which is the part of your new image that's the delta between your new firmware image and the previous image.

Note

AWS IoT Core for LoRaWAN doesn't provide the digital signature generation tool and the firmware version management system. You can use any third-party tool to generate the digital signature for your firmware image. We recommend that you use a digital signature tool such as the one embedded in the [ARM Mbed GitHub repository](#), which also includes tools for generating the delta image and for devices to use that image.

2. Identify and configure the devices for FUOTA

After you identify the devices for FUOTA, send firmware updates to individual or multiple devices.

- To send your firmware updates to multiple devices, create a multicast group, and configure the multicast group with end devices. For more information, see [Create multicast groups to send a downlink payload to multiple devices](#).
- To send firmware updates to individual devices, add those devices to your FUOTA session and then perform the firmware update.

3. Schedule a distribution window and set up fragmentation session

If you created a multicast group, you can specify the class B or class C distribution window to determine when the devices can receive the fragments from AWS IoT Core for LoRaWAN. Your devices might be operating in class A before they switch to class B or class C mode. You must also specify the start time of the session.


Class B or class C devices wake up at the specified distribution window and start receiving the downlink packets. Devices operating in class C mode can consume more power than class B devices. For more information, see [Device classes](#).

4. End devices report status to AWS IoT Core for LoRaWAN and update firmware image

After you set up a fragmentation session, your end devices and AWS IoT Core for LoRaWAN perform the following steps to update the firmware of your devices.

1. Because LoRaWAN devices have a low data rate, to start the FUOTA process, AWS IoT Core for LoRaWAN sets up a fragmentation session to fragment the firmware image. Then it sends these fragments to the end devices.

2. After AWS IoT Core for LoRaWAN sends the image fragments, your LoRaWAN end devices perform the following tasks.
 - a. Collect the fragments and then reconstruct the binary image from these fragments.
 - b. Check the digital signature of the reconstructed image to authenticate the image and verify that it's coming from the right source.
 - c. Compare the firmware version from AWS IoT Core for LoRaWAN to the current version.
 - d. Report the status of the fragmented images that were transferred to AWS IoT Core for LoRaWAN, and then apply the new firmware image.

 **Note**

In some cases, the end devices report the status of the fragmented images that were transferred to AWS IoT Core for LoRaWAN before checking the digital signature of the firmware image.

Now that you've learned the FUOTA process, you can create your FUOTA task and add devices to the task for updating their firmware. For more information, see [Create FUOTA task and provide firmware image](#).

Create FUOTA task and provide firmware image

To update the firmware of your LoRaWAN devices, first create a FUOTA task and provide the digitally signed firmware image you want to use for the update. You can then add your devices and multicast groups to the task and schedule a FUOTA session. When the session starts, AWS IoT Core for LoRaWAN sets up a fragmentation session and your end devices collect the fragments, reconstruct the image, and apply the new firmware. For information about the FUOTA process, see [FUOTA process overview](#).

The following shows how you can create a FUOTA task and upload the firmware image or delta image that you'll store in an S3 bucket.

Prerequisites

Before you can perform FUOTA, the firmware image must be digitally signed so that your end devices can verify the authenticity of the image when applying the image. You can use any third-party tool to generate the digital signature for your firmware image. We recommend that you use

a digital signature tool such as the one embedded in the [ARM Mbed GitHub repository](#), which also includes tools for generating the delta image and for devices to use that image.

Create FUOTA task and upload firmware image by using the console

To create a FUOTA task and upload your firmware image by using the console, go to the [FUOTA tasks](#) tab of the console and then choose **Create FUOTA task**.

1. Create FUOTA task

To create your FUOTA task, specify the task properties and tags.

1. Specify FUOTA task properties

To specify FUOTA task properties, enter the following information for your FUOTA task.

- **Name:** Enter a unique name for your FUOTA task. The name must contain only letters, numbers, hyphens, and underscores. It can't contain spaces.
- **Description:** You can provide an optional description for your multicast group. The description field can be up to 2,048 characters.
- **RFRegion:** Set the frequency band for your FUOTA task. The frequency band must match the one you used to provision your wireless devices or multicast groups.

2. Tags for FUOTA task

You can optionally provide any key-value pairs as **Tags** for your FUOTA task. To continue creating your task, choose **Next**.

2. Upload firmware image

Choose the firmware image file that you want to use to update the firmware of the devices you add to the FUOTA task. The firmware image file is stored in an S3 bucket. You can provide AWS IoT Core for LoRaWAN the permissions to access the firmware image on your behalf. We recommend that you digitally sign the firmware images so that its authenticity is verified when the firmware update is performed.

1. Choose firmware image file

You can either upload a new firmware image file to an S3 bucket or choose an existing image that has already been uploaded to an S3 bucket.

Note

The firmware image file must not be larger than 1 megabyte in size. The larger your firmware size, the longer it can take for your update process to complete.

- To use an existing image, choose **Select an existing firmware image**, choose **Browse S3**, and then choose the firmware image file you want to use.

AWS IoT Core for LoRaWAN populates the S3 URL, which is the path to your firmware image file in the S3 bucket. The format of the path is `s3://bucket_name/file_name`. To view the file in the [Amazon Simple Storage Service](#) console, choose **View**.

- To upload a new firmware image.
 - a. Choose **Upload a new firmware image** and upload your firmware image. The image file must not be larger than 1 megabyte.
 - b. To create an S3 bucket and enter a **Bucket name** for storing your firmware image file, choose **Create S3 bucket**.

2. Permissions to access the bucket

You can either create a new service role or choose an existing role to allow AWS IoT Core for LoRaWAN to access the firmware image file in the S3 bucket on your behalf. Choose **Next**.

To create a new role, you can enter a role name or leave it blank for a random name to be generated automatically. To view the policy permissions that grant access to the S3 bucket, choose **View policy permissions**.

For more information about using an S3 bucket to store your image and granting AWS IoT Core for LoRaWAN permissions to access it, see [Upload the firmware file to an S3 bucket and add an IAM role](#).

3. Review and create

To create your FUOTA task, review the FUOTA task and configuration details that you specified and the choose **Create task**.

Create FUOTA task and upload firmware image by using the API

To create a FUOTA task and specify your firmware image file by using the API, use the [CreateFuotaTask](#) API operation or the [create-fuota-task](#) CLI command. You can provide an `input.json` file as input to the `create-fuota-task` command. When you use the API or CLI, the firmware image file that you provide as input must be already uploaded to an S3 bucket. You also specify the IAM role that gives AWS IoT Core for LoRaWAN access to the firmware image in the S3 bucket.

```
aws iotwireless create-fuota-task \  
  --cli-input-json file://input.json
```

where:

Contents of `input.json`

```
{  
  "Description": "FUOTA task to update firmware of devices in multicast group.",  
  "FirmwareUpdateImage": "S3:/firmware_bucket/firmware_image",  
  "FirmwareUpdateRole": "arn:aws:iam::123456789012:role/service-role/ACF1zBEI",  
  "LoRaWAN": {  
    "RfRegion": "US915"  
  },  
  "Name": "FUOTA_Task_MC"  
}
```

After you create your FUOTA task, you can use the following API operations or CLI commands to update, delete, or get information about your FUOTA task.

- [UpdateFuotaTask](#) or [update-fuota-task](#)
- [GetFuotaTask](#) or [get-fuota-task](#)
- [ListFuotaTasks](#) or [list-fuota-tasks](#)
- [DeleteFuotaTask](#) or [delete-fuota-task](#)

Next steps

Now that you've created a FUOTA task and provided the firmware image, you can add devices to the task for updating their firmware. You can either add individual devices or multicast groups

to the task. For more information, see [Add devices and multicast groups to a FUOTA task and schedule a FUOTA session](#).

Add devices and multicast groups to a FUOTA task and schedule a FUOTA session

After you've created a FUOTA task, you can add devices to your task for which you want to update the firmware. After your devices have been added successfully to the FUOTA task, you can schedule a FUOTA session to update the device firmware.

- If you have only a small number of devices, you can add those devices directly to your FUOTA task.
- If you have a large number of devices that you want to update firmware for, you can add these devices to your multicast groups, and then add the multicast groups to your FUOTA task. For information about creating and using multicast groups, see [Create multicast groups to send a downlink payload to multiple devices](#).

Note

You can add either individual devices or multicast groups to the FUOTA task. You can't add both devices and multicast groups to the task.

After you've added your devices or multicast groups, you can start a firmware update session. AWS IoT Core for LoRaWAN collects the firmware image, fragments the images, and then stores the fragments in an encrypted format. Your end devices collect the fragments and apply the new firmware image. The time that it takes for the firmware update depends on the image size and how the images were fragmented. After the firmware update is complete, the encrypted fragments of the firmware image stored by AWS IoT Core for LoRaWAN is deleted. You can still find the firmware image in the S3 bucket.

Prerequisites

Before you can add devices or multicast groups to your FUOTA task, do the following.

- You must have already created the FUOTA task and provided your firmware image. For more information, see [Create FUOTA task and provide firmware image](#).

- Provision the wireless devices that you want to update the device firmware for. For more information about onboarding your device, see [Onboard your devices to AWS IoT Core for LoRaWAN](#).
- To update the firmware of multiple devices, you can add them to a multicast group. For more information, see [Create multicast groups to send a downlink payload to multiple devices](#).
- When you onboard the devices to AWS IoT Core for LoRaWAN, specify the FUOTA configuration parameter `FPorts`. If you're using a LoRaWAN v1.0.x device, you must also specify the `GenAppKey`. For more information about the FUOTA configuration parameters, see [Prepare devices for multicast and FUOTA configuration](#).

Add devices to a FUOTA task and schedule a FUOTA session by using the console

To add devices or multicast groups and schedule a FUOTA session by using the console, go to the [FUOTA tasks](#) tab of the console. Then, choose the FUOTA task that you want to add devices to and perform the firmware update.

Add devices and multicast groups

1. You can add either individual devices or multicast groups to your FUOTA task. However, you can't add both individual devices and multicast groups to the same FUOTA task. To add devices using the by console, do the following.
 1. In the **FUOTA task details**, choose **Add device**.
 2. Choose the frequency band or **RFRegion** for the devices you add to the task. This value must match the **RFRegion** that you chose for the FUOTA task.
 3. Choose whether you want to add individual devices or multicast groups to the task.
 - To add individual devices, choose **Add individual devices** and enter the device ID of each device that you want to add to your FUOTA task.
 - To add multicast groups, choose **Add multicast groups** and add your multicast groups to the task. You can filter the multicast groups you want to add to the task by using the device profile or tags. When you filter by device profile, you can choose multicast groups that with devices that have a profile with **Supports Class B** or **Supports Class C** enabled.
2. **Schedule FUOTA session**

After your devices or multicast groups have been added successfully, you can schedule a FUOTA session. To schedule a session, do the following.

1. Choose the FUOTA task for which you want to update the device firmware and then choose **Schedule FUOTA session**.
2. Specify a **Start date** and **Start time** for your FUOTA session. Make sure that the start time is 30 minutes or later from the current time.

Add devices to a FUOTA task and schedule a FUOTA session by using the API

You can use the AWS IoT Wireless API or the CLI to add your wireless devices or multicast groups to your FUOTA task. You can then schedule a FUOTA session.

1. Add devices and multicast groups

You can associate either wireless devices or multicast groups with your FUOTA task.

- To associate individual devices to your FUOTA task, use the [AssociateWirelessDeviceWithFuotaTask](#) API operation or the [associate-wireless-device-with-fuota-task](#) CLI command, and provide the `WirelessDeviceID` as input.


```
aws iotwireless associate-wireless-device-with-fuota-task \  
  --id "01a23cde-5678-4a5b-ab1d-33456808ecb2" \  
  --wireless-device-id "ab0c23d3-b001-45ef-6a01-2bc3de4f5333"
```

- To associate multicast groups to your FUOTA task, use the [AssociateMulticastGroupWithFuotaTask](#) API operation or the [associate-multicast-group-with-fuota-task](#) CLI command, and provide the `MulticastGroupID` as input.

```
aws iotwireless associate-multicast-group-with-FUOTA-task \  
  --id 01a23cde-5678-4a5b-ab1d-33456808ecb2" \  
  --multicast-group-id
```

After you've associated your wireless devices or multicast group to your FUOTA task, use the following API operations or CLI commands to list your devices or multicast groups or to disassociate them from your task.

- [DisassociateWirelessDeviceFromFuotaTask](#) or [disassociate-wireless-device-from-fuota-task](#)
- [DisassociateMulticastGroupFromFuotaTask](#) or [disassociate-multicast-group-from-fuota-task](#)
- [ListWirelessDevices](#) or [list-wireless-devices](#)
- [ListMulticastGroups](#) or [list-multicast-groups-by-fuota-task](#)

 **Note**

The API:

- `ListWirelessDevices` can list wireless devices in general, and devices associated with a multicast group, when `MulticastGroupID` is used as the filter. The API lists wireless devices associated with a FUOTA task when `FuotaTaskID` is used as the filter.
- `ListMulticastGroups` can list multicast groups in general and multicast groups associated with a FUOTA task when `FuotaTaskID` is used as the filter.

2. Schedule FUOTA session

After your devices or multicast groups have been successfully added to the FUOTA task, you can start a FUOTA session to update the device firmware. The start time must be 30 minutes or later from the current time. To schedule a FUOTA session by using the API or CLI, use the [StartFuotaTask](#) API operation or the [start-fuota-task](#) CLI command.

After you've started a FUOTA session, You can no longer add devices or multicast groups to the task. You can get information about the status of your FUOTA session by using the [GetFuotaTask](#) API operation or the [get-fuota-task](#) CLI command.

Monitor and troubleshoot the status of your FUOTA task and devices added to the task

After you have provisioned the wireless devices and created any multicast groups that you might want to use, you can start a FUOTA session by performing the following steps.

FUOTA task status

Your FUOTA task can have one of the following status messages displayed in the AWS Management Console.

- **Pending**

This status indicates that you've created a FUOTA task, but it doesn't yet have a firmware update session. You'll see this status message displayed when your task has been created. During this time, you can update your FUOTA task, and associate or disassociate devices or multicast groups with your task. After the status changes from **Pending**, additional devices can't be added to the task.

- **FUOTA session waiting**

After your devices have been added successfully to the FUOTA task, when your task has a scheduled firmware update session, you'll see this status message displayed. During this time, you can't update or add devices to your FUOTA session. If you cancel your FUOTA session, the group status changes to **Pending**.

- **In FUOTA session**

When your FUOTA session begins, you'll see this status message displayed. The fragmentation session starts and your end devices collect the fragments, reconstruct the firmware image, compare the new firmware version with the original version, and apply the new image.

- **FUOTA done**

After your end devices report to AWS IoT Core for LoRaWAN that the new firmware image has been applied, or when the session times out, the FUOTA session is marked as done, and you'll see this status displayed.

You'll also see this status displayed in any of the following cases so be sure to check whether the firmware update was applied correctly to the devices.

- When the FUOTA task status was **FUOTA session waiting**, and there's an S3 bucket error, such as the link to the image file in the S3 bucket is incorrect or AWS IoT Core for LoRaWAN doesn't have sufficient permissions to access the file in the bucket.
- When the FUOTA task status was **FUOTA session waiting**, and there's a request to start a FUOTA session, but a response isn't received from the devices or multicast groups in your FUOTA task.

- When the FUOTA task status was **In FUOTA session**, and the devices or multicast groups haven't sent any fragments for a certain time period, which results in the session to timeout.
- **Delete waiting**

If you delete your FUOTA task that's in any of the other states, you'll see this status displayed. A deletion action is permanent and can't be undone. This action can take time and the task status will be **Delete waiting** until the FUOTA task has been deleted. After your FUOTA task enters this state, it can't transition to one of the other states.

Status of devices in a FUOTA task

The devices in your FUOTA task can have one of the following status messages displayed in the AWS Management Console. You can hover over each status message to get more information about what it indicates.

- **Initial**

When it's the start time of your FUOTA session, AWS IoT Core for LoRaWAN checks whether your device has the supported package for the firmware update. If your device has the supported package, the FUOTA session for the device starts. The firmware image is fragmented and the fragments are sent to your device. When you see this status displayed, it indicates that the FUOTA session for the device hasn't started yet.

- **Package unsupported**

If the device doesn't have the supported FUOTA package, you'll see this status displayed. If the firmware update package isn't supported, the FUOTA session for your device can't start. To resolve this error, check whether your device's firmware can receive firmware updates using FUOTA.

- **Fragmentation algorithm unsupported**

At the start of your FUOTA session, AWS IoT Core for LoRaWAN sets up a fragmentation session for your device. If you see this status displayed, it means that the type of fragmentation algorithm used can't be applied for your device's firmware update. The error occurs because your device doesn't have the supported FUOTA package. To resolve this error, check whether your device's firmware can receive firmware updates using FUOTA.

- **Not enough memory**

After AWS IoT Core for LoRaWAN sends the image fragments, your end devices collect the image fragments and reconstruct the binary image from these fragments. This status is displayed when your device doesn't have enough memory to assemble the incoming fragments of the firmware image, which can result in your firmware update session ending prematurely. To resolve the error, check whether your device's hardware can receive this update. If your device can't receive this update, use a delta image to update the firmware.

- **Fragmentation index unsupported**

The fragmentation index identifies one of the four simultaneously possible fragmentation sessions. If your device doesn't support the indicated fragmentation index value, this status is displayed. To resolve this error, do one or more of the following.

- Start a new FUOTA task for the device.
- If the error persists, switch from unicast to multicast mode.
- If the error still isn't resolved, check your device firmware.

- **Memory error**

This status indicates that your device has experienced a memory error when receiving the incoming fragments from AWS IoT Core for LoRaWAN. If this error occurs, your device might not be capable of receiving this update. To resolve the error, check whether your device's hardware can receive this update. If needed, use a delta image to update the device firmware.

- **Wrong descriptor**

Your device doesn't support the indicated descriptor. The descriptor is a field that describes the file that will be transported during the fragmentation session. If you see this error, contact [AWS Support Center](#).

- **Session count replay**

This status indicates that your device has previously used this session count. To resolve the error, start a new FUOTA task for the device.

- **Missing fragments**

As your device collects the image fragments from AWS IoT Core for LoRaWAN, it reconstructs the new firmware image from the independent, coded fragments. If your device hasn't received all the fragments, the new image can't be reconstructed, and you'll see this status. To resolve the error, start a new FUOTA task for the device.

- **MIC error**

When your device reconstructs the new firmware image from the collected fragments, it performs a MIC (Message Integrity Check) to verify the authenticity of your image and whether it's coming from the right source. If your device detects a mismatch in the MIC after reassembling the fragments, this status is displayed. To resolve the error, start a new FUOTA task for the device.

- **Device exists in conflict FUOTA task**

If a device has already used in another conflict FUOTA task, then it will generate an error when retrying the new FUOTA task. For example, if a device is part of a multicast group, and it has already been used with another FUOTA task as part of a multicast group, the device will exist in this conflict status as it retries to run the new FUOTA task.

- **Successful**

The FUOTA session for your device was successful.

Note

While this status message indicates that the devices have reconstructed the image from the fragments and verified it, the device firmware might not have been updated when the device reports the status to AWS IoT Core for LoRaWAN. Check whether your device firmware has been updated.

Next steps

You've learned about the different statuses of the FUOTA task and its devices and how you can troubleshoot any issues. For more information about each of these statuses, see the [LoRaWAN Fragmented Data Block Transportation Specification, TS004-1.0.0](#).

Monitoring your wireless resource fleet in real time using network analyzer

Network analyzer uses a default WebSocket connection to receive real-time trace message logs for your wireless connectivity resources. By using network analyzer, you can add the resources you want to monitor, activate a trace messaging session, and start receiving trace messages in real time.

To monitor your resources, you can also use Amazon CloudWatch. To use CloudWatch, you set up an IAM role to configure logging and then wait for the log entries to be displayed in the console. Network analyzer significantly reduces the time that it takes to set up a connection and start receiving trace messages, providing you with just-in-time log information for your fleet of resources. For information about monitoring by using CloudWatch, see [Monitoring your AWS IoT Wireless resources using Amazon CloudWatch Logs](#).

By reducing your setup time and using the information from the trace messages, you can monitor your resources more effectively, get meaningful insights, and troubleshoot errors. You can monitor both LoRaWAN devices and LoRaWAN gateways. For example, you can quickly identify a join error when onboarding one of your LoRaWAN devices. To debug the error, use the information in the provided trace message log.

How to use network analyzer

To monitor your resource fleet and start receiving trace messages, perform the following steps

1. Create network analyzer configuration and add resources

Before you can activate trace messaging, create a network analyzer configuration and add resources to your configuration. First, specify the configuration settings, which include log levels and wireless device frame information. Then, add the wireless resources you want to monitor by using the wireless gateway and wireless device identifiers.

2. Stream trace messages with WebSockets

You can generate a presigned request URL using the credentials for your IAM role to stream network analyzer trace messages by using the WebSocket protocol.

3. Activate trace messaging session and monitor trace messages

To start receiving trace messages, activate your trace messaging session. To avoid incurring additional costs, you can either deactivate or close your network analyzer trace messaging session.

The following video describes how AWS IoT Core for LoRaWAN network analyzer works and walks you through the process of adding resources and tracing join activities using network analyzer.

The following topics show how to create your configuration, add resources, and activate your trace messaging session.

Topics

- [Add necessary IAM role for network analyzer](#)
- [Create a network analyzer configuration and add resources](#)
- [Stream network analyzer trace messages with WebSockets](#)
- [View and monitor network analyzer trace message logs in real time](#)
- [Debug and troubleshoot your multicast groups and FUOTA tasks using network analyzer](#)

Add necessary IAM role for network analyzer

When you use network analyzer, you must grant a user permission to use the API operations [UpdateNetworkAnalyzerConfiguration](#) and [GetNetworkAnalyzerConfiguration](#) to access network analyzer resources. The following shows the IAM policies that you use to grant permissions.

IAM policies for network analyzer

Use either of the following:

- **Full access wireless policy**

Grant AWS IoT Core for LoRaWAN the full access policy by attaching the policy **AWSIoTWirelessFullAccess** to your role. For more information, see [AWSIoTWirelessFullAccess policy summary](#).

- **Scoped IAM policy for Get and Update API**

Create the following IAM policy by going to the [Create policy](#) page of the IAM console, and on the **Visual editor** tab:

1. Choose **IoTWireless** for **Service**.
2. Under **Access level**, expand **Read** and choose **GetNetworkAnalyzerConfiguration**, and then expand **Write** and choose **UpdateNetworkAnalyzerConfiguration**.
3. Choose **Next:Tags**, and enter a **Name** for the policy, such as **IoTWirelessNetworkAnalyzerPolicy**. Choose **Create policy**.

The following shows the policy **IoTWirelessNetworkAnalyzerPolicy** that you created. For more information about creating a policy, see [Create IAM policies](#).

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

```

    "Statement": [
      {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
          "iotwireless:GetNetworkAnalyzerConfiguration",
          "iotwireless:UpdateNetworkAnalyzerConfiguration"
        ],
        "Resource": "*"
      }
    ]
  }
}

```

Scoped policy to access specific resources

To configure more fine-grained access control, you must add the wireless gateways and devices to the **Resource** field. The following policy uses the wildcard ARN to grant access to all gateways and devices. You can control access to specific gateways and devices by using the `WirelessGatewayId` and `WirelessDeviceId`.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iotwireless:GetNetworkAnalyzerConfiguration",
        "iotwireless:UpdateNetworkAnalyzerConfiguration"
      ],
      "Resource": [
        "arn:aws:iotwireless:*:{accountId}:WirelessDevice/*",
        "arn:aws:iotwireless:*:{accountId}:WirelessGateway/*",
        "arn:aws:iotwireless:*:{accountId}:NetworkAnalyzerConfiguration/*"
      ]
    }
  ]
}

```

To grant a user permission to use network analyzer but not to use any wireless gateways or devices, use the following policy. Unless specified, permissions to use the resources are implicitly denied.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "iotwireless:GetNetworkAnalyzerConfiguration",
        "iotwireless:UpdateNetworkAnalyzerConfiguration"
      ],
      "Resource": [
        "arn:aws:iotwireless:*:{accountId}:NetworkAnalyzerConfiguration/*"
      ]
    }
  ]
}
```

Next steps

Now that you've created the policy, you can add resources to your network analyzer configuration and receive trace messaging information for those resources. For more information, see [Create a network analyzer configuration and add resources](#).

Create a network analyzer configuration and add resources

Before you can stream trace messages, create a network analyzer configuration and add the resources you want to monitor to this configuration. When you create a configuration, you can:

- Specify a configuration name and optional description.
- Customize the configuration settings such as frame info and level of detail for your log messages.
- Add the resources that you want to monitor. The resources can be wireless devices or wireless gateways, or both.

The configuration settings that you specify will determine the trace messaging information that you'll receive for resources you add to the configuration. You may also want to create multiple configurations depending on your monitoring use case.

The following shows how to create a configuration and add resources.

Topics

- [Create a network analyzer configuration](#)
- [Add resources and update the network analyzer configuration](#)

Create a network analyzer configuration

Before you can monitor your wireless gateways or wireless devices, you must create a network analyzer configuration. When creating the configuration, you only need to specify a configuration name. You can customize your configuration settings and add the resources that you want to monitor to your configuration even after it's created. The configuration settings determine the trace messaging information that you'll receive for those resources.

Depending on the resources you want to monitor and the level of information you want to receive for them, you may want to create multiple configurations. For example, you can create a configuration that displays only error information for a set of gateways in your AWS account. You can also create a configuration that displays all information about a wireless device that you want to monitor.

The following sections show the various configuration settings and how to create your configuration.

Configuration settings

When creating or updating your network analyzer configuration, you can also customize the following parameters to filter the log stream information.

- **Frame info**

This setting is the frame info for your wireless device resources for trace messages. The frame info can be used to debug the communication between your network server and the end devices. It is enabled by default.

- **Log levels**

You can view Info or Error logs, or you can turn off logging.

- **Info**

Logs with a log level of **Info** are more verbose and contain both error log streams and informational log streams. The informational logs can be used to view changes to the state of a device or gateway.

Note

Collecting more verbose log streams can incur additional costs. For more information about pricing, see [AWS IoT Core pricing](#).

- **Error**

Logs with a log level of **Error** are less verbose and display only error information. You can use these logs when an application has an error, such as a device connection error. By using the information from the log stream, you can identify and troubleshoot errors for resources in your fleet.

Create a configuration using the console

You can create a network analyzer configuration and customize the optional parameters using the AWS IoT console or the AWS IoT Wireless API. You can also create multiple configurations and later delete any configurations that you're no longer using.

Create a network analyzer configuration

1. Open the [Network Analyzer hub of the AWS IoT console](#) and choose **Create configuration**.
2. Specify the configuration settings.

- **Name, description, and tags**

Specify a unique **Configuration name** that only contains letters, numbers, hyphens, or underscores. Use the optional **Description** field to provide information about the configuration, and the **Tags** field to add key-value pairs of metadata about the configuration. For more information about naming and describing your resources, see [Describing your AWS IoT Wireless resources](#).

- **Configuration settings**

Choose whether to disable frame info, and use **Select log levels** to choose the log levels that you want to use for your trace message logs. Choose **Next**.

3. Add resources to your configuration. You can either add your resources now or choose **Create** and then add your resources later. To add resources later, choose **Create**.

In the **Network Analyzer hub page**, you'll see the configuration that you created along with its settings. To view the details of the new configuration, choose the configuration name.

Delete your network analyzer configuration

You can create multiple network analyzer configurations depending on the resources you want to monitor and the level of trace messaging information that you want to receive for them.

To remove configurations from the console

1. Go to the [Network Analyzer hub of the AWS IoT console](#) and choose the configuration that you want to remove.
2. Choose **Actions**, and then choose **Delete**.

Create a configuration using the API

To create a network analyzer configuration using the API, use the [CreateNetworkAnalyzerConfiguration](#) API operation or the [create-network-analyzer-configuration](#) CLI command.

When you create your configuration, you only need to specify a configuration name. You can also use this API operation to specify the configuration settings and add resources when creating the configuration. Alternatively, you can specify them later by using the [UpdateNetworkAnalyzerConfiguration](#) API operation or the [update-network-analyzer-configuration](#) CLI.

- **Create a configuration**

When you create your configuration, you must specify a name. For example, the following command creates a configuration by providing only a name and an optional description. By default, the configuration has frame info activated and uses a log level of INFO.

```
aws iotwireless create-network-analyzer-configuration \  
  --configuration-name My_Network_Analyzer_Config \  
  --description "My first network analyzer configuration"
```

Running this command displays the ARN and ID of your network analyzer configuration.

```
{
  "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:NetworkAnalyzerConfiguration/12345678-a1b2-3c45-67d8-
e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

- **Create configuration with resources**

To customize the configuration settings, use the `trace-content` parameter. To add resources, use the `WirelessDevices` and `WirelessGateways` parameters to specify the gateways, devices, or both that you want to add to your configuration. For example, the following command customizes the configuration settings and adds to your configuration the wireless resources, specified by their `WirelessGatewayID` and `WirelessDeviceID`.

```
aws iotwireless create-network-analyzer-configuration \
  --configuration-name My_NetworkAnalyzer_Config \
  --trace-content WirelessDeviceFrameInfo=DISABLED,LogLevel="ERROR" \
  --wireless-gateways "12345678-a1b2-3c45-67d8-e90fa1b2c34d" "90123456-
de1f-2b3b-4c5c-bb1112223cd1"
  --wireless-devices "1ffd32c8-8130-4194-96df-622f072a315f"
```

The following example shows the output of running the command:

```
{
  "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:NetworkAnalyzerConfiguration/12345678-a1b2-3c45-67d8-
e90fa1b2c34d",
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

List network analyzer configurations

You can create multiple network analyzer configurations depending on the resources that you want to monitor and the level of detail of trace messaging information that you want to receive for the resources. After you create these configurations, you can use the [ListNetworkAnalyzerConfigurations](#) API operation or the [list-network-analyzer-configuration](#) CLI command to get a list of these configurations.

```
aws iotwireless list-network-analyzer-configurations
```

Running this command displays all the network analyzer configurations in your AWS account. You can also use the `max-results` parameter to specify how many configurations you want to display. The following shows the output of running this command.

```
{
  "NetworkAnalyzerConfigurationList": [
    {
      "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:NetworkAnalyzerConfiguration/12345678-a1b2-3c45-67d8-e90fa1b2c34d",
      "Name": "My_Network_Analyzer_Config1"
    },
    {
      "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:NetworkAnalyzerConfiguration/90123456-a1a2-9a87-65b4-c12bf3c2d09a",
      "Name": "My_Network_Analyzer_Config2"
    }
  ]
}
```

Delete your network analyzer configuration

You can delete a configuration that you're no longer using with the [DeleteNetworkAnalyzerConfiguration](#) API operation or the [delete-network-analyzer-configuration](#) CLI command.

```
aws iotwireless delete-network-analyzer-configuration \
  --configuration-name My_NetworkAnalyzer_Config
```

Running this command doesn't produce any output. To see the available configurations, you can use the `ListNetworkAnalyzerConfigurations` API operation.

Next steps

Now that you've created a network analyzer configuration, you can add resources to your configuration or update your configuration settings. For more information, see [Add resources and update the network analyzer configuration](#).

Add resources and update the network analyzer configuration

Before you can activate trace messaging, you must add resources to your configuration. You can use only a single, default network analyzer configuration. AWS IoT Core for LoRaWAN assigns the name, **NetworkAnalyzerConfig_Default**, to this configuration, and this field can't be edited. This configuration is automatically added to your AWS account when you use network analyzer from the console.

You can add the resources that you want to monitor to this default configuration. Resources can be either or both LoRaWAN devices and LoRaWAN gateways. To add each individual resource to the configuration, use the wireless gateway and wireless device identifiers.

Configuration settings

To configure settings, first add resources to your default configuration and activate trace messaging. After you've received the trace message logs, you can also customize the following parameters to update your default configuration and filter the log stream.

- **Frame info**

This setting is the frame info of your wireless device resources for trace messages. The frame info is enabled by default, and can be used to debug the communication between your network server and the end devices.

- **Log levels**

You can view Info or Error logs, or you can turn off logging.

- **Info**

Logs with a log level of **Info** are more verbose and contain log streams that are both informative and contain errors. The informative logs can be used to view changes to the state of a device or gateway.

 **Note**

Collecting more verbose log streams can incur additional costs. For more information about pricing, see [AWS IoT Core pricing](#).

- **Error**

Logs with a log level of **Error** are less verbose and display only error information. You can use these logs when an application has an error, such as a device connection error. By using the information from the log stream, you can identify and troubleshoot errors for resources in your fleet.

Prerequisites

Before you can add resources, you must have onboarded the gateways and devices that you want to monitor to AWS IoT Core for LoRaWAN. For more information, see [Connecting gateways and devices to AWS IoT Core for LoRaWAN](#).

Add resources and update the network analyzer configuration by using the console

You can add resources and customize the optional parameters by using the AWS IoT console or the AWS IoT Wireless API. In addition to resources, you can also edit your configuration settings and save the updated configuration.

To add resources to your configuration (console)

1. Open the [Network Analyzer hub of the AWS IoT console](#) and choose the network analyzer configuration, **NetworkAnalyzerConfig_Default**.
2. Choose **Add resources**.
3. Add the resources you want to monitor by using the wireless gateway and wireless device identifiers. You can add up to 250 wireless gateways or wireless devices. To add your resource:
 - a. Use the **View gateways** or **View devices** tab to see the list of gateways and devices that you've added to your AWS account.
 - b. Copy the `WirelessDeviceID` or the `WirelessGatewayID` of the device or gateway that you want to monitor and enter the identifier value for the corresponding resource.
 - c. To continue adding resources, choose **Add gateway** or **Add device** and add your wireless gateway or device. If you added a resource that you no longer want to monitor, choose **Remove resource**.
4. After you've added all the resources, choose **Add**.

You'll see the number of gateways and devices that you added in the **Network Analyzer hub page**. You can still continue adding gateways and devices until you activate the trace messaging

session. After the session has been activated, to add resources, you'll have to deactivate the session.

To edit the network analyzer configuration (console)

You can also edit the network analyzer configuration and choose whether to disable frame info and the log level for your trace message logs.

1. Open the [Network Analyzer hub of the AWS IoT console](#) and choose the network analyzer configuration, **NetworkAnalyzerConfig_Default**.
2. Choose **Edit**.
3. Choose whether to disable frame info and use **Select log levels** to choose the log levels that you want to use for your trace message logs. Choose **Save**.

You'll see the configuration settings that you specified in the details page of your network analyzer configuration.

Add resources and update the network analyzer configuration by using the API

You can use the [AWS IoT Wireless API operations](#) or the [AWS IoT Wireless CLI commands](#) to add resources and update the configuration settings for your network analyzer configuration.

- To add resources and update your network analyzer configuration, use the [UpdateNetworkAnalyzerConfiguration](#) API or the [update-network-analyzer-configuration](#) CLI.

- **Add resources**

For the wireless devices you want to add, use `WirelessDevicesToAdd` to enter the `WirelessDeviceID` for the devices as an array of strings. For the wireless gateways you want to add, use `WirelessGatewaysToAdd` to enter the `WirelessGatewayID` for the gateways as an array of strings.

- **Edit configuration**

To edit your network analyzer configuration, use the `TraceContent` parameter to specify whether `WirelessDeviceFrameInfo` should be `ENABLED` or `DISABLED`, and whether the `LogLevel` parameter should be `INFO`, `ERROR`, or `DISABLED`.

```
{  
  "TraceContent": {
```

```
    "LogLevel": "string",
    "WirelessDeviceFrameInfo": "string"
  },
  "WirelessDevicesToAdd": [ "string" ],
  "WirelessDevicesToRemove": [ "string" ],
  "WirelessGatewaysToAdd": [ "string" ],
  "WirelessGatewaysToRemove": [ "string" ]
}
```

- To get information about the configuration and the resources that you've added, use the [GetNetworkAnalyzerConfiguration](#) API operation or the [get-network-analyzer-configuration](#) command. Provide the name of the network analyzer configuration, `NetworkAnalyzerConfig_Default`, as input.

Next steps

Now that you've added resources and specified any optional configuration settings for your configuration, you can use the WebSocket protocol to establish a connection with AWS IoT Core for LoRaWAN for using network analyzer. You can then activate trace messaging and start receiving trace messages for your resources. For more information, see [Stream network analyzer trace messages with WebSockets](#).

Stream network analyzer trace messages with WebSockets

When you use the WebSocket protocol, you can stream network analyzer trace messages in real time. When you send a request, the service responds with a JSON structure. After you activate trace messaging, you can use the message logs to get information about your resources and troubleshoot errors. For more information, see [WebSocket protocol](#).

The following shows how to stream network analyzer trace messages with WebSockets.

Topics

- [Generate a presigned request with the WebSocket library](#)
- [Sample Python code to generate presigned URL](#)
- [WebSocket messages and status codes](#)

Generate a presigned request with the WebSocket library

The following shows how you to generate a presigned request so that you can use the WebSocket library to send requests to the service,.

Add a policy for WebSocket requests to your IAM role

To use the WebSocket protocol to call network analyzer, attach the following policy to the AWS Identity and Access Management (IAM) role that makes this request.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotwireless:StartNetworkAnalyzerStream",
      "Resource": "*"
    }
  ]
}
```

Create a presigned URL

Construct a URL for your WebSocket request that contains the information needed to set up communication between your application and the network analyzer. To verify the identity of the request, WebSocket streaming uses the Amazon Signature Version 4 process for signing requests. For more information about Signature Version 4, see [Signing AWS API Requests](#) in the *Amazon Web Services General Reference*.

To call network analyzer, use the `StartNetworkAnalyzerStream` request URL. The request will be signed using the credentials for the IAM role mentioned previously. The URL has the following format with line breaks added for readability.

```
GET wss://api.iotwireless.<region>.amazonaws.com/start-network-analyzer-stream?X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=Signature Version 4 credential scope
  &X-Amz-Date=date
  &X-Amz-Expires=time in seconds until expiration
  &X-Amz-Security-Token=security-token
  &X-Amz-Signature=Signature Version 4 signature
  &X-Amz-SignedHeaders=host
```


Use the following values for the Signature Version 4 parameters:

- **X-Amz-Algorithm** – The algorithm you're using in the signing process. The only valid value is `AWS4-HMAC-SHA256`.
- **X-Amz-Credential** – A string separated by slashes ("/") that is formed by concatenating your access-key ID and your credential scope components. Credential scope includes the date in YYYYMMDD format, the AWS Region, the service name, and a termination string (`aws4_request`).
- **X-Amz-Date** – The date and time that the signature was created. Generate the date and time by following the instructions in [Handling Dates in Signature Version 4](#) in the *Amazon Web Services General Reference*.
- **X-Amz-Expires** – The length of time in seconds until the credentials expire. The maximum value is 300 seconds (5 minutes).
- **X-Amz-Security-Token** – (optional) A Signature Version 4 token for temporary credentials. If you specify this parameter, include it in the canonical request. For more information, see [Requesting Temporary Security Credentials](#) in the *AWS Identity and Access Management User Guide*.
- **X-Amz-Signature** – The Signature Version 4 signature that you generated for the request.
- **X-Amz-SignedHeaders** – The headers that are signed when creating the signature for the request. The only valid value is `host`.

Construct the request URL and create Signature Version 4 signature

To construct the URL for the request and create the Signature Version 4 signature, use the following steps.

Note

The examples in this section are in pseudocode. For a sample Python code that shows how to create the signature, see [Sample Python code to generate presigned URL](#).

Task 1: Create a canonical request

Create a string that includes information from your request in a standardized format. This ensures that when AWS receives the request, it can calculate the same signature that you calculate in [Task 3: Calculate the signature](#). For more information, see [Create a Canonical Request for Signature Version 4](#) in the *Amazon Web Services General Reference*.

1. Define variables for the request in your application.

```
# HTTP verb
method = "GET"
# Service name
service = "iotwireless"
# AWS Region
region = "AWS Region"
# Service streaming endpoint
endpoint = "wss://api.iotwireless.<region>.amazonaws.com"
# Host
host = "api.iotwireless.<region>.amazonaws.com"
# Date and time of request
amz-date = 'YYYYMMDD'T'HHMMSS'Z'
# Date without time for credential scope
datestamp = YYYYMMDD
```

2. Create a canonical URI (uniform resource identifier). The canonical URI is the part of the URI between the domain and the query string.

```
canonical_uri = "/start-network-analyzer-stream"
```

3. Create the canonical headers and signed headers. Note the trailing `\n` in the canonical headers.

- Append the lowercase header name followed by a colon.
- Append a comma-separated list of values for that header. Don't sort the values in headers that have multiple values.
- Append a new line (`\n`).

```
canonical_headers = "host:" + host + "\n"
signed_headers = "host"
```

4. Match the algorithm to the hashing algorithm. You must use SHA-256.

```
algorithm = "AWS4-HMAC-SHA256"
```

5. Create the credential scope, which scopes the derived key to the date, Region, and service to which the request is made.

```
credential_scope = datestamp + "/" + region + "/" + service + "/" + "aws4_request"
```

6. Create the canonical query string. Query string values must be URI-encoded and sorted by name.
 - Sort the parameter names by character code point in ascending order. Parameters with duplicate names should be sorted by value. For example, a parameter name that begins with the uppercase letter F precedes a parameter name that begins with a lowercase letter b.
 - Do not URI-encode any of the unreserved characters that [RFC 3986](#) defines: A–Z, a–z, 0–9, hyphen (-), underscore (_), period (.), and tilde (~).
 - Percent-encode all other characters with %XY, where X and Y are hexadecimal characters (0-9 and uppercase A-F). For example, the space character must be encoded as %20 (not using '+', as some encoding schemes do) and extended UTF-8 characters must be in the form %XY%ZA%BC.
 - Double-encode any equals (=) characters in parameter values.

```
canonical_querystring = "X-Amz-Algorithm=" + algorithm
canonical_querystring += "&X-Amz-Credential=" + URI-encode(access key + "/" +
  credential_scope)
canonical_querystring += "&X-Amz-Date=" + amz_date
canonical_querystring += "&X-Amz-Expires=300"
canonical_querystring += "&X-Amz-Security-Token=" + token
canonical_querystring += "&X-Amz-SignedHeaders=" + signed_headers
canonical_querystring += "&language-code=en-US&media-encoding=pcm&sample-
  rate=16000"
```

7. Create a hash of the payload. For a GET request, the payload is an empty string.

```
payload_hash = HashSHA256(("").Encode("utf-8")).HexDigest()
```

8. Combine all of the elements to create the canonical request.

```
canonical_request = method + '\n'
  + canonical_uri + '\n'
  + canonical_querystring + '\n'
  + canonical_headers + '\n'
  + signed_headers + '\n'
  + payload_hash
```

Task 2: Create the string to sign

The string to sign contains meta information about your request. You use the string to sign in the next step when you calculate the request signature. For more information, see [Create a String to Sign for Signature Version 4](#) in the *Amazon Web Services General Reference*.

```
string_to_sign=algorithm + "\n"
+ amz_date + "\n"
+ credential_scope + "\n"
+ HashSHA256(canonical_request.Encode("utf-8")).HexDigest()
```

Task 3: Calculate the signature

You derive a signing key from your AWS secret access key. For a greater degree of protection, the derived key is specific to the date, service, and AWS Region. You use the derived key to sign the request. For more information, see [Calculate the Signature for AWS Signature Version 4](#) in the *Amazon Web Services General Reference*.

The code assumes that you have implemented the `GetSignatureKey` function to derive a signing key. For more information and example functions, see [Examples of How to Derive a Signing Key for Signature Version 4](#) in the *Amazon Web Services General Reference*.

The function `HMAC(key, data)` represents an HMAC-SHA256 function that returns the results in binary format.

```
#Create the signing key
signing_key = GetSignatureKey(secret_key, timestamp, region, service)

# Sign the string_to_sign using the signing key
signature = HMAC.new(signing_key, (string_to_sign).Encode("utf-8"), Sha256()).HexDigest
```

Task 4: Add signing information to request and create request URL

After you calculate the signature, add it to the query string. For more information, see [Add the Signature to the Request](#) in the *Amazon Web Services General Reference*.

```
#Add the authentication information to the query string
canonical_querystring += "&X-Amz-Signature=" + signature

# Sign the string_to_sign using the signing key
```

```
request_url = endpoint + canonical_uri + "?" + canonical_querystring
```

Next steps

You can now use the request URL with your WebSocket library to make the request to the service and observe the messages. For more information, see [WebSocket messages and status codes](#). For a sample Python code that shows how to generate a presigned request, see [Sample Python code to generate presigned URL](#).

Sample Python code to generate presigned URL

The following code shows an example for generating the pre-signed URL using Python as the programming language.

Pre-requisites

To use the Python programming language to generate requests, you must have:

- Python installed on your computer. You can either run the following command or download the [Python installer](#) and then run it.

```
sudo apt install python3
```

- The Python requests library. You can either run the following command or download the [Requests library](#), which is used in the example script to make web requests.

```
pip install requests
```

- An access key that consists of the access key ID and secret access key in environment variables named `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Alternatively, you can keep these values in a credentials file and read them from that file.

Note

As a best practice, we recommend that you do not embed credentials in code. For more information, see [Best Practices for AWS accounts](#) in the AWS account Management Reference Guide.

```
$ export AWS_ACCESS_KEY_ID=My_Access_Key
```

```
$ export AWS_SECRET_ACCESS_KEY=My_Secret_Key

# Session token is required only if you use temporary access key starting with "ASIA"
$ export AWS_SESSION_TOKEN=My_Session_token
```

Sample Python code

This Python code generates the pre-signed URL that the WebSocket library can use to send requests to the service. The function creates a canonical request, then creates the string to sign which is used to calculate the signature, and then adds the signature to the HTTP request to create the pre-signed URL. You can then use the WebSocket library to request the pre-signed URL.

To run the script, `generate_presigned_url.py`, run the following command if you're running it from the same path where the script is located.

```
python generate_presigned_url.py
```

The following shows the contents of the `generate_presigned_url.py` script.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# Version 4 signing example

"""
    Sample Python code to generate the pre-signed URL. You can
    change the parameters in this code to your own values, such
    as the variables that are required for the request URL, the
    network analyzer configuration name, and Region.
"""

# -----
# Step 1. Import the required libraries and define the functions
# sign and getSignatureKey that will be used to derive a signing key.
# -----
import sys, os, base64, datetime, hashlib, hmac, urllib.parse
import requests      # pip install requests

def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()
```

```

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(("AWS4" + key).encode("utf-8"), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, "aws4_request")
    return kSigning

# -----
# Step 2. Define the variables required for the request URL. Replace
# values for the variables, such as region, with your own values.
# -----
method = "GET"
service = "iotwireless"
region = "us-east-1"

# Host and endpoint information.
host = "api.iotwireless." + region + ".amazonaws.com"
endpoint = "wss://" + host

# Create a date for headers and the credential string.
t = datetime.datetime.utcnow()
amz_date = t.strftime("%Y%m%dT%H%M%SZ")

# For date stamp, the date without time is used in credential scope.
datestamp = t.strftime("%Y%m%d")

# -----
# Step 3. Create the canonical URI and canonical headers for the request.
# -----
canonical_uri = "/start-network-analyzer-stream"
configuration_name = "My_Network_Analyzer_Config"

canonical_headers = "host:" + host + "\n"
signed_headers = "host"
algorithm = "AWS4-HMAC-SHA256"
credential_scope = datestamp + "/" + region + "/" + service + "/" + "aws4_request"

# -----
# Step 4. Read the credentials that are required for the request
# from environment variables or configuration file.
# -----

# IMPORTANT: Best practice is NOT to embed credentials in code.

```

```

access_key = os.environ.get("AWS_ACCESS_KEY_ID")
secret_key = os.environ.get("AWS_SECRET_ACCESS_KEY")
token = os.environ.get("AWS_SESSION_TOKEN")

if access_key is None or secret_key is None:
    print("No access key is available.")
    sys.exit()

if access_key.startswith("ASIA") and token is None:
    print("Detected temporary credentials. You must specify a token.")
    sys.exit()

# -----
# Step 5. Create the canonical query string. Query string values must be
# URI-encoded and sorted by name. Query headers must in alphabetical order.
# -----
canonical_querystring = "X-Amz-Algorithm=" + algorithm

canonical_querystring += "&X-Amz-Credential=" + \
    urllib.parse.quote(access_key + "/" + credential_scope, safe="-_~")

canonical_querystring += "&X-Amz-Date=" + amz_date
canonical_querystring += "&X-Amz-Expires=300"

if access_key.startswith("ASIA"):
    # percent encode the token and double encode "="
    canonical_querystring += "&X-Amz-Security-Token=" + \
        urllib.parse.quote(token, safe="-_~").replace("=", "%253D")

canonical_querystring += "&X-Amz-SignedHeaders=" + signed_headers
canonical_querystring += "&configuration-name=" + configuration_name

# -----
# Step 6. Create a hash of the payload.
# -----
payload_hash = hashlib.sha256(("").encode("utf-8")).hexdigest()

# -----
# Step 7. Combine the elements, which includes the query string, the
# headers, and the payload hash, to form the canonical request.
# -----
canonical_request = method + "\n" + canonical_uri + "\n" + canonical_querystring \
+ "\n" + canonical_headers + "\n" + signed_headers + "\n" + payload_hash

```



```

# -----
# Step 8. Create the metadata string to store the information required to
# calculate the signature in the following step.
# -----
string_to_sign = algorithm + "\n" + amz_date + "\n" + \
credential_scope + "\n" +
    hashlib.sha256(canonical_request.encode("utf-8")).hexdigest()

# -----
# Step 9. Calculate the signature by using a signing key that's obtained
# from your secret key.
# -----
# Create the signing key from your secret key.
signing_key = getSignatureKey(secret_key, datestamp, region, service)

# Sign the string_to_sign using the signing key.
signature = hmac.new(signing_key, (string_to_sign).encode("utf-8"),
    hashlib.sha256).hexdigest()

# -----
# Step 10. Create the request URL using the calculated signature and by
# combining it with the canonical URI and the query string.
# -----
canonical_querystring += "&X-Amz-Signature=" + signature

request_url = endpoint + canonical_uri + "?" + canonical_querystring

print("\n-----PRESIGNED URL-----")
print(request_url)

```

Next steps

You can now use the request URL with your WebSocket library to make the request to the service and observe the messages.

To install a WebSocket library to use with Python, run the following command. For information about how you can use a WebSocket client with Python, see [WebSocket client for Python with low level API options](#).

```
pip install websocket-client
```

After you install the client and make the request, you'll see messages and status codes that indicate the status of your request. For more information, see [WebSocket messages and status codes](#).

WebSocket messages and status codes

After you've created a presigned request, you can use the request URL with your WebSocket library, , or a library that's suited to your programming language, to make requests to the service. For more information about how you can generate this presigned request, see [Generate a presigned request with the WebSocket library](#).

WebSocket messages

The WebSocket protocol can be used to establish a bi-directional connection. Messages can be transmitted from client to server and from server to client. However, network analyzer supports only messages that are sent from server to client. Any message received from the client is unexpected and the server will automatically close the WebSocket connection if a message is received from client.

When the request is received and a trace messaging session has started, the server responds with a JSON structure, which is the payload. For more information about the payload and how you can activate trace messaging from the AWS Management Console, see [View and monitor network analyzer trace message logs in real time](#).

WebSocket status codes

The following shows the WebSocket status codes for the communication from the server to client. The WebSocket status codes follow the [RFC Standard of Normal closure of connections](#).

The following shows the supported status codes:

- **1000**

This status code indicates a normal closure, which means that the WebSocket connection has been established and the request has been fulfilled. This status can be observed when a session is idle, causing the connection to time out.

- **1002**

This status code indicates that the endpoint is terminating the connection because of a protocol error.

- **1003**

This status code indicates an error status where the endpoint terminated the connection because it received data in a format that it can't accept. The endpoint supports only text data and might

display this status code if it receives a binary message or a message from the client that's using an unsupported format.

- **1008**

This status code indicates an error status where the endpoint terminated the connection because it received a message that violates its policy. This status is generic and is displayed when the other status codes, such as 1003 or 1009, aren't applicable. You'll also see this status displayed if there's a need to hide the policy, or when there's an authorization failure, such as an expired signature.

- **1011**

This status code indicates an error status where the server is terminating the connection because it encountered an unexpected condition or internal error that prevented it from fulfilling the request.

Next steps

Now that you've learned how to generate a presigned request and how you can observe messages from the server by using the WebSocket connection, you can activate trace messaging and start receiving message logs for your wireless gateway and wireless device resources. For more information, see [View and monitor network analyzer trace message logs in real time](#).

View and monitor network analyzer trace message logs in real time

If you've added resources to your network analyzer configuration, you can activate trace messaging to start receiving trace messages for your resources. You can use either the AWS Management Console, the AWS IoT Wireless API, or the AWS CLI.

Prerequisites

Before you can activate trace messaging by using network analyzer, you must have:

- Added the resources that you want to monitor to your default network analyzer configuration. For more information, see [Add resources and update the network analyzer configuration](#).
- Generated a presigned request by using the `StartNetworkAnalyzerStream` request URL. The request will be signed using the credentials for the AWS Identity and Access Management role that makes this request. For more information, see [Create a presigned URL](#).

Activate trace messaging by using the console

To activate trace messaging

1. Open the [Network Analyzer hub of the AWS IoT console](#) and choose your network analyzer configuration, **NetworkAnalyzerConfig_Default**.
2. In the details page of your network analyzer configuration, choose **Activate trace messaging** and then choose **Activate**.

You'll start receiving trace messages where the newest trace message appears first in the console.

Note

After the messaging session starts, receiving trace messages can incur additional costs until you deactivate the session or leave the trace session. For more information about pricing, see [AWS IoT Core pricing](#).

View and monitor trace messages

After you activate trace messaging, the WebSocket connection is established and trace messages start appearing in real time, newest first. You can customize the preferences to specify the number of trace messages to be displayed in each page and to display only the relevant fields for each message. For example, you can customize the trace message log to show only logs for wireless gateway resources that have **Log level** set to ERROR, so that you can quickly identify and debug errors with your gateways. The trace messages contain the following information.

- **Message Number:** A unique number that shows the last message received first.
- **Resource ID:** The wireless gateway or wireless device ID of the resource.
- **Timestamp:** The time when the message was received.
- **Message ID:** An identifier that AWS IoT Core for LoRaWAN assigns to each received message.
- **FPort:** The frequency port for communicating with the device by using the WebSocket connection.
- **DevEui:** The extended unique identifier (EUI) for your wireless device.
- **Resource:** Whether the monitored resource is a wireless device or a wireless gateway.

- **Event:** The event for a log message for a wireless device, which can be **Join**, **Rejoin**, **Uplink_Data**, **Downlink_Data**, or **Registration**.
- **Log level:** Information about INFO or ERROR log streams for your device.

Network analyzer JSON log message

You can also choose one trace message at a time to view the JSON payload for that message. Depending on the message that you select in the trace message logs, you'll see information in the JSON payload that indicates contains 2 parts: **CustomerLog** and **LoRaFrame**.

CustomerLog

The **CustomerLog** part of the JSON displays the type and identifier of the resource that received the message, the log level, and the message content. The following example shows a **CustomerLog** log message. You can use the message field in the JSON to get more information about the error and how it can be resolved.

LoRaFrame

The **LoRaFrame** part of the JSON has a **Message ID** and contains information about the physical payload for the device and the wireless metadata. The wireless metadata also contains information about the gateway metadata, and whether the trace message was received from the public network or from a private LoRaWAN gateway.

The following shows examples of the trace message.

Trace message log for private gateways

The following example shows a sample trace message received using network analyzer if your devices connect to AWS IoT Core for LoRaWAN using your own private LoRaWAN gateways. The metadata consists of the ID of the gateway and its EUI, and the SNR and RSSI values. These values can help you determine the strength of your gateway channel and whether to switch to a stronger channel. For more information about the public network, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

```
{
  "resource_id": "d05bef08-cab2-41bf-b69e-ce306b9a5f81",
  "frame_type": "LoRa",
  "timestamp": "2024-02-15T16:49:35.966023978Z",
  "lora_frame":
  {
```

```

    "dev_eui": "4d767373e0ec05c4",
    "message_id": "8e6dcc61-80b6-45c1-89d3-c712cf5603fb",
    "phy_payload": "XXX",
    "wireless_metadata":
    {
        "dev_eui": "4d767373e0ec05c4",
        "m_type": "CONFIRMED_DATA_UP",
        "f_port": 22,
        "data_rate": 3,
        "frequency": 904300000,
        "timestamp": "2024-02-15T16:49:35.966023978Z",
        "lorawan_gateways":
        {
            "wireless_gateway_id": "d0bfb1d8-b0b6-48f8-b9bb-d0aadf1ab2cf",
            "gateway_eui": "4b634d3cc5879def",
            "snr": 5.099999904632568,
            "rssi": -35
        }
    },
    "dev_addr": "012c58d1"
}
}

```

Trace message log for public network

The following example shows a sample trace message received using network analyzer if your devices use the public network to connect to AWS IoT Core for LoRaWAN. The public network is provided and operated as a service directly by Everynet. The following example shows the public LoRaWAN network metadata in the uplink message. The metadata consists of the ID of the gateway and the network provider (Everynet), whether downlink is allowed, and the SNR and RSSI values. These values can help you determine the strength of your public network. For more information about the public network, see [Managing LoRaWAN traffic from public LoRaWAN devices networks \(Everynet\)](#).

Note

The uplink message will mention `public_lorawan_gateways` to indicate that it's received from the public network and not a private LoRaWAN gateway.

```
{
```

```

"resource_id": "d05bef08-cab2-41bf-b69e-ce306b9a5f81",
"frame_type": "LoRa",
"timestamp": "2024-02-15T16:49:35.966023978Z",
"lora_frame":
{
  "dev_eui": "4d767373e0ec05c4",
  "message_id": "8e6dcc61-80b6-45c1-89d3-c712cf5603fb",
  "phy_payload": "XXX",
  "wireless_metadata":
  {
    "dev_eui": "4d767373e0ec05c4",
    "m_type": "CONFIRMED_DATA_UP",
    "f_port": 22,
    "data_rate": 3,
    "frequency": 904300000,
    "timestamp": "2024-02-15T16:49:35.966023978Z",
    "public_lorawan_gateways":
    {
      "provider_net_id": "0x0000b",
      "id": "3abe094",
      "snr": 5.0999999904632568,
      "rssi": -35,
      "rfregion": US915,
      "dl_allowed": true
    }
  },
  "dev_addr": "012c58d1"
}
}

```

Trace message log without gateway metadata

If you want to exclude the gateway metadata information from your trace message, disable the **AddGwMetadata** parameter when you create the service profile. For information about disabling this parameter, see [Add service profiles](#).

The following example shows a trace message with the `lora_frame` and `customer_log` information and doesn't contain any gateway information.

```

{
  "resource_id": "d05bef08-cab2-41bf-b69e-ce306b9a5f81",
  "frame_type": "LoRa",
  "timestamp": "2024-02-15T16:49:35.966023978Z",

```

```
"lora_frame":
  {
    "dev_eui": "4d767373e0ec05c4",
    "message_id": "8e6dcc61-80b6-45c1-89d3-c712cf5603fb",
    "phy_payload": "XXX",
    "wireless_metadata":
      {
        "dev_eui": "4d767373e0ec05c4",
        "m_type": "CONFIRMED_DATA_UP",
        "f_port": 22,
        "data_rate": 3,
        "frequency": 904300000,
        "timestamp": "2024-02-15T16:49:35.966023978Z"
      },
    "dev_addr": "012c58d1"
  },
  "customer_log"
  {
    "resource": "WirelessDevice",
    "wireless_device_id": "8 ab0c23d3-b001-45ef-6a01-2bc3de4f5333",
    "wireless_device_type": "LoRaWAN",
    "log_level": "INFO",
    "event": "Uplink_Data",
    "message": "Uplink message received",
    "messageId": "59e7d840-d756-4978-8c64-7f60cfd3fd3b"
  }
}
```

Review and next steps

In this section, you've viewed trace messages and learned how you can use the information to debug errors. After you've viewed all messages, you can:

- **Deactivate trace messaging**

To avoid incurring any additional costs, you can deactivate the trace messaging session. Deactivating the session disconnects your WebSocket connection so you won't receive any additional trace messages. You can still continue viewing the existing messages in the console.

- **Edit frame info for your configuration**

You can edit the network analyzer configuration and choose whether to deactivate frame info and choose the log levels for your messages. Before you update your configuration, consider

deactivating your trace messaging session. To make these edits, open the [Network Analyzer details page in the AWS IoT console](#) and choose **Edit**. You can then update your configuration with the new configuration settings and activate trace messaging to see the updated messages.

- **Add resources to your configuration**

You can also add more resources to your network analyzer configuration and monitor them in real time. You can add up to a combined total of 250 wireless gateway and wireless device resources. To add resources, on the [Network Analyzer details page of the AWS IoT console](#), choose the **Resources** tab and choose **Add resources**. You can then update your configuration with the new resources and activate trace messaging to see the updated messages for the additional resources.

For more information about updating your network analyzer configuration by editing the configuration settings and adding resources, see [Add resources and update the network analyzer configuration](#).

Debug and troubleshoot your multicast groups and FUOTA tasks using network analyzer

The wireless resources that you can monitor include LoRaWAN devices, LoRaWAN gateways, and multicast groups. You can also use network analyzer to debug and troubleshoot any issues with your FUOTA task. You can also monitor and track messages related to setup, data transmission, and status query when the FUOTA task is in progress.

To monitor your FUOTA task, if the task contains multicast groups, you must add both the multicast group and the devices in the group to your network analyzer configuration. You must also activate frame info and multicast frame info to track the unicast and multicast uplink and downlink messages that are exchanged with the multicast group and the devices while the FUOTA task is in progress.

To monitor multicast groups, you can add them to your network analyzer configuration and use multicast frame info to troubleshoot multicast downlink messages that are sent to these groups. For troubleshooting devices that are attempting to join a group, where unicast communication is used, you must also include these devices in the network analyzer configuration. To monitor only the unicast communication with the devices in the group, activate the frame info for your wireless devices. This approach ensures comprehensive monitoring and diagnostics for both multicast groups and devices that are joining the group.

The following sections describe how to debug and troubleshoot your multicast groups and FUOTA tasks using network analyzer.

Topics

- [Debug FUOTA tasks that only contains devices](#)
- [Debug FUOTA tasks with multicast groups](#)
- [Debug devices that are attempting to join a multicast group](#)
- [Debug a multicast group session](#)

Debug FUOTA tasks that only contains devices

You can use network analyzer to debug a FUOTA task that only has LoRaWAN devices added to the task. For information about adding devices to a FUOTA task, see [Add devices and multicast groups to a FUOTA task and schedule a FUOTA session](#). To debug the FUOTA task, perform the following steps:

1. Create a network analyzer configuration by activating frame info for the wireless devices so that you can monitor the FUOTA uplink and downlink messages that are exchanged with the devices while the task is in progress.
2. Add the devices in your FUOTA task to the network analyzer configuration by using their wireless device identifiers.
3. Activate trace messaging to start receiving trace messages for the devices in your network analyzer configuration.

In the `applicationCommandType` column of the trace message information, you'll start receiving unicast downlink messages related to data transmission and fragmentation setup.

Note

If you don't see the `applicationCommandType` column in the trace message table, you can adjust the settings to show this column in the table.

You can also see the `applicationCommandType` and other detailed messages in the JSON log message under **WirelessMetadata > ApplicationInfo**.

Debug FUOTA tasks with multicast groups

You can use network analyzer to debug a FUOTA task that has multicast groups and LoRaWAN devices added to the group. For information about adding devices to a FUOTA task, see [Add devices and multicast groups to a FUOTA task and schedule a FUOTA session](#). To debug the FUOTA task, perform the following steps:

1. Create a network analyzer configuration by activating the frame info and multicast frame info settings for the wireless devices and multicast groups.
2. Add the multicast group in your FUOTA task to the network analyzer configuration by using their multicast group identifier. By enabling multicast frame info, you can debug the firmware data message and FUOTA status query messages that are sent to the group while the FUOTA task is in progress.
3. Add the devices in your multicast group to the network analyzer configuration by using their wireless device identifiers. By activating frame info, you can monitor uplink and downlink messages that are exchanged directly with the devices while the FUOTA task is in progress.
4. Activate trace messaging to start receiving trace messages for the devices and multicast groups in your network analyzer configuration.

You can then view the trace messages and debug them using the `applicationCommandType` column of the trace message table and using the details in the JSON log message as described in [Debug FUOTA tasks that only contains devices](#).

Debug devices that are attempting to join a multicast group

You can use network analyzer to debug devices that are attempting to join a multicast group. For information about adding devices to a multicast group, see [Create multicast groups and add devices to the group](#). To debug the multicast group, perform the following steps:

1. Create a network analyzer configuration by activating frame info for the wireless devices.
2. Add the devices you want to monitor to the network analyzer configuration by using their wireless device identifiers.
3. Activate trace messaging to start receiving trace messages for the devices in your network analyzer configuration.
4. Start associating the devices to the multicast group after trace messaging has been activated for the devices in the group.

Debug a multicast group session

You can use network analyzer to debug a multicast group session. For more information, see [Schedule a downlink message to send to devices in your multicast group](#). To debug a multicast group session, perform the following steps:

1. Create a network analyzer configuration by activating multicast frame info for the multicast group.
2. Add the multicast group that you want to monitor to the network analyzer configuration by using their multicast group identifier.
3. Before the multicast session starts, activate trace messaging to start receiving trace messages for the multicast group session.
4. Start the multicast group session and monitor the status by viewing the messages that are displayed in the trace message table and the JSON log message.

In the trace message table, the `MuLticastAddr` will be displayed in the `DevAddr` column. In the JSON log message, you can view detailed information such as the `MuLticastGroupId` under **WirelessMetadata > ApplicationInfo**.

Use AWS IoT Core for LoRaWAN metrics

Use AWS IoT Core for LoRaWAN metrics to monitor the health of your LoRaWAN resources in a dashboard view. It provides information about the connectivity of your devices with the cloud, how they are functioning, and whether they are operating within specifications. These metrics can be aggregated to provide historical and up-to-the-minute view of data and trends for your resources.

What metrics can I view?

When you activate summary metrics, you can view the following information for all resources, or for your individual devices and gateways.

If you see that no data is available for one or more widgets, check whether you have completed the activation steps for viewing summary metrics. You can use the **Activation steps** widget in the dashboard to check whether you have correctly configured your LoRaWAN devices, gateways, and destinations, and activated summary metrics. For information about onboarding your LoRaWAN resources, see [Connecting gateways and devices to AWS IoT Core for LoRaWAN](#).

List of summary metrics

Metric name	AWS account	Individual devices	Individual gateways
Active devices/gateways	Yes	–	–
Uplink message count	Yes	Yes	Yes
Downlink message count	Yes	Yes	Yes
Join metrics	Yes	Yes	Yes
Message lost rate	Yes	Yes	Yes
Signal to noise ratio (SNR) average	–	Yes	Yes
Received signal strength indicator (RSSI) average	–	Yes	Yes
Gateway availability	–	–	Yes

How to view summary metrics?

To view the metrics for your LoRaWAN resources, you can activate the summary metrics dashboard either from the console or using the AWS IoT Wireless API operations. You activate summary metrics for all LoRaWAN resources in your AWS account, which includes all LoRaWAN devices and gateways. The data can be aggregated to provide you with hourly, daily, or weekly information for your resources.

The LoRaWAN metrics dashboard uses the compute API to display information for your LoRaWAN resources. When you activate summary metrics, charges might be incurred for using the LoRaWAN dashboard. To avoid incurring additional charges, you can deactivate summary metrics. For information about pricing, see <https://aws.amazon.com/iot-core/pricing> AWS IoT Core pricing.

Note

The summary metrics have an expiration time. Hourly metrics expire after six months, and daily and weekly metrics expire after 12 months.

Topics

- [Activate summary metrics \(console\)](#)
- [Activate summary metrics \(CLI\)](#)

Activate summary metrics (console)

1. Go to the [Monitor](#) dashboard of the AWS IoT console and choose **LoRaWAN metrics**.
2. Choose **Activate summary metrics** and specify the period that you want to use for data to be aggregated, which can be **Daily**, **Hourly**, or **Weekly**.

You'll see data flowing through the dashboard and displayed for your resources in the **LoRaWAN overview** widget and in the widget corresponding to each metric. If you don't see any data displayed, check whether you have correctly configured your LoRaWAN resources and performed the join procedure for your LoRaWAN devices so that they can start sending uplink data.

Activate summary metrics (CLI)

To activate summary metrics and see data flowing through the dashboard, use the [UpdateMetricConfiguration](#) API operation or the [update-metric-configuration](#) CLI command. The following code shows a sample request body.

```
{
  "SummaryMetricQueries": [
    {
      "AggregationPeriod": "OneWeek",
      "Dimensions": [
        {
          "name": "DeviceId",
          "value": "30758fe6-56f1-4f32-8c4c-f17f000e01d3"
        }
      ],
      "EndTimestamp": 1699574400,
      "MetricName": "DeviceUplinkLostRate",
      "QueryId": "DeviceUplinkLostRate",
      "StartTimestamp": 1698969600
    }
  ]
}
```

After you've activated summary metrics, you can use the [GetMetrics](#) API operation or the [get-metrics](#) CLI command.

```
{
  "SummaryMetric": {
    "Status": Disabled
  }
}
```

You can then use the [GetMetricConfiguration](#) API operation or the [get-metric-configuration](#) CLI command to view the metric configuration status. The following shows a sample response.

```
{
  "SummaryMetric": {
    "Status": Disabled
  }
}
```

LoRaWAN metrics

After you activate summary metrics, you can view metrics pertaining to historical data for all LoRaWAN devices and gateways in your AWS account. You can also find metrics for each of your LoRaWAN devices and gateways. The following sections describe the overall metrics for all your resources in general, and for each resource in your AWS account.

If you don't see any data in the widgets for the metrics, make sure that you have performed the activation steps correctly for viewing the metrics. You can also configure logging and use the network analyzer to debug any issues.

Note

In the console, you can drag the widgets for the metrics to different locations on the dashboard. Relocating the widgets might temporarily change the layout of the dashboard.

The following shows the metrics that you can see in the dashboard and in the details pages of your gateways and devices on the console.

LoRaWAN summary metrics

To view the summary metrics for all resources, go to the [Monitor](#) dashboard of the AWS IoT console and then choose the **LoRaWAN metrics** tab.

- An overview of key LoRaWAN summary metrics.
- The activation steps and their progress.
- The number of active devices and gateways.
- The count of uplink and downlink messages.
- The message loss rate, which shows the ratio of the total number of packets that are lost.
- The join metrics, which shows the number of join requests and join accepts.

LoRaWAN device metrics

You can view metrics showing historical data for each LoRaWAN device that you onboarded to AWS IoT Core for LoRaWAN. You'll find similar metrics as in the **Monitor** dashboard with some additional metrics such as the average received signal strength indicator (RSSI) and signal to noise ratio (SNR).

You can also find a summary of the metrics for all your LoRaWAN devices in the [Devices](#) hub of the AWS IoT console. This includes:

- The number of active devices within the specified time duration, which can be the last hour, day, or week
- The total number of devices that have been provisioned up to the specified timestamp range, which can be the last hour, day, or week.
- The uplink packet loss rate, which corresponds to the ratio of total number of uplink packets that are lost during transmission.
- The individual device metrics, go to the [Devices](#) hub of the AWS IoT console and then choose the device for which you want to see the metrics. For information about activating these metrics, see [How to view summary metrics?](#).

You can view the following metrics for each LoRaWAN device.

Note

If you don't see any data in the widgets for the metrics, make sure that you have onboarded your device correctly for viewing the metrics. You can also configure logging and use the network analyzer to debug any issues.

- A summary of key LoRaWAN device metrics
- The count of uplink and downlink messages
- The uplink packet loss rate, which shows the ratio of the total number of packets that are lost.
- The join metrics, which shows the number of join requests and join accepts.
- The average received signal strength indicator (RSSI) and signal to noise ratio (SNR).

LoRaWAN gateways metrics

You can view metrics showing historical data for each LoRaWAN gateway that you onboarded to AWS IoT Core for LoRaWAN. You'll find similar metrics as in the **Monitor** dashboard with some additional metrics such as the average received signal strength indicator (RSSI), signal to noise ratio (SNR), and gateway availability. You can also find a summary of the metrics for all your LoRaWAN gateways in the **Gateways** hub of the AWS IoT console.

You can also find a summary of the metrics for all your LoRaWAN gateways in the [Gateways](#) hub of the AWS IoT console. This includes:

- The number of uplink and downlink messages that were exchanged between the devices and the cloud using the gateways within the specified time duration, which can be the last hour, day, or week.
- The total number of gateways that have been provisioned up to the specified timestamp range, which can be the last hour, day, or week.

You can view the following metrics for each LoRaWAN gateway.

Note

If you don't see any data in the widgets for the metrics, make sure that you have onboarded your gateway correctly for viewing the metrics. You can also configure logging and use the network analyzer to debug any issues.

- A summary of key LoRaWAN gateway metrics
- The count of uplink and downlink messages
- The uplink packet loss rate, which shows the ratio of the total number of packets that are lost.
- The join metrics, which shows the number of join requests and join accepts.
- The gateway availability information.
- The average received signal strength indicator (RSSI) and signal to noise ratio (SNR).

Active devices and gateways

After you've onboarded your gateways and devices to AWS IoT Core for LoRaWAN, your devices will start exchanging messages with the cloud. This metric displays the number of active LoRaWAN devices and gateways within a specified time duration in your AWS account. Active devices and gateways are those devices that have transmitted or received any uplink or downlink data, or gateways that facilitate such data transmission.

In the Monitor dashboard, you can use the LoRaWAN overview section to see the total number of gateways and devices that have been provisioned, and the number of gateways and devices that are active. Using this information, you can identify what percentage of devices and gateways that have been provisioned are active.

Uplink message count

This metric displays the number of uplink messages that are sent within a specified time duration for all active gateways and devices in your AWS account. Uplink messages are messages that are sent from your device to AWS IoT Core for LoRaWAN.

The count of uplink messages includes the total number of uplink messages and the number of uplink messages that are sent from your device to AWS IoT Core for LoRaWAN using a public network. You can use this information to identify the volume of uplink messages and the quantity of uplink messages that are arriving at the cloud over the public network during that period.

Downlink message count

This metric displays the number of downlink messages that are sent within a specified time duration for all active gateways and devices in your AWS account. Downlink messages are messages that are sent from AWS IoT Core for LoRaWAN to your devices.

The count of downlink messages includes the total number of downlink messages and the number of downlink messages that are sent from AWS IoT Core for LoRaWAN to your devices using a public network. You can use this information to identify the volume of downlink messages and the quantity of downlink messages that are arriving at your devices over the public network during that period.

Message loss rate

After you've added your device and connected to AWS IoT Core for LoRaWAN, your device can initiate an uplink message to start exchanging messages with the cloud. You can use this metric to then track the rate of uplink messages that are lost. Uplink messages are lost due to the loss of signal during radio transmission from the device to gateway.

The rate of uplink message loss is indicated by non-sequential frame counters (FCnt) over the specified time duration. This information can be used to assess the stability of the connection. An increase in the message loss rate can indicate that the connection is unstable. To improve the stability of the connection, you can use adaptive data rate (ADR) by setting the ADR bit in the frame header of your devices. If the connection still doesn't improve, you can move the device closer to the gateway or add more gateways around your devices.

Join metrics

After you've added your device and gateway, you perform a join procedure so that your device can send uplink data and communicate with AWS IoT Core for LoRaWAN. You can use this metric to obtain information about join metrics for all active devices in your AWS account.

This metric displays the rate of total number of device join requests and join accepts under a gateway within a certain time duration. This information can be used to determine the total number of join requests and the proportion of requests that have been approved.

If all join requests haven't been accepted, you can use network analyzer or configure CloudWatch Logs to check whether AWS IoT Core for LoRaWAN receives the join requests and if the requests get accepted. If your LoRaWAN devices are setup correctly, the number of join requests and join

accepts must be equal. If all requests are not accepted, check whether you specified the correct DevEUI and root key or session keys when provisioning the device.

Average received signal strength indicator (RSSI)

You can use this metric to monitor the average RSSI (Received signal strength indicator) within the specified time duration.

This metric can be used with the average SNR (signal to noise ratio) to measure signal strength and provide information about connectivity status. RSSI is a measurement that indicates if the signal is strong enough for a good wireless connection. The RSSI average is an average of all RSSI values for the device over a specified time duration, which can help provide information about the device's connection for that duration.

The RSSI value is negative and must be closer to zero for a strong wireless connection. If the signal is weak, you can use adaptive data rate (ADR) by setting the ADR bit in the frame header of your devices to make it stronger. If the connection still doesn't improve, you can move the device closer to the gateway or add more gateways around your device.

Average signal to noise ratio (SNR)

You can use this metric to monitor the average SNR (Signal to noise ratio) within the specified time duration.

This metric can be used with RSSI to measure signal strength and provide information about connectivity status. SNR is a measurement that indicates if the received signal is strong enough compared to the noise level for a good wireless connection. The SNR average is an average of all SNR values for the device over a specified time duration, which can help provide information about the device's connection for that duration.

The SNR value is positive and must be greater than zero to indicate that the signal power is stronger than the noise power. If the signal is weak, you can use adaptive data rate (ADR) by setting the ADR bit in the frame header of your devices to make it stronger. If the connection still doesn't improve, you can move the device closer to the gateway or add more gateways around your device.

Gateway availability

You can use this metric to obtain information about the availability of this gateway within a specified time duration.

This metric displays the websocket connection time of this gateway for a specified time duration. This connection time includes the gateway up time and the gateway down time. You can use this information to identify when a gateway was available. It can also be used with the message loss rate to identify when a packet got lost in transmission.

AWS IoT Core for LoRaWAN and interface VPC endpoints (AWS PrivateLink)

You can connect directly to AWS IoT Core for LoRaWAN through [Interface VPC endpoints \(AWS PrivateLink\)](#) in your Virtual Private Cloud (VPC) instead of connecting over the public internet. When you use a VPC interface endpoint, communication between your VPC and AWS IoT Core for LoRaWAN is conducted entirely and securely within the AWS network.

AWS IoT Core for LoRaWAN supports Amazon Virtual Private Cloud interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more [Elastic Network Interfaces](#) with private IP addresses in your VPC subnets. For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

For more information about VPC and endpoints, see [What is Amazon VPC](#).

For more information about AWS PrivateLink, see [AWS PrivateLink and VPC endpoints](#).

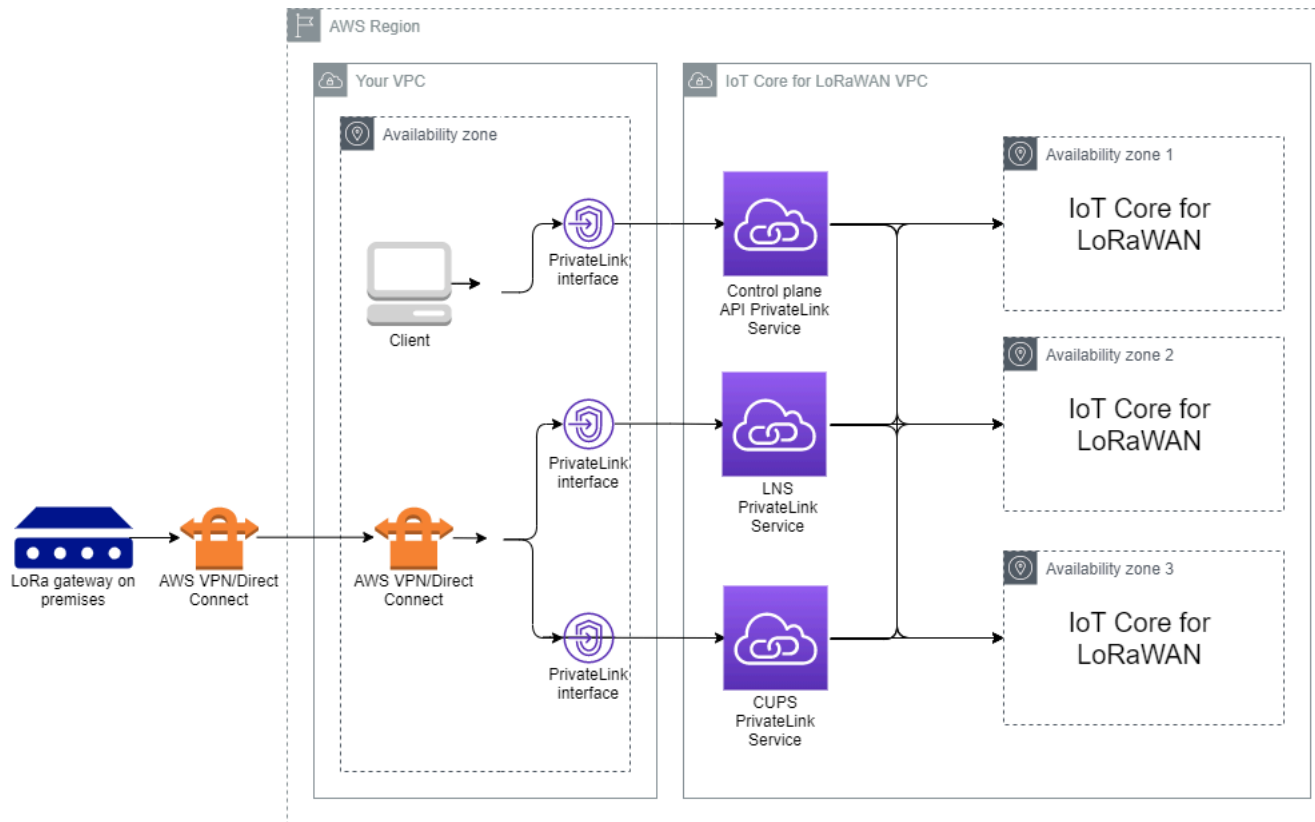
Considerations for AWS IoT Wireless VPC endpoints

Before you set up an interface VPC endpoint for AWS IoT Wireless, ensure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

AWS IoT Wireless supports making calls to all of its API actions from your VPC. VPC endpoint policies are not supported for AWS IoT Wireless. By default, full access to AWS IoT Wireless is allowed through the endpoint. For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

AWS IoT Core for LoRaWAN privatelink architecture

The following diagram shows the privatelink architecture of AWS IoT Core for LoRaWAN. The architecture uses a Transit Gateway and Route 53 Resolver to share the AWS PrivateLink interface endpoints between your VPC, the AWS IoT Core for LoRaWAN VPC, and an on-premises environment. You'll find a more detailed architecture diagram when setting up the connection to the VPC interface endpoints.



AWS IoT Core for LoRaWAN endpoints

AWS IoT Core for LoRaWAN has three public endpoints. Each public endpoint has a corresponding VPC interface endpoint. The public endpoints can be classified into control plane and data plane endpoints. For information about these endpoints, see [AWS IoT Core for LoRaWAN API endpoints](#).

- **Control plane API endpoints**

You can use control plane API endpoints to interact with the AWS IoT Wireless APIs. These endpoints can be accessed from a client that is hosted in your Amazon VPC by using AWS PrivateLink.

- **Data plane API endpoints**

Data plane API endpoints are LoRaWAN Network Server (LNS) and Configuration and Update Server (CUPS) endpoints that you can use to interact with the AWS IoT Core for LoRaWAN LNS and CUPS endpoints. These endpoints can be accessed from your LoRa gateways on premises by using AWS VPN or AWS Direct Connect. You get these endpoints when onboarding your gateway to AWS IoT Core for LoRaWAN. For more information, see [Add a gateway to AWS IoT Core for LoRaWAN](#).

Topics

- [Onboard AWS IoT Core for LoRaWAN control plane API endpoint](#)
- [Onboard AWS IoT Core for LoRaWAN data plane API endpoints](#)

Onboard AWS IoT Core for LoRaWAN control plane API endpoint

You can use AWS IoT Core for LoRaWAN control plane API endpoints to interact with the AWS IoT Wireless APIs. For example, you can use this endpoint to run the [SendDataToWirelessDevice](#) API to send data from AWS IoT to your LoRaWAN device. For more information, see [AWS IoT Core for LoRaWAN Control Plane API Endpoints](#).

You can use the client hosted in your Amazon VPC to access the control plane endpoints that are powered by AWS PrivateLink. You use these endpoints to connect to the AWS IoT Wireless API through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the public internet.

To onboard the control plane endpoint:

- [Create your Amazon VPC and subnet](#)
- [Launch an Amazon EC2 instance in your subnet](#)
- [Create Amazon VPC interface endpoint](#)
- [Test your connection to the interface endpoint](#)

Create your Amazon VPC and subnet

Before you can connect to the interface endpoint, you must create a VPC and subnet. You'll then launch an EC2 instance in your subnet, which you can use to connect to the interface endpoint.

To create your VPC:

1. Navigate to the [VPCs](#) page of the Amazon VPC console and choose **Create VPC**.
2. On the **Create VPC** page:
 - Enter a name for **VPC Name tag - optional** (for example, **VPC-A**).
 - Enter an IPv4 address range for your VPC in the IPv4 CIDR block (for example, **10.100.0.0/16**).
3. Keep the default values for other fields and choose **Create VPC**.

To create your subnet:

1. Navigate to the [Subnets](#) page of the Amazon VPC console and choose **Create subnet**.
2. On the **Create subnet** page:
 - For **VPC ID**, choose the VPC that you created earlier (for example, VPC-A).
 - Enter a name for **Subnet name** (for example, **Private subnet**).
 - Choose the **Availability Zone** for your subnet.
 - Enter your subnet's IP address block in the **IPv4 CIDR block** in CIDR format (for example, **10.100.0.0/24**).
3. To create your subnet and add it to your VPC, choose **Create subnet**.

For more information, see [Work with VPCs and subnets](#).

Launch an Amazon EC2 instance in your subnet

To launch your EC2 instance:

1. Navigate to the [Amazon EC2](#) console and choose **Launch Instance**.
2. For AMI, choose **Amazon Linux 2 AMI (HVM), SSD Volume Type** and then choose the **t2 micro** instance type. To configure the instance details, choose **Next**.
3. In the **Configure Instance Details** page:
 - For **Network**, choose the VPC that you created earlier (for example, VPC-A).
 - For **Subnet**, choose the subnet that you created earlier (for example, **Private subnet**).
 - For **IAM role**, choose the role **AWSIoTWirelessFullAccess** to grant AWS IoT Core for LoRaWAN full access policy. For more information, see [AWSIoTWirelessFullAccess policy summary](#).
 - For **Assume Private IP**, use an IP address, for example, **10.100.0.42**.
4. Choose **Next: Add Storage** and then choose **Next: Add Tags**. You can optionally add any tags to associate with your EC2 instance. Choose **Next: Configure Security Group**.
5. In the **Configure Security Group** page, configure the security group to allow:
 - Open **All TCP** for Source as **10.200.0.0/16**.
 - Open **All ICMP - IPV4** for Source as **10.200.0.0/16**.
6. To review the instance details and launch your EC2 instance, choose **Review and Launch**.

For more information, see [Get started with Amazon EC2 Linux instances](#).

Create Amazon VPC interface endpoint

You can create a VPC endpoint for your VPC, which can then be accessed by the EC2 API. To create the endpoint:

1. Navigate to the [VPC Endpoints](#) console and choose **Create Endpoint**.
2. In the **Create Endpoint** page, specify the following information.
 - Choose **AWS services** for **Service category**.
 - For **Service Name**, search by entering the keyword **iotwireless**. In the list of `iotwireless` services displayed, choose the control plane API endpoint for your Region. The endpoint will be in the format `com.amazonaws.region.iotwireless.api`.
 - For **VPC** and **Subnets**, choose the VPC where you want to create the endpoint, and the Availability Zones (AZs) in which you want to create the endpoint network.

Note

The `iotwireless` service might not support all Availability Zones.

- For **Enable DNS name**, choose **Enable for this endpoint**.

Choosing this option will automatically resolve the DNS and create a route in Amazon Route 53 Public Data Plane so that the APIs you use later to test the connection will go through the `privatelink` endpoints.
 - For **Security group**, choose the security groups you want to associate with the endpoint network interfaces.
 - Optionally, you can add or remove tags. Tags are name-value pairs that you use to associate with your endpoint.
3. To create your VPC endpoint, choose **Create endpoint**.

Test your connection to the interface endpoint

You can use an SSH to access your Amazon EC2 instance and then use the AWS CLI to connect to the `privatelink` interface endpoints.

Before you connect to the interface endpoint, download the most recent AWS CLI version by following the instructions described in [Installing, updating, and uninstalling AWS CLI version 2 on Linux](#).

The following examples show how you can test your connection to the interface endpoint using the CLI.

```
aws iotwireless create-service-profile \  
  --endpoint-url https://api.iotwireless.region.amazonaws.com \  
  --name='test-privatelink'
```

The following shows an example of running the command.

```
Response:  
{  
  "Arn": "arn:aws:iotwireless:region:acct_number:ServiceProfile/1a2345ba-4c5d-67b0-ab67-  
e0c8342f2857",  
  "Id": "1a2345ba-4c5d-67b0-ab67-e0c8342f2857"  
}
```

Similarly, you can run the following commands to get the service profile information or list all service profiles.

```
aws iotwireless get-service-profile \  
  --endpoint-url https://api.iotwireless.region.amazonaws.com  
  --id="1a2345ba-4c5d-67b0-ab67-e0c8342f2857"
```

The following shows an example for the list-device-profiles command.

```
aws iotwireless list-device-profiles \  
  --endpoint-url https://api.iotwireless.region.amazonaws.com
```

Onboard AWS IoT Core for LoRaWAN data plane API endpoints

AWS IoT Core for LoRaWAN data plane endpoints consist of the following endpoints. You get these endpoints when adding your gateway to AWS IoT Core for LoRaWAN. For more information, see [Add a gateway to AWS IoT Core for LoRaWAN](#).

- **LoRaWAN Network Server (LNS) endpoints**

The LNS endpoints are of the format *account-specific-prefix*.lns.lorawan.*region*.amazonaws.com. You can use this endpoint to establish a connection for exchanging LoRa uplink and downlink messages.

- **Configuration and Update Server (CUPS) endpoints**

The CUPS endpoints are of the format *account-specific-prefix*.cups.lorawan.*region*.amazonaws.com. You can use this endpoint for credentials management, remote configuration, and firmware update of gateways.

For more information, see [Using CUPS and LNS protocols](#).

To find the Data Plane API endpoints for your AWS account and Region, use the [get-service-endpoint](#) CLI command shown here, or the [GetServiceEndpoint](#) REST API. For more information, see [AWS IoT Core for LoRaWAN Data Plane API Endpoints](#).

You can connect your LoRaWAN gateway on premises to communicate with AWS IoT Core for LoRaWAN endpoints. To establish this connection, first connect your on premises gateway to your AWS account in your VPC by using a VPN connection. You can then communicate with the data plane interface endpoints in the AWS IoT Core for LoRaWAN VPC that are powered by privatelink.

The following shows how to onboard these endpoints.

- [Create VPC interface endpoint and private hosted zone](#)
- [Use VPN to connect LoRa gateways to your AWS account](#)

Create VPC interface endpoint and private hosted zone

AWS IoT Core for LoRaWAN has two data plane endpoints, Configuration and Update Server (CUPS) endpoint and LoRaWAN Network Server (LNS) endpoint. The setup process to establish a privatelink connection to both endpoints is the same, so we can use the LNS endpoint for illustration purposes.

For your data plane endpoints, the LoRa gateways first connect to your AWS account in your Amazon VPC, which then connects to the VPC endpoint in the AWS IoT Core for LoRaWAN VPC.

When connecting to the endpoints, the DNS names can be resolved within one VPC but can't be resolved across multiple VPCs. To disable private DNS when creating the endpoint, disable the **Enable DNS name** setting. You can use private hosted zone to provide information about how you want Route 53 to respond to DNS queries for your VPCs. To share your VPC with an on-premises environment, you can use a Route 53 Resolver to facilitate hybrid DNS.

To complete this procedure, perform the following steps.

- [Create an Amazon VPC and subnet](#)

- [Create an Amazon VPC interface endpoint](#)
- [Configure private hosted zone](#)
- [Configure Route 53 inbound resolver](#)
- [Next steps](#)

Create an Amazon VPC and subnet

You can reuse your Amazon VPC and subnet that you created when onboarding your control plane endpoint. For information, see [Create your Amazon VPC and subnet](#).

Create an Amazon VPC interface endpoint

You can create a VPC endpoint for your VPC, which is similar to how you would create one for your control plane endpoint.

1. Navigate to the [VPC Endpoints](#) console and choose **Create Endpoint**.
2. In the **Create Endpoint** page, specify the following information.
 - Choose **AWS services** for **Service category**.
 - For **Service Name**, search by entering the keyword **lns**. In the list of lns services displayed, choose the LNS data plane API endpoint for your Region. The endpoint will be of the format `com.amazonaws.region.lorawan.lns`.

Note

If you're following this procedure for your CUPS endpoint, search for cups. The endpoint will be of the format `com.amazonaws.region.lorawan.cups`.

- For **VPC** and **Subnets**, choose the VPC where you want to create the endpoint, and the Availability Zones (AZs) in which you want to create the endpoint network.

Note

The `iotwireless` service might not support all Availability Zones.

- For **Enable DNS name**, make sure that **Enable for this endpoint** is not selected.

By not selecting this option, you can disable private DNS for the VPC endpoint and use private hosted zone instead.

- For **Security group**, choose the security groups you want to associate with the endpoint network interfaces.
 - Optionally, you can add or remove tags. Tags are name-value pairs that you use to associate with your endpoint.
3. To create your VPC endpoint, choose **Create endpoint**.

Configure private hosted zone

After you create the privatelink endpoint, in the **Details** tab of your endpoint, you'll see a list of DNS names. You can use one of these DNS names to configure your private hosted zone. The DNS name will be of the format `vpce-xxxx.lns.lorawan.region.vpce.amazonaws.com`.

Create the private hosted zone

To create the private hosted zone:

1. Navigate to the [Route 53 Hosted zones](#) console and choose **Create hosted zone**.
2. In the **Create hosted zone** page, specify the following information.
 - For **Domain name**, enter the full service name for your LNS endpoint, `lns.lorawan.region.amazonaws.com`.

Note

If you're following this procedure for your CUPS endpoint, enter `cups.lorawan.region.amazonaws.com`.

- For **Type**, choose **Private hosted zone**.
 - Optionally, you can add or remove tags to associate with your hosted zone.
3. To create your private hosted zone, choose **Create hosted zone**.

For more information, see [Creating a private hosted zone](#).

After you have created a private hosted zone, you can create a record that tells the DNS how you want traffic to be routed to that domain.

Create a record

After you have created a private hosted zone, you can create a record that tells the DNS how you want traffic to be routed to that domain. To create a record:

1. In the list of hosted zones displayed, choose the private hosted zone that you created earlier and choose **Create record**.
2. Use the wizard method to create the record. If the console presents you the **Quick create** method, choose **Switch to wizard**.
3. Choose **Simple Routing** for **Routing policy** and then choose **Next**.
4. In the **Configure records** page, choose **Define simple record**.
5. In the **Define simple record** page:
 - For **Record name**, enter the alias of your AWS account number. You get this value when onboarding your gateway or by using the [GetServiceEndpoint](#) REST API.
 - For **Record type**, keep the value as A - Routes traffic to an IPv4 address and some AWS resources.
 - For **Value/Route traffic to**, choose **Alias to VPC endpoint**. Then choose your **Region** and then choose the endpoint that you created previously, as described in [Create an Amazon VPC interface endpoint](#) from the list of endpoints displayed.
6. Choose **Define simple record** to create your record.

Configure Route 53 inbound resolver

To share a VPC endpoint to an on-premises environment, a Route 53 Resolver can be used to facilitate hybrid DNS. The inbound resolver will enable you to route traffic from the on-premises network to the data plane endpoints without going over the public internet. To return the private IP address values for your service, create the Route 53 Resolver in the same VPC as the VPC endpoint.

When you create the inbound resolver, you only have to specify your VPC and the subnets that you created previously in your Availability Zones (AZs). The Route 53 Resolver uses this information to automatically assign an IP address to route traffic to each of the subnets.

To create the inbound resolver:

1. Navigate to the [Route 53 Inbound endpoints](#) console and choose **Create inbound endpoint**.

Note

Make sure that you're using the same AWS Region that you used when creating the endpoint and private hosted zone.

2. In the **Create inbound endpoint** page, specify the following information.
 - Enter a name for **Endpoint name** (for example, **VPC_A_Test**).
 - For **VPC in the region**, choose the same VPC that you used when creating the VPC endpoint.
 - Configure the **Security group for this endpoint** to allow incoming traffic from the on premises network.
 - For IP address, choose **Use an IP address that is selected automatically**.
3. Choose **Submit** to create your inbound resolver.

For this example, let's assume that the IP addresses `10.100.0.145` and `10.100.192.10` were assigned for the inbound Route 53 Resolver for routing traffic.

Next steps

You've created the private hosted zone and an inbound resolver to route traffic for your DNS entries. You can now use either a Site-to-Site VPN or a Client VPN endpoint. For more information, see [Use VPN to connect LoRa gateways to your AWS account](#).

Use VPN to connect LoRa gateways to your AWS account

To connect your gateways on premises to your AWS account, you can use either a Site-to-Site VPN connection or a Client VPN endpoint.

Before you can connect your on premises gateways, you must have created the VPC endpoint, and configured a private hosted zone and inbound resolver so that traffic from the gateways don't go over the public internet. For more information, see [Create VPC interface endpoint and private hosted zone](#).

Site-to-Site VPN endpoint

If you don't have the gateway hardware or want to test the VPN connection using a different AWS account, you can use a Site-to-Site VPN connection. You can use Site-to-Site VPN to connect to the VPC endpoints from the same AWS account or another AWS account that you might be using in a different AWS Region.

Note

If you've the gateway hardware with you and want to set up a VPN connection, we recommend that you use Client VPN instead. For instructions, see [Client VPN endpoint](#).

To set up a Site-to-Site VPN:

1. Create another VPC in the site from which you want to set up the connection. For VPC-A, you can reuse the VPC that you created previously. To create another VPC (for example, VPC-B), use a CIDR block that doesn't overlap with the CIDR block of the VPC you created previously.

For information about setting up the VPCs, follow the instructions described in [AWS setup Site-to-Site VPN connection](#).

Note

The Site-to-Site VPN VPN method described in the document uses OpenSWAN for the VPN connection, which supports only one VPN tunnel. If you use a different commercial software for the VPN, you might be able to set up two tunnels between the sites.

2. After you set up the VPN connection, update the `/etc/resolv.conf` file by adding the inbound resolver's IP address from your AWS account. You use this IP address for the nameserver. For information about how to obtain this IP address, see [Configure Route 53 inbound resolver](#). For this example, we can use the IP address `10.100.0.145` that was assigned when you created the Route 53 Resolver.

```
options timeout:2 attempts:5
; generated by /usr/sbin/dhclient-script
search region.compute.internal
nameserver 10.100.0.145
```

3. We can now test whether the VPN connection uses the AWS PrivateLink endpoint instead of going over the public internet by using an `nslookup` command. The following shows an example of running the command.

```
nslookup account-specific-prefix.lns.lorawan.region.amazonaws.com
```


The following shows an example output of running the command, which shows a private IP address indicating that the connection has been established to the AWS PrivateLink LNS endpoint.

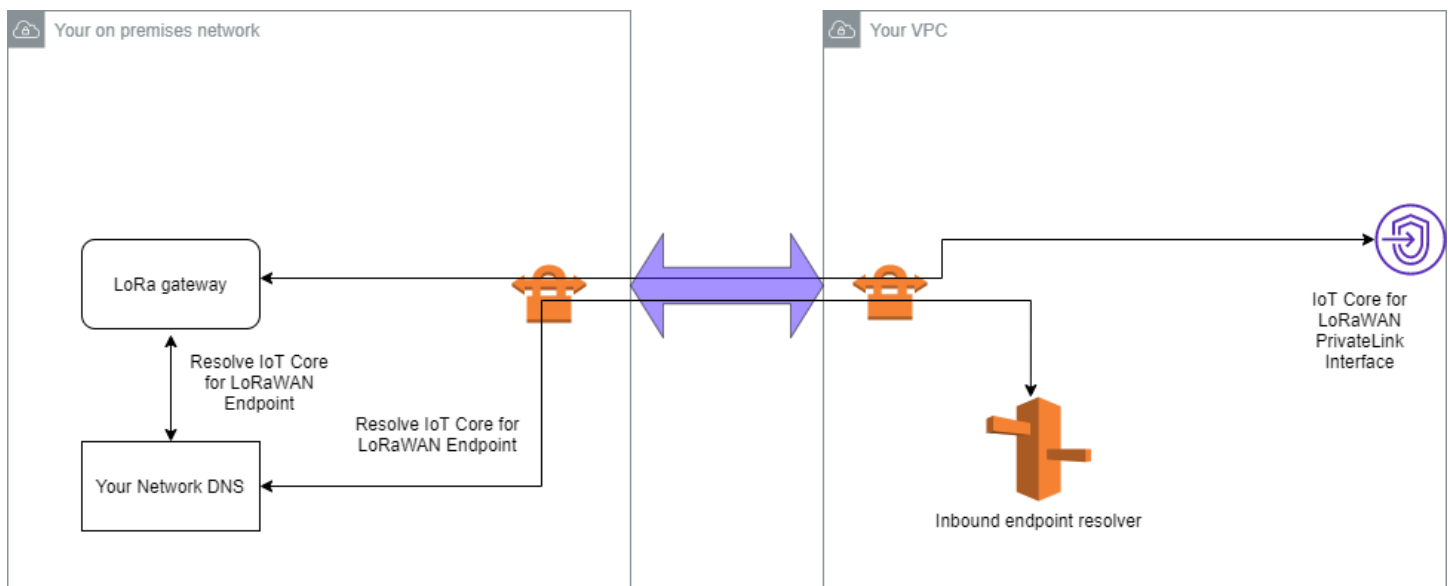
```
Server: 10.100.0.145
Address: 10.100.0.145

Non-authoritative answer:
Name: https://xxxxx.lns.lorawan.region.amazonaws.com
Address: 10.100.0.204
```

For information about using a Site-to-Site VPN connection, see [How Site-to-Site VPN works](#).

Client VPN endpoint

AWS Client VPN is a managed client-based VPN service that enables you to securely access AWS resources and resources in your on-premises network. The following shows the architecture for the client VPN service.



To establish a VPN connection to a Client VPN endpoint:

1. Create a Client VPN endpoint by following the instructions described in [Getting started with AWS Client VPN](#).
2. Log in to your on-premises network (for example, a Wi-Fi router) by using the access URL for that router (for example, 192.168.1.1), and find the root name and password.

3. Set up your LoRaWAN gateway by following the instructions in the gateway's documentation and then add your gateway to AWS IoT Core for LoRaWAN. For information about how to add your gateway, see [Onboard your gateways to AWS IoT Core for LoRaWAN](#).
4. Check whether your gateway's firmware is up to date. If the firmware is out of date, you can follow the instructions provided in the on-premises network to update your gateway's firmware. For more information, see [Update gateway firmware using CUPS service with AWS IoT Core for LoRaWAN](#).
5. Check whether OpenVPN has been enabled. If it has been enabled, skip to the next step to configure the OpenVPN client inside the on-premises network. If it hasn't been enabled, follow the instructions in [Guide to install OpenVPN for OpenWrt](#).

Note

For this example, we use OpenVPN. You can use other VPN clients such as AWS VPN or AWS Direct Connect to set up your Client VPN connection.

6. Configure the OpenVPN client based on information from the client configuration and how you can use [OpenVPN client using LuCi](#).
7. SSH to your on-premises network and update the `/etc/resolv.conf` file by adding the IP address of the inbound resolver in your AWS account (10.100.0.145).
8. For the gateway traffic to use AWS PrivateLink to connect to the endpoint, replace the first DNS entry for your gateway to the inbound resolver's IP address.

For information about using a Site-to-Site VPN connection, see [Getting started with Client VPN](#).

Connect to LNS and CUPS VPC endpoints

The following shows how you can test your connection to the LNS and CUPS VPC endpoints.

Test CUPS endpoint

To test your AWS PrivateLink connection to the CUPS endpoint from your LoRa gateway, run the following command:

```
curl -k -v -X POST https://xxxx.cups.region.iotwireless.iot:443/update-info
  --cacert cups.trust --cert cups.crt --key cups.key --header "Content-Type:
application/json"
  --data '{
```

```
    "router": "xxxxxxxxxxxxx",
    "cupsUri": "https://xxxx.cups.lorawan.region.amazonaws.com:443",
    "cupsCredCrc":1234, "tcCredCrc":552384314
  }'
-output cups.out
```

Test LNS endpoint

To test your LNS endpoint, first provision a LoRaWAN device that will work with your wireless gateway. You can then add your device and perform the *join* procedure after which you can start sending uplink messages.

AWS IoT Core for Amazon Sidewalk

AWS IoT Core for Amazon Sidewalk provides the cloud services that you can use to connect your Sidewalk end devices to the AWS Cloud and use other AWS services.

Amazon Sidewalk is a secure, shared network that enables devices in your community to get connected and stay connected. Amazon Sidewalk transfers data between Sidewalk end devices and Sidewalk gateways, and between Sidewalk gateways and the Sidewalk cloud.

Accessing AWS IoT Core for Amazon Sidewalk

You can onboard your Sidewalk end devices to AWS IoT by using the console or the AWS IoT Wireless API operations. After your devices are onboarded, their messages are sent to AWS IoT Core. You can then start developing your business applications on the AWS Cloud, which uses the data from your Amazon Sidewalk end devices.

Using the console

To onboard your Sidewalk end devices, sign in to the AWS Management Console and navigate to the [Devices](#) page on the AWS IoT console. After your devices are onboarded, you can view and manage them on this page of the IoT console.

Using the API or CLI

You can onboard both Sidewalk and LoRaWAN devices by using the [AWS IoT Wireless API operations](#). The AWS IoT Wireless API that AWS IoT Core is built on is supported by the AWS SDK. For more information, see [AWS SDKs and Toolkits](#).

You can use the AWS CLI to run commands for onboarding and managing your Sidewalk end devices. For more information, see [AWS IoT Wireless CLI reference](#).

AWS IoT Core for Amazon Sidewalk Regions and endpoints

Amazon Sidewalk is only available in the us-east-1 AWS Region. AWS IoT Core for Amazon Sidewalk provides support for control plane and data plane API endpoints in this Region. The data plane API endpoints are specific to your AWS account. For more information, see [AWS IoT Wireless Service endpoints](#) in the *AWS General Reference*.

AWS IoT Core for Amazon Sidewalk has quotas that apply to device data that is transmitted between the device and the AWS Cloud, and the maximum TPS for the AWS IoT Wireless API operations. For more information, see [AWS IoT Wireless quotas](#) in the *AWS General Reference*.

AWS IoT Core for Amazon Sidewalk pricing

When you sign up for AWS, you can get started with AWS IoT Core for Amazon Sidewalk for no charge by using the [AWS Free Tier](#).

For more information about general product overview and pricing, see [AWS IoT Core pricing](#).

What is AWS IoT Core for Amazon Sidewalk?

With AWS IoT Core for Amazon Sidewalk, you can onboard your Amazon Sidewalk end devices to AWS IoT and manage and monitor them. It also manages the destinations that send device data to other AWS services.

Features of AWS IoT Core for Amazon Sidewalk

Using AWS IoT Core for Amazon Sidewalk, you can:

- Onboard your Sidewalk end devices to AWS IoT using the AWS IoT console, AWS IoT Core for Amazon Sidewalk API operations, or AWS CLI commands.
- Leverage the capabilities offered by the AWS Cloud.
- Create a destination that uses AWS IoT rules to process incoming payload messages and to interact with other AWS services.
- Enable event notifications to receive messages about events such as when your Sidewalk end device has been provisioned or registered, or whether a downlink message has been successfully delivered to your device.
- Log and monitor your Sidewalk end devices in real time, obtain useful insights, and identify and troubleshoot errors.
- Associate your Sidewalk end devices with an AWS IoT thing, which helps you store a representation of your device on the cloud. Things in AWS IoT make it easier to search and manage your features, and access other AWS IoT Core features.

The following topics will help you learn about Amazon Sidewalk and AWS IoT Core for Amazon Sidewalk.

Topics

- [What is Amazon Sidewalk?](#)
- [How AWS IoT Core for Amazon Sidewalk works](#)

What is Amazon Sidewalk?

Amazon Sidewalk is a secure community network that uses Amazon Sidewalk Bridges, such as compatible Amazon Echo and Ring devices, to provide cloud connectivity for IoT devices. Amazon Sidewalk enables low-bandwidth and long-range connectivity at home and beyond using Bluetooth LE for short-distance communication and LoRa and FSK radio protocols at 900MHz frequencies to cover longer distances.

When Amazon Sidewalk is enabled, this network can support other Sidewalk end devices in your community, and can be used for applications such as sensing your environment. Amazon Sidewalk helps your devices get and stay connected.

Features of Amazon Sidewalk

The following are features of Amazon Sidewalk.

- Amazon Sidewalk creates a low-bandwidth network using Sidewalk gateways that include Ring and select Echo devices. Using gateways, you can share a portion of your internet bandwidth, which is then used to connect your end devices to the network.
- Amazon Sidewalk offers a secure networking mechanism with multiple layers of encryption and security.
- Amazon Sidewalk offers a simple mechanism to enable or disable participation in Sidewalk.

Amazon Sidewalk concepts

The following are some key concepts of Amazon Sidewalk.

Sidewalk gateways

Sidewalk gateways, or Amazon Sidewalk bridges, route data between your Sidewalk end devices and the cloud. Gateways are Amazon devices, such as the Echo device or the Ring Floodlight Cam, that support SubG-CSS (asynchronous, LDR), SubG-FSK (synchronous, HDR), or Bluetooth LE for Sidewalk communication. Sidewalk gateways share a portion of your internet bandwidth with the Sidewalk community to provide connectivity to a group of Sidewalk-enabled devices.

Sidewalk end devices

Sidewalk end devices roam on Amazon Sidewalk by connecting to Sidewalk gateways. The end devices are low-bandwidth, low-power smart products, such as Sidewalk-enabled lights or door locks.

Note

Certain Sidewalk gateways can also act as end devices.

Sidewalk Network Server

The Sidewalk Network Server, operated by Amazon, verifies the incoming packets and routes uplink and downlink messages to the desired destination, while keeping the Sidewalk network time-synchronized.

Learn more about Amazon Sidewalk

For more information about Amazon Sidewalk, see the following web pages:

- [Amazon Sidewalk](#)
- [Amazon Sidewalk documentation](#)
- [AWS IoT Core for Amazon Sidewalk](#)

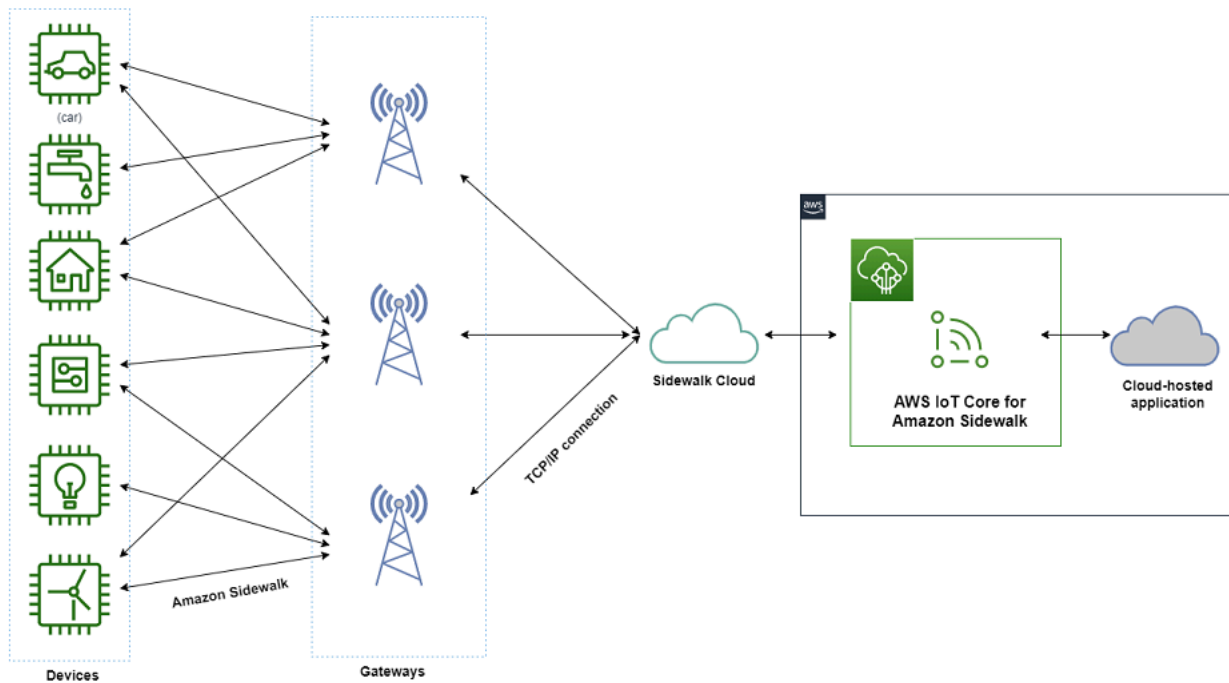
How AWS IoT Core for Amazon Sidewalk works

With AWS IoT Core for Amazon Sidewalk, you can onboard your Amazon Sidewalk end devices to AWS IoT and manage and monitor them. It also manages the destinations that send device data to other AWS services

AWS IoT Core for Amazon Sidewalk provides the cloud services that you can use to connect your Sidewalk end devices to the AWS Cloud and use other AWS services. You can also use AWS IoT Core for Amazon Sidewalk to manage your Sidewalk devices, and monitor and build applications on them.

Sidewalk end devices communicate with AWS IoT Core through Sidewalk gateways. AWS IoT Core for Amazon Sidewalk manages the service and device policies that AWS IoT Core requires

to manage and communicate with the Sidewalk end devices and gateways. It also manages the destinations that send device data to other AWS services.



Get started using AWS IoT Core for Amazon Sidewalk

You can use the AWS IoT console, the AWS IoT Core for Amazon Sidewalk API, or the AWS CLI to create and onboard Sidewalk end devices and connect them to the Sidewalk network. For information about getting started with Amazon Sidewalk and onboarding end devices to AWS IoT, see the following topics.

- [Getting started with AWS IoT Core for Amazon Sidewalk](#)

This topic walks through the prerequisites for onboarding your Sidewalk end devices, illustrates the workflow using a sensor monitoring application, and provides an overview of how to onboard your device using AWS CLI commands.

- [Connecting to AWS IoT Core for Amazon Sidewalk](#)

This section describes the different steps in the onboarding workflow introduction, and walks through onboarding your end devices using the console, and the API operations. You'll also connect your device and view messages that are exchanged between your device and AWS IoT Core for Amazon Sidewalk.

- [Bulk provisioning devices with AWS IoT Core for Amazon Sidewalk](#)

This section provides a detailed step-by-step tutorial for bulk provisioning your Sidewalk end devices using AWS IoT Core for Amazon Sidewalk. You'll learn the bulk provisioning workflow, and how to onboard a large number of Sidewalk devices.

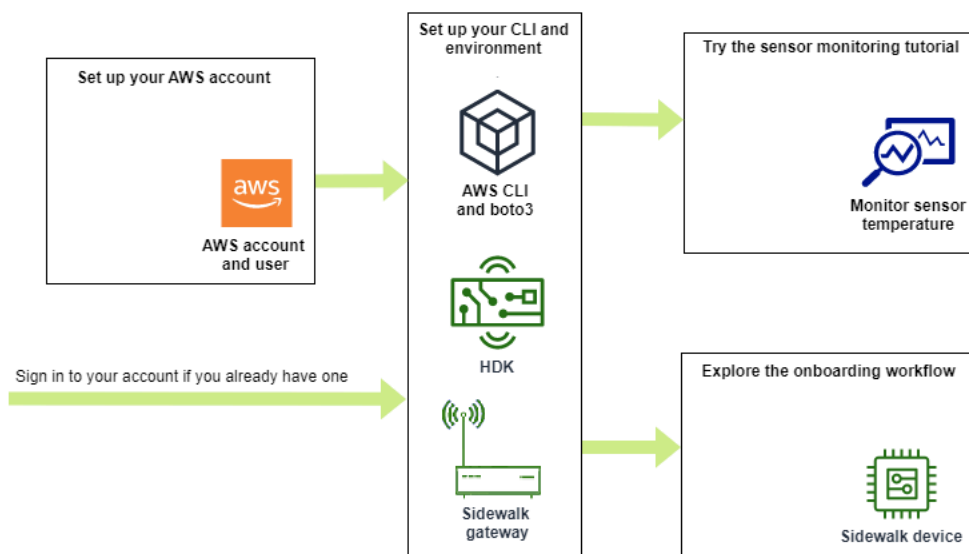
Learn more about AWS IoT Core for Amazon Sidewalk

For more information about AWS IoT Core for Amazon Sidewalk, see the following web pages:

- [Amazon Sidewalk](#)
- [Amazon Sidewalk documentation](#)
- [AWS IoT Core for Amazon Sidewalk](#)

Getting started with AWS IoT Core for Amazon Sidewalk

This section shows you how to get started with connecting your Sidewalk end devices to AWS IoT Core for Amazon Sidewalk. It explains how you can connect an end device to Amazon Sidewalk and pass messages between them. You'll also learn about the Sidewalk sample application and an overview of how to perform sensor monitoring using AWS IoT Core for Amazon Sidewalk. The sample application provides you with a dashboard to view and monitor changes to the sensor temperature.



The following topics will help you get started with AWS IoT Core for Amazon Sidewalk.

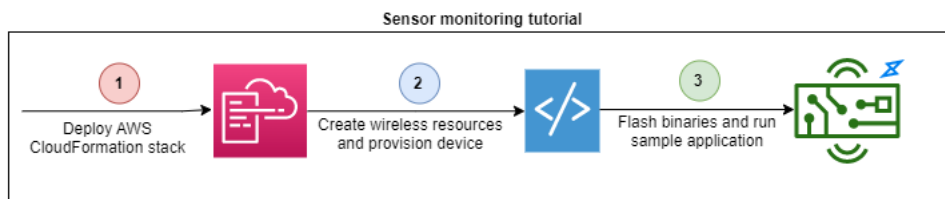
Topics

- [Try the sensor monitoring tutorial](#)
- [Introduction to onboarding your Sidewalk devices](#)

Try the sensor monitoring tutorial

This section provides you an overview of the Amazon Sidewalk sample application on GitHub that shows you how to monitor the temperature of a sensor. In this tutorial, you use scripts that programmatically create the required wireless resources, provision the end device and flash the binaries, and then connect your end device to the application. The scripts that use the AWS CLI and Python commands create an AWS CloudFormation stack and wireless resources, and then flash the binaries and deploy the application onto your hardware development kit (HDK).

The following diagram shows the steps are involved when you run the [sample application](#) and connect your Sidewalk end device to the application. For detailed instructions including pre-requisites and configuration for this tutorial, see the [README document](#) in *GitHub*.



Introduction to onboarding your Sidewalk devices

This section shows you how to onboard your Sidewalk end devices to AWS IoT Core for Amazon Sidewalk. To onboard your devices, first add your Sidewalk device, then provision and register your device, and then connect your hardware to the cloud application. Before running this tutorial, review and complete [Installing Python and the AWS CLI](#).

The following steps show you how to onboard and connect your Sidewalk end devices to AWS IoT Core for Amazon Sidewalk. If you want to onboard devices using the AWS CLI, you can refer to the sample commands provided in this section. For information about onboarding devices using the AWS IoT console, see [Connecting to AWS IoT Core for Amazon Sidewalk](#).

⚠ Important

To perform the entire onboarding workflow, you also provision and register your end device, and connect your hardware development kit (HDK). For more information, see [Provisioning and registering your end device](#) in the *Amazon Sidewalk documentation*.

Topics

- [Step 1: Add your Sidewalk device to AWS IoT Core for Amazon Sidewalk](#)
- [Step 2: Create a destination for your Sidewalk end device](#)
- [Step 3: Provision and register the end device](#)
- [Step 4: Connect to Sidewalk end device and exchange messages](#)

Step 1: Add your Sidewalk device to AWS IoT Core for Amazon Sidewalk

Here's an overview of the steps that you'll perform to add your Sidewalk end device to AWS IoT Core for Amazon Sidewalk. Store the information you obtain about the device profile and the wireless device that you create. You'll use this information to provision and register the end device. For more information about these steps, see [Add your device to AWS IoT Core for Amazon Sidewalk](#).

1. Create a device profile

Create a device profile that contains the shared configurations for your Sidewalk devices. When creating the profile, specify a *name* for the profile as an alphanumeric string. To create a profile, either go to the [Sidewalk tab of the Profiles hub](#) in the AWS IoT console and choose **Create profile**, or use the [CreateDeviceProfile](#) API operation or the [create-device-profile](#) CLI command as shown in this example.

```
// Add your device profile using a name and the sidewalk object.  
aws iotwireless create-device-profile --name sidewalk_profile --sidewalk {}
```

2. Create your Sidewalk end device

Create your Sidewalk end device with AWS IoT Core for Amazon Sidewalk. Specify a destination name and the ID of the device profile obtained from the previous step. To add a device, either go to the [Sidewalk tab of the Devices hub](#) in the AWS IoT console and choose **Provision device**, or use the [CreateWirelessDevice](#) API operation or the [create-wireless-device](#) CLI command as shown in this example.

Note

Specify a name for your destination that's unique to your AWS account and AWS Region. You'll use the same destination name when you add your destination to AWS IoT Core for Amazon Sidewalk.

```
// Add your Sidewalk device by using the device profile ID.
aws iotwireless create-wireless-device --type "Sidewalk" --name sidewalk_device \
  --destination-name SidewalkDestination \
  --sidewalk DeviceProfileId="12345678-234a-45bc-67de-e8901234f0a1"
```

3. Get device profile and wireless device information

Get the device profile and wireless device information as a JSON. The JSON will contain information about the device details, device certificates, private keys, DeviceTypeId, and the Sidewalk manufacturing serial number (SMSN).

- If you're using the AWS IoT console, you can use the [Sidewalk tab of the Devices hub](#) to download a combined JSON file for your Sidewalk end device.
- If you're using the API operations, store the responses obtained from the API operations [GetDeviceProfile](#) and [GetWirelessDevice](#) as separate JSON files, such as *device_profile.json* and *wireless_device.json*.

```
// Store device profile information as a JSON file.
aws iotwireless get-device-profile \
  --id "12345678-a1b2-3c45-67d8-e90fa1b2c34d" > device_profile.json

// Store wireless device information as a JSON file.
aws iotwireless get-wireless-device --identifier-type WirelessDeviceId \
  --identifier "23456789-abcd-0123-bcde-fabc012345678" > wireless_device.json
```

Step 2: Create a destination for your Sidewalk end device

Here's an overview of the steps that you'll perform to add your destination to AWS IoT Core for Amazon Sidewalk. Using the AWS Management Console, or the AWS IoT Wireless API operations, or the AWS CLI, you run the following steps to create an AWS IoT rule and destination. You can then

connect to the hardware platform, and view and exchange messages. For a sample IAM role and AWS IoT rule used for the AWS CLI examples in this section, see [Create an IAM role and IoT rule for your destination](#).

1. Create IAM role

Create an IAM role that grants AWS IoT Core for Amazon Sidewalk permission to send data to the AWS IoT rule. To create the role, use the [CreateRole](#) API operation or [create-role](#) CLI command. You can name the role as *SidewalkRole*.

```
aws iam create-role --role-name lambda-ex \  
  --assume-role-policy-document file://lambda-trust-policy.json
```

2. Create a rule for the destination

Create an AWS IoT rule that will process the device's data and specify the topic to which messages are published. You'll observe messages on this topic after connecting to the hardware platform. Use the AWS IoT Core API operation, [CreateTopicRule](#), or the AWS CLI command, [create-topic-rule](#), to create a rule for the destination.

```
aws iot create-topic-rule --rule-name Sidewalkrule \  
  --topic-rule-payload file://myrule.json
```

3. Create a destination

Create a destination that associates your Sidewalk device with the IoT rule that processes it for use with other AWS services. You can add a destination using the [Destinations hub](#) of the AWS IoT console, or the [CreateDestination](#) API operation or the [create-destination](#) CLI command.

```
aws iotwireless create-destination --name SidewalkDestination \  
  --expression-type RuleName --expression SidewalkRule \  
  --role-arn arn:aws:iam::123456789012:role/SidewalkRole
```

Step 3: Provision and register the end device

Using Python commands, you can provision and register your end device. The provisioning script uses the device JSON data that you obtained to generate a manufacturing binary image, which is then flashed on the hardware board. You then register your end device for connecting to the

hardware platform. For more information, see [Provisioning and registering your end device](#) in the *Amazon Sidewalk documentation*.

Note

When registering your Sidewalk end device, your gateway must be opted in to Amazon Sidewalk, and your gateway and device must be in range of each other.

Step 4: Connect to Sidewalk end device and exchange messages

After you've registered your end device, you can then connect your end device and start exchanging messages and device data.

1. Connect your Sidewalk end device

Connect the HDK to your computer and follow the instructions provided by the vendor documentation to connect to your HDK. For more information, see [Provisioning and registering your end device](#) in the *Amazon Sidewalk documentation*.

2. View and exchange messages

Use the MQTT client to subscribe to the topic specified in the rule and view the message received. You can also use the [SendDataToWirelessDevice](#) API operation or the [send-data-to-wireless-device](#) CLI command to send a downlink message to your device and verify the connectivity status.

(Optional) You can enable the message delivery status event to check whether the downlink message was successfully received.

```
aws iotwireless send-data-to-wireless-device \  
  --id "<Wireless_Device_ID>" \  
  --payload-data "SGVsbG8gVG8gRGV2c2lt" \  
  --wireless-metadata Sidewalk={Seq=1,AckModeRetryDurationSecs=10}
```

Connecting to AWS IoT Core for Amazon Sidewalk

This section shows you how to onboard your Sidewalk end device and then connect your device to the Sidewalk network. It describes the steps that you perform in the onboarding tutorial, as

mentioned in [Introduction to onboarding your Sidewalk devices](#). You'll learn how to onboard devices by using the AWS IoT console and the AWS IoT Core for Amazon Sidewalk API operations. You'll also learn about the AWS CLI commands that perform these operations.

Prerequisites

To add your end device and destination to AWS IoT Core for Amazon Sidewalk, you must set up your AWS account. To perform these operations using the AWS IoT Wireless API or the AWS CLI commands, you must also set up the AWS CLI. For more information about the prerequisites and setting up, see [Installing Python and the AWS CLI](#).

Note

To perform the entire onboarding workflow for provisioning and registering your end device, and connecting to your hardware development kit (HDK), you must also set up your Sidewalk gateway and HDK. For more information, see [Setting up the hardware development kit \(HDK\)](#) and [Setting up a Sidewalk gateway](#) in the *Amazon Sidewalk documentation*.

Describing your Sidewalk resources

Before you get started and create the resources, we recommend that you consider the naming convention of your Sidewalk end devices, device profiles, and destinations. AWS IoT Core for Amazon Sidewalk assigns a unique identifier to the resources that you create. However, you can give them more descriptive names, add a description, or add optional tags to help identify and manage them.

Note

The destination name can't be changed after it's created. Use a name that's unique to your AWS account and AWS Region.

For more information, see [Describing your AWS IoT Wireless resources](#).

Topics

- [Add your device to AWS IoT Core for Amazon Sidewalk](#)

- [Add a destination for your Sidewalk end device](#)
- [Connect your Sidewalk device and view uplink metadata format](#)

Add your device to AWS IoT Core for Amazon Sidewalk

Before creating a wireless device, first create a device profile. Device profiles define the device capabilities and other parameters for your Sidewalk devices. A single device profile can be associated with multiple devices.

After you create a device profile, when you retrieve information about the profile, it returns a `DeviceTypeId`. When you provision your end device, you'll use this ID, the device certificates, application server public key, and the SMSN.

How to create and add your device

1. Create a device profile for your Sidewalk end devices. Specify a profile name to use for your Sidewalk devices as an alphanumeric string. The profile will help identify the devices to associate it with.
 - (Console) When adding your Sidewalk device, you can also create a new profile. This helps you quickly add your device to AWS IoT Core for Amazon Sidewalk and associate it with a profile.
 - (API) Use the `CreateDeviceProfile` API operation by specifying a profile name and the Sidewalk object, `sidewalk {}`. The API response will contain a profile ID and ARN (Amazon Resource Name).
2. Add your wireless device to AWS IoT Core for Amazon Sidewalk. Specify a destination name and choose the device profile that you created in the previous step.
 - (Console) When adding your Sidewalk device, enter a destination name, and choose the profile that you created.
 - (API) Use the `CreateWirelessDevice` API operation. Specify a destination name and the ID of the device profile obtained previously.

Wireless device parameters

Parameter	Description	Notes
Destination name	The name of the destination that describes the AWS IoT rules for processing the	If you haven't already created a destination, you can provide any string value. AWS IoT Core for Amazon Sidewalk

Parameter	Description	Notes
	device's data that other AWS services will use.	will create an empty destination when creating the device, which you can then update when adding your destination.
Device profile	The device profile that you previously created.	–

- Obtain the JSON file that contains the required information for provisioning your end device.
 - (Console) Download this file from the details page of the Sidewalk device that you created.
 - (API) Use the `GetDeviceProfile` and `GetWirelessDevice` API operations to retrieve information about your device profile and wireless device. Store the API response information as JSON files, such as *device_profile.json* and *wireless_device.json*.

Add your device profile and Sidewalk end device


This section shows how you can create a device profile. It also shows how you can use the AWS IoT console and the AWS CLI to add your Sidewalk end device to AWS IoT Core for Amazon Sidewalk.

Add your Sidewalk device (console)

To add your Sidewalk device using the AWS IoT console, go to the [Sidewalk tab of the Devices hub](#), choose **Provision device**, and then perform the following steps.


LoRaWAN
Sidewalk

▼ How it works
With AWS IoT Core for Sidewalk, you can add your Sidewalk device fleet to the AWS Cloud. Use the following steps to get started.




Step 1. Add your Sidewalk device

First, create a device profile and retrieve the application server public key. Next, create your Sidewalk device and retrieve information about it, including device certificates and private keys.



Step 2. Provision & register your Sidewalk device

Provision your hardware as a Sidewalk endpoint by flashing the device certificates and the application server public key that you have generated. Register your device so that it can connect to AWS IoT Core for Amazon Sidewalk.



Step 3. Connect your Sidewalk endpoint to the cloud

Create a destination and use [AWS IoT Rules](#) to process and route data to other AWS services. Your endpoint can now exchange messages with your cloud application.

Sidewalk devices (2) [Info](#)

Provision and manage all your Sidewalk devices.

Edit
Delete
Provision device

< 1 >
⚙️

1. Specify device details

Specify the configuration information for your Sidewalk device. You can also create a new device profile, or choose an existing profile for your Sidewalk device.

- a. Specify a device name and optional description. The description can be up to 2,048 characters long. These fields can be edited after you create the device.
- b. Choose a device profile to associate with your Sidewalk device. If you have any existing device profiles, you can choose your profile. To create a new profile, choose **Create new profile**, and then enter a name for the profile.

Note

To attach tags to your device profile, after you create your profile, go to the [Profiles hub](#) and then edit your profile to add this information.

- c. Specify the name of your destination that will route messages from your device to other AWS services. If you haven't already created a destination, go to the [Destinations hub](#) to create your destination. You can then choose that destination for your Sidewalk device. For more information, see [Add a destination for your Sidewalk end device](#).
- d. Choose **Next** to continue adding your Sidewalk device.

2. Associate Sidewalk device with AWS IoT thing (Optional)

You can optionally associate your Sidewalk device to an AWS IoT thing. IoT things are entries in the AWS IoT device registry. Things make it easier to search and manage your devices. Associating a thing with your device lets your device access other AWS IoT Core features.

To associate your device with a thing, choose **Automatic thing registration**.

- a. Enter a unique name for the IoT thing that you want to associate your Sidewalk device. Thing names are case sensitive and must be unique in your AWS account and AWS Region.
- b. Provide any additional configurations for your IoT thing, such as using a thing type, or searchable attributes that can be used to filter from a list of things.
- c. Choose **Next** and verify the information about your Sidewalk device, and then choose **Create**.

Add your Sidewalk device (CLI)

To add your Sidewalk device and download the JSON files that will be used to provision your Sidewalk device, perform the following API operations.

Topics

- [Step 1: Create a device profile](#)
- [Step 2: Add your Sidewalk device](#)

Step 1: Create a device profile

To create a device profile in your AWS account, use the [CreateDeviceProfile](#) API operation or the [create-device-profile](#) CLI command. When creating your device profile, specify the name and provide any optional tags as name-value pairs.

For example, the following command creates a device profile for your Sidewalk devices:

```
aws iotwireless create-device-profile \  
  --name sidewalk_profile --sidewalk {}
```

Running this command returns the Amazon Resource Name (ARN) and the ID of the device profile as output.

```
{
  "DeviceProfileArn": "arn:aws:iotwireless:us-
east-1:123456789012:DeviceProfile/12345678-a1b2-3c45-67d8-e90fa1b2c34d",
  "DeviceProfileId": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
}
```

Step 2: Add your Sidewalk device

To add your Sidewalk device to your account for AWS IoT Core for Amazon Sidewalk, use the [CreateWirelessDevice](#) API operation or the [create-wireless-device](#) CLI command. When creating your device, specify the following parameters, in addition to an optional name and description for your Sidewalk device.

Note

If you want to associate your Sidewalk device with an AWS IoT thing, use the [AssociateWirelessDeviceWithThing](#) API operation or the [associate-wireless-device-with-thing](#) CLI command.

The following command shows an example of creating a Sidewalk device:

```
aws iotwireless create-wireless-device \
  --cli-input-json "file://device.json"
```

The following shows the contents of the file `device.json`.

Contents of device.json

```
{
  "Type": "Sidewalk",
  "Name": "SidewalkDevice",
  "DestinationName": "SidewalkDestination",
  "Sidewalk": {
    "DeviceProfileId": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
  }
}
```

Running this command returns the device ID and Amazon Resource Name (ARN) as output.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:WirelessDevice/23456789-
abcd-0123-bcde-fabc012345678",
  "Id": "23456789-abcd-0123-bcde-fabc012345678"
}
```

Obtain device JSON files for provisioning

After you've added your Sidewalk device to AWS IoT Core for Amazon Sidewalk, download the JSON file that contains the information required to provision your end device. You can retrieve this information using the AWS IoT console or the AWS CLI. For more information about how to provision the device, see [Provisioning and registering your end device](#) in the *Amazon Sidewalk documentation*.

Obtain JSON file (console)

To obtain the JSON file for provisioning your Sidewalk device:

1. Go to the [Sidewalk devices hub](#).
2. Choose the device that you added to AWS IoT Core for Amazon Sidewalk to view its details.
3. Obtain the JSON file by choosing **Download device JSON file** in the details page of the device that you added.

A `certificate.json` file will be downloaded that contains the required information for provisioning your end device. The following shows a sample JSON file. It contains the device certificates, private keys, the Sidewalk manufacturing serial number (SMSN), and the DeviceTypeID.

```
{
  "p256R1": "grg8izXoVvQ86cPvm0GMyWuZYHEBbbH ... DANKk0KoNT3bUGz+/f/pyTE
+xMRdIUBZ1Bw==",
  "eD25519": "grg8izXoVvQ86cPvm0GMyWuZYHEBbbHD ... UiZmntHiUr1GfkT0FMYqRB+Aw==",
  "metadata": {
    "devicetypeid": "fe98",
    "applicationDeviceArn": "arn:aws:iotwireless:us-
east-1:123456789012:WirelessDevice/897ce68e-3ca2-4ed0-85a2-30b0666c4052",
    "applicationDeviceId": "897ce68e-3ca2-4ed0-85a2-30b0666c4052",
    "smsn": "82B83C8B35E856F43CE9C3D59B418CC96B996071016DB1C3BE5901F0F3071A4A",
    "devicePrivKeyP256R1":
    "3e704bf8d319b3a475179f1d68c60737b28c708f845d0198f2d00d00c88ee018",

```

```
"devicePrivKeyEd25519":  
  "17dacb3a46ad9a42d5c520ca5f47f0167f59ce54d740aa13918465faf533b8d0"  
  },  
  "applicationServerPublicKey":  
  "5ce29b89c2e3ce6183b41e75fe54e45f61b8bb320efbdd2abd7aefa5957a316b"  
}
```

In the details page of your Sidewalk device, you'll also see information about:

- The device ID, its Amazon Resource Name (ARN), and details about any AWS IoT thing that the device is associated with.
- The device profile and destination details.
- The time at which the last uplink message was received from the device.
- The status that indicates whether your device has been provisioned or registered.

Obtain JSON file (CLI)

To obtain the JSON files for provisioning your Sidewalk end device using the AWS IoT Core for Amazon Sidewalk API or the AWS CLI, save the API response from retrieving information about your device profile and wireless device as JSON files, such as *wireless_device.json* and *device_profile.json* temporarily. You'll use them for provisioning your Sidewalk device.

The following shows how to retrieve the JSON files.

Topics

- [Step 1: Get device profile information as JSON file](#)
- [Step 2: Get Sidewalk device information as JSON file](#)

Step 1: Get device profile information as JSON file

Use the [GetDeviceProfile](#) API operation or the [get-device-profile](#) CLI command to get information about your device profile that you added to your account for AWS IoT Core for Amazon Sidewalk. To retrieve information about your device profile, specify the profile ID.

The API will then return information about the device profile matching the specified identifier and the device ID. You save this response information as a file, and give it a name such as *device_profile.json*.

The following shows an example CLI command:

```
aws iotwireless get-device-profile \  
  --id "12345678-a1b2-3c45-67d8-e90fa1b2c34d" > device_profile.json
```

Running this command returns the parameters of your device profile, the application server public key, and the DeviceTypeID. The following shows a JSON file that contains a sample response information from the API. For more information about the parameters in the API response, see [GetDeviceProfile](#).

GetDeviceProfile API response (Contents of *device_profile.json*)

```
{  
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/12345678-  
a1b2-3c45-67d8-e90fa1b2c34d",  
  "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",  
  "Name": "Sidewalk_profile",  
  "LoRaWAN": null,  
  "Sidewalk":  
  {  
    "ApplicationServerPublicKey":  
    "a123b45c6d78e9f012a34cd5e6a7890b12c3d45e6f78a1b234c56d7e890a1234",  
    "DAKCertificateMetadata": [  
      {  
        "DeviceTypeId": "fe98",  
        "CertificateId": "43564A6D2D50524F544F54595045",  
        "FactorySupport": false,  
        "MaxAllowedSignature": 1000  
      }  
    ],  
    "QualificationStatus": false  
  }  
}
```

Step 2: Get Sidewalk device information as JSON file

Use the [GetWirelessDevice](#) API operation or the [get-wireless-device](#) CLI command to get information about your Sidewalk device that you added to your account for AWS IoT Core for Amazon Sidewalk. To get information about your end device, provide the identifier of the wireless device that you obtained when adding your device.

The API will then return information about the device matching the specified identifier and the device ID. Save this response information as a JSON file. Give the file a meaningful name, such as *wireless_device.json*.

The following shows an example of running the command using the CLI:

```
aws iotwireless get-wireless-device --identifier-type WirelessDeviceId \
  --identifier "23456789-abcd-0123-bcde-fabc012345678" > wireless_device.json
```

Running this command returns the device details, device certificates, private keys, and the Sidewalk manufacturing serial number (SMSN). The following shows an example output of running this command. For more information about the parameters in the API response, see [GetWirelessDevice](#).

GetWirelessDevice API response (Contents of *wireless_device.json*)

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:WirelessDevice/23456789-
abcd-0123-bcde-fabc012345678",
  "Id": "23456789-abcd-0123-bcde-fabc012345678",
  "DestinationName": "SidewalkDestination",
  "Type": "Sidewalk",
  "Sidewalk": {
    "CertificateId": "4C7438772D50524F544F54595045",
    "DeviceCertificates": [
      {
        "SigningAlg": "Ed25519",

        "Value": "hDdkJw9L2uMCoRjImjMHqzNR6nYYh6QKncS15GthQN17NKe4ounb5UMQtLjnm7z0UPY0qghCeVOLCBUiQe2Z
F+GeItcafZcFKhS+05NPcVNR/fHYaf/cn5iUbRwLz/T
+ODXvGdwkBkgDyFgoUJgn7JdzFjaneE5qzTWXUbL79i1sXToGGjP8hiD9jJhidPWhIswLeydAWg010ZGA4CjzIaSGVM1Vta
uMMBfgAeL8Tdv5LkFIPIB3ZX9zt8zzmAuFRzI4MuNjWfIDn0F6AKu37WwU6/
QYhZoQrW9D/wndiCcsRGL+ANn367r/HE02Re4D0iCfs9f2rjc4LT1LKt7g/KW2ii+W
+9HYvvY0bBAI+AHx6Cx4j+djabTsvrgW2k6NU2zUSM7bdDP3z2a2+Z4WzBji/jYwt/
OP8rpsy5Ee4ywXUfCsF0rK0r0zay6yh27p3I3MZle2oC04JIlqK0VbIQqsXzSSyp6XXS0lhmuGugZ1AAADGz
+gFBeX/ZNN8VJwnsNfgzj4me1HgVJdUo4W9kvx9cr2jHWkC30j/bdBTh1+yBj0C53yHLQK/
11GhrEWiWPPnE434LRxnWkwr8EHD4oieJxC8fkIxxQfj+gHhU79Z
+oAAYAAAzsnf9SDIZPoDXF0TdC9P0qTglD0oXD12XPaVD4CvvLearr0SlFv+lsNbc4rgZn23MtIBM/7YQmJwmQ
+FXRup6Tkubg1hpz04J/09dxg8UiZmntHiUr1GfkTOFMYqRB+Aw=="
      }
    ],
    "SigningAlg": "P256r1",
  }
}
```



```

    "Value": "hDdkJw9L2uMCoRjImjMHqzNR6nYYh6QKncS15GthQNmHmGU8a
+S0qDXWwDnt3VSntpbTTQL7cMIusqweQo+JPXXWE1bGh7eaxPGz4ZeF5yM2cqVNUrQr1LX/6LZ
+0LuycrFrLzzB9APi0NIMLqV/Rt7XJssHQs2RPeT1uL/2XVpa6ztULJeQi2JwhTb/k48wbh/EvafG/
ibrIBIx9v7/
dwGRAPKHq7Uwb9hHnhpa8qN0UtjeUdIwJNh9vCBFX9s22t4PdortoFxbXo9C149PDDD4wqUHJGYLcsVX/
Sqqjf7Aug3h5dwdYN6cDgsuui0m0+aBcXBGpkh70xVxLwXkIP
+11dt23TkrSUKd0B01sc9Mc/0yEBCzx5RutKBwsefzy0L4vQX3AHgV7oD/XV73THMgGiDxQ55CPaaxN/
pm791VkQ76BSZaBeF+Su6tg0k/
eQnek1t8Du5uqkyBHVxy8MvxsBIMZ73vIFwUrLHjDeq3+n00yQqSBMnrHKU2mAwN3zb2Lo1wjPkKN0h1+NNnv99L2pBcNCn
+BgewyNdWrxYkKp403ZDa4f+5SVWvbY5eyDDXcohvz/
OcCtuRjAkzKBCvIjBDn Cv1McyjVdC03+utizGntfhAo1RZstn0oRkgVF2WuMT9IrUmzYximuTXUmWtjyFSTqgNBZwHWUTLm
csC4HPTKr3dazdvEkhwGAAAIFFByCjSp/5WHc4AhsyjMvKCsZQiI8ECwjfXBaSZdY4zYsRl03FC428H1atrFChFCZT0Bq
+vAUJiP8XqiEdXe qf2mYMJ5ykoDpwkve/cUQfPpjzFQLQfvwjBwiJDANKk0KoNT3bUGz+/f/pyTE
+xMRdIUBZ1Bw=="
    }
  ],
  "DeviceProfileId": "0ff5b0c6-f149-4498-af34-21993acd52a7",
  "PrivateKeys": [
    {
      "SigningAlg": "Ed25519",

      "Value": "2c24d4572327f23b9bef38097137c29224a9e979081b3d90124ac9dfa477934e"
    },
    {
      "SigningAlg": "P256r1",

      "Value": "38d526f29cfaf142f596deca187bd809ef71bc13435eedc885b63bb825d63def"
    }
  ],
  "SidewalkManufacturingSn": "843764270F4BDAE3023918C89A3307AB3351EA761887A40A9DC4A5E46B6140D9",
  "Status": "PROVISIONED"
},
...
}

```

Next steps

Store the JSON files, *wireless_device.json* and *device_profile.json* temporarily, as you'll use them in the next step to provision and register your end device for connecting to the hardware

platform. For more information, see [Provisioning and registering your end device](#) in the *Amazon Sidewalk documentation*.

Add a destination for your Sidewalk end device

Use AWS IoT rules to process the data and device messages and route it to other services. You can also define rules to process the binary messages received from a device and convert the messages to other formats that make other services easy to use them. Destinations associate your Sidewalk end device with the rule that processes the device data to send to other AWS services.

How to create and use a destination

1. Create an AWS IoT rule and an IAM role for the destination. The AWS IoT rule specifies the rules that will process the device's data and routes it for use by other AWS services and your applications. The IAM role grants permission to access the rule.
2. Create a destination for your Sidewalk devices using the `CreateDestination` API operation. Specify the destination name, rule name, role name, and any optional parameters. The API will return a unique identifier for the destination, which you can specify when adding your end device to AWS IoT Core for Amazon Sidewalk.

The following shows how to create a destination, and an AWS IoT rule and IAM role for the destination.

Topics

- [Create a destination for your Sidewalk device](#)
- [Create an IAM role and IoT rule for your destination](#)

Create a destination for your Sidewalk device

You can add a destination to your account for AWS IoT Core for Amazon Sidewalk either from the using the [Destinations hub](#) or using the `CreateDestination`. When creating your destination, specify:

- A unique name for the destination to use for your Sidewalk end device.

Note

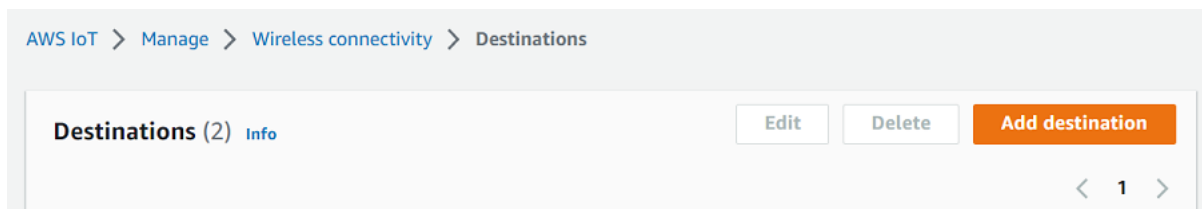
If you already add your device using a destination name, you must use that name when creating your destination. For more information, see [Step 2: Add your Sidewalk device](#).

- The name of the AWS IoT rule that will process the device's data, and the topic to which messages are published.
- An IAM role that grants the device's data permission to access the rule.

The following sections describe how to create the AWS IoT rule and IAM role for your destination.

Create a destination (console)

To create a destination using the AWS IoT console, go to the [Destinations hub](#) and choose **Add destination**.



To process a device's data, specify the following fields when creating a destination, and then choose **Add destination**.

- **Destination details**

Enter a **Destination name** and an optional description for your destination.

- **Rule name**

The AWS IoT rule that is configured to evaluate messages sent by your device and process the device's data. The rule name will be mapped to your destination. The destination requires the rule to process the messages that it receives. You can choose for the messages to be processed by either invoking an AWS IoT rule or by publishing to the AWS IoT message broker.

- If you choose **Enter a rule name**, enter a name, and then choose **Copy** to copy the rule name that you'll enter when creating the AWS IoT rule. You can either choose **Create rule** to create the rule now or navigate to the [Rules](#) Hub of the AWS IoT console and create a rule with that name.

You can also enter a rule and use the **Advanced** setting to specify a topic name. The topic name is provided during rule invocation and is accessed by using the topic expression inside the rule. For more information about AWS IoT rules, see [AWS IoT rules](#).

- If you choose **Publish to AWS IoT message broker**, enter a topic name. You can then copy the MQTT topic name and multiple subscribers can subscribe to this topic to receive messages published to that topic. For more information, see [MQTT topics](#).

For more information about AWS IoT rules for destinations, see [Create rules to process LoRaWAN device messages](#).

- **Role name**

The IAM role that grants the device's data permission to access the rule named in **Rule name**. In the console, you can create a new service role or select an existing service role. If you're creating a new service role, you can either enter a role name (for example, **SidewalkDestinationRole**), or leave it blank for AWS IoT Core for LoRaWAN to generate a new role name. AWS IoT Core for LoRaWAN will then automatically create the IAM role with the appropriate permissions on your behalf.

Create a destination (CLI)

To create a destination, use the [CreateDestination](#) API operation or the [create-destination](#) CLI command. For example, the following command creates a destination for your Sidewalk end device:

```
aws iotwireless create-destination --name SidewalkDestination \  
  --expression-type RuleName --expression SidewalkRule \  
  --role-arn arn:aws:iam::123456789012:role/SidewalkRole
```

Running this command returns the destination details, which include the Amazon Resource Name (ARN) and the destination name.

```
{  
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:Destination/SidewalkDestination",  
  "Name": "SidewalkDestination"  
}
```

For more information about creating a destination, see [Create rules to process LoRaWAN device messages](#).

Create an IAM role and IoT rule for your destination

AWS IoT rules send device messages to other services. AWS IoT rules can also process the binary messages received from a Sidewalk end device for other services to use. AWS IoT Core for Amazon Sidewalk destinations 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 Amazon Sidewalk receives it. For all devices that send their data to the same service, you can create a destination that can be shared by all devices. You must also create an IAM role that grants permission to send data to the rule.

Create an IAM role for your destination

Create an IAM role that grants AWS IoT Core for Amazon Sidewalk permission to send data to the AWS IoT rule. To create the role, use the [CreateRole](#) API operation or [create-role](#) CLI command. You can name the role as *SidewalkRole*.

```
aws iam create-role --role-name SidewalkRole \  
  --assume-role-policy-document '{"Version": "2012-10-17", "Statement":  
  [{ "Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action":  
  "sts:AssumeRole"}]}'
```

You can also define the trust policy for the role using a JSON file.

```
aws iam create-role --role-name SidewalkRole \  
  --assume-role-policy-document file://trust-policy.json
```

The following shows the contents of the JSON file.

Contents of trust-policy.json

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

```
    },
    "Action": "sts:AssumeRole"
  }
]
}
```

Create a rule for your destination

Use the AWS IoT Core API operation, [CreateTopicRule](#), or the AWS CLI command, [create-topic-rule](#), to create a rule. The topic rule will be used by your destination to route the data received from your Sidewalk end device to other AWS services. For example, you can create a rule action that sends a message to a Lambda function. You can define the Lambda function such that it receives the application data from your device and uses base64 to decode the payload data so that it can be used by other applications.

The following steps show how you create the Lambda function and then a topic rule that sends a message to this function.

1. Create execution role and policy

Create an IAM role that grants your function permission to access AWS resources. You can also define the trust policy for the role using a JSON file.

```
aws iam create-role --role-name lambda-ex \  
  --assume-role-policy-document file://lambda-trust-policy.json
```

The following shows the contents of the JSON file.

Contents of `lambda-trust-policy.json`

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

```
}

```

2. Create and test Lambda function

Perform the following steps to create a AWS Lambda function that base64 decodes the payload data.

- a. Write the code for decoding the payload data. For example, you can use the following sample Python code. Specify a name for the script, such as *base64_decode.py*.

Contents of *base64_decode.py*

```
// -----
// ----- Python script to decode incoming binary payload -----
// -----
import json
import base64

def lambda_handler(event, context):

    message = json.dumps(event)
    print (message)

    payload_data = base64.b64decode(event["PayloadData"])
    print(payload_data)
    print(int(payload_data,16))

```

- b. Create a deployment package as a zip file that contains the Python file and name it as *base64_decode.zip*. Use the CreateFunction API or the create-function CLI command to create a Lambda function for the sample code, *base64_decode.py*.

- c.


```
aws lambda create-function --function-name my-function \
--zip-file fileb://base64_decode.zip --handler index.handler \
--runtime python3.9 --role arn:aws:iam::123456789012:role/lambda-ex
```

You should see the following output. You'll use the Amazon Resource Name (ARN) value from the output, `FunctionArn`, when creating the topic rule.

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",

```

```

"Runtime": "python3.9",
"Role": "arn:aws:iam::123456789012:role/lambda-ex",
"Handler": "index.handler",
"CodeSha256": "FpFMvUhayLk0oVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
"Version": "$LATEST",
"TracingConfig": {
  "Mode": "PassThrough"
},
"RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
...
}

```

- d. To get logs for an invocation from the command line, use the `--log-type` option with the `invoke` command. The response includes a `LogResult` field that contains up to 4 KB of base64-encoded logs from the invocation.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

You should receive a response with a `StatusCode` of 200. For more information about creating and using Lambda functions from the AWS CLI, see [Using Lambda with the AWS CLI](#).

3. Create a topic rule

Use the `CreateTopicRule` API or the `create-topic-rule` CLI command to create a topic rule that sends a message to this Lambda function. You can also add a second rule action that republishes to an AWS IoT topic. Name this topic rule as *Sidewalkrule*.

```
aws iot create-topic-rule --rule-name Sidewalkrule \
  --topic-rule-payload file://myrule.json
```

You can use the `myrule.json` file to specify more details about the rule. For example, the following JSON file shows how to republish to an AWS IoT topic and send a message to a Lambda function.

```

{
  "sql": "SELECT * ",
  "actions": [
    {
      // You obtained this functionArn when creating the Lambda function
      using the

```



```
// create-function command.
"lambda": {
  "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-
function"
},
{
  // This topic can be used to observe messages exchanged between the
  device and
  // AWS IoT Core for Amazon Sidewalk after the device is connected.
  "republiSh": {
    "roleArn": "arn:aws:iam::123456789012:role/service-
role/SidewalkRepubliShRole",
    "topic": "project/sensor/observed"
  }
},
],
}
```

Connect your Sidewalk device and view uplink metadata format

In this tutorial, you'll use the MQTT test client to test the connectivity and see messages exchanged between your end device and the AWS Cloud. To receive messages, in the MQTT test client, subscribe to the topic specified when creating the IoT rule for your destination. You can also send a downlink message from AWS IoT Core for Amazon Sidewalk to your device using the `SendDataToWirelessDevice` API operation. You can verify that the message was delivered by enabling the message delivery status event notification.

Note

For information about connecting your hardware platform and setting it up, see [Provisioning and registering your end device](#) and [Setting up the hardware development kit \(HDK\)](#) in the *Amazon Sidewalk documentation*..

Send downlink messages to your end device

Use the [SendDataToWirelessDevice](#) API operation or the [send-data-to-wireless-device](#) CLI command to send downlink messages from AWS IoT Core for Amazon Sidewalk to your

Sidewalk end device. Following shows an example of how to run this command. The payload data is the binary to be sent, encoded in base64.

```
aws iotwireless send-data-to-wireless-device \  
  --id "<Wireless_Device_ID>" \  
  --payload-data "SGVsbG8gVG8gRGV2c2lt" \  
  --wireless-metadata Sidewalk={Seq=1,AckModeRetryDurationSecs=10}
```

Following shows a sample output of running this command, which is an ID of the downlink message sent to the device.

```
{  
  MessageId: "6011dd36-0043d6eb-0072-0008"  
}
```

Note

The `SendDataToWirelessDevice` API can return a message ID but the message might not be successfully delivered. To check the status of the message that was sent to the device, you can enable message delivery status events for your Sidewalk accounts and devices. For information about how to enable this event, see [Event notifications for Sidewalk resources](#). For more information about this event type, see [Message delivery events](#).

View format of uplink messages from the device

After you've connected your device, you can subscribe to the topic (for example, `project/sensor/observed`) that you specified when creating the destination rule, and observe uplink messages from the device.

If you specified a topic name when creating your destination, you can subscribe to the topic to monitor uplink messages from your end device. Go to the [MQTT test client](#) on the **Test** page of the AWS IoT console, enter the topic name (for example, `project/sensor/observed`), and then choose **Subscribe**.

The following example shows the format of the uplink messages that are sent from Sidewalk devices to AWS IoT. The `WirelessMetadata` contains metadata about the message request.

```
{
  "PayloadData": "ZjRlNjY1ZWw==",
  "WirelessDeviceId": "wireless_device_id",
  "WirelessMetadata": {
    "Sidewalk": {
      "CmdExStatus": "Cmd",
      "SidewalkId": "device_id",
      "Seq": 0,
      "MessageType": "messageType"
    }
  }
}
```

The following table shows a definition of the different parameters in the uplink metadata. The *device-id* is the ID of the wireless device, such as *ABCDEF1234* and the *messageType* is the type of uplink message that's received from the device.

Sidewalk uplink metadata parameters

Parameter	Description	Type	Required
PayloadData	The message payload that is sent from the wireless device.	String	Yes
WirelessDeviceID	The identifier of the wireless device that's sending the data	String	Yes
Sidewalk.CmdExStatus	Command runtime status. Response-type messages shall include the status code, <code>COMMAND_EXEC_STATUS_SUCCESS</code> . However, notifications might not include the status code.	Enumeration	No
Sidewalk.NackExStatus	Response nack status, which can be <code>RADIO_TX_ERROR</code> or <code>MEMORY_ERROR</code> .	Array of strings	No

Bulk provisioning devices with AWS IoT Core for Amazon Sidewalk

You can use bulk provisioning to onboard a large number of end devices to AWS IoT Core for Amazon Sidewalk in bulk. Bulk provisioning is useful especially when you manufacture a large number of devices in a factory and want to onboard these devices to AWS IoT. For more information about manufacturing devices, see [Manufacturing Amazon Sidewalk devices](#) in the *Amazon Sidewalk documentation*.

The following topics show you how bulk provisioning works.

- [Amazon Sidewalk bulk provisioning workflow](#)

This topic shows you some key concepts of bulk provisioning and how it works. It also shows the steps that must be performed so that your Sidewalk devices can be imported to AWS IoT Core for Amazon Sidewalk.

- [Creating device profiles with factory support](#)

This topic explains how to create a device profile and obtain factory support for it. You'll also learn how to retrieve the YubiHSM key and send it to your manufacturer for obtaining the control log after the devices are manufactured.

- [Provisioning Sidewalk devices using import tasks](#)

This topic shows you how to bulk provision your Sidewalk devices by creating and using import tasks. You'll also learn how to update or delete your import tasks, and view the status of the import task and devices in the task.

Topics

- [Amazon Sidewalk bulk provisioning workflow](#)
- [Creating device profiles with factory support](#)
- [Provisioning Sidewalk devices using import tasks](#)

Amazon Sidewalk bulk provisioning workflow

The following sections show you key concepts of bulk provisioning and how it works. The steps that are involved in bulk provisioning include:

1. Create a device profile using AWS IoT Core for Amazon Sidewalk.
2. Request the Amazon Sidewalk team for a YubiHSM key and to update your device profile with factory support.
3. Send the YubiHSM key to your manufacturer so that AWS IoT Core for Amazon Sidewalk can obtain the control log after the devices are manufactured.
4. Create an import task and provide the serial numbers (SMSN) of the devices to be onboarded to AWS IoT Core for Amazon Sidewalk.

Components of bulk provisioning

The following concepts show you some key components of bulk provisioning and how to use them as part of bulk provisioning your Sidewalk devices.

YubiHSM key

Amazon creates one or more HSMs (hardware security modules) for each of your Sidewalk products. Each HSM has a unique serial number, called YubiHSM key, that's printed on the hardware module. This key can be purchased from the [Yubico webpage](#).

The key is unique to each HSM and tied to each device profile that you create with AWS IoT Core for Amazon Sidewalk. To obtain the YubiHSM key, contact the Amazon Sidewalk team. If you send the YubiHSM key to the manufacturer, after the Sidewalk devices are manufactured in the factory, AWS IoT Core for Amazon Sidewalk will receive a control log file that contains the serial numbers of the devices. It then compares this information with your input CSV file for onboarding the devices to AWS IoT.

Device attestation key (DAK)

When a Sidewalk end device joins the Sidewalk network, it must be provisioned with a Sidewalk device certificate. The certificates that are used for setting up your device include a private device-specific certificate, and the public device certificates, which correspond to the Sidewalk certificate chain. When your Sidewalk devices are manufactured, the YubiHSM signs the device certificates.

The following shows a sample JSON file that contains the device certificates and the private keys. For more information, see [Obtain device JSON files for provisioning](#).

```
{  
  "p256R1": "grg8izXoVvQ86cPVm0GMyWuZYHEBbbH ... DANKk0KoNT3bUGz+/f/pyTE  
+xMRdIUBZ1Bw==",
```

```
"eD25519": "grg8izXoVvQ86cPvm0GMyWuZYHEBbbHD ... UiZmntHiUr1GfkTOFMYqRB+Aw==",
"metadata": {
  "devicetypeid": "fe98",

  ...

  "devicePrivKeyP256R1":
  "3e704bf8d319b3a475179f1d68c60737b28c708f845d0198f2d00d00c88ee018",
  "devicePrivKeyEd25519":
  "17dacb3a46ad9a42d5c520ca5f47f0167f59ce54d740aa13918465faf533b8d0"
},
"applicationServerPublicKey":
"5ce29b89c2e3ce6183b41e75fe54e45f61b8bb320efbdd2abd7aefa5957a316b"
}
```

The device attestation key (DAK) is a private key that you obtain when creating your device profile. It corresponds to the product certificate, which is a unique certificate that's issued to each Sidewalk product. When you contact the Amazon Sidewalk team, you'll receive the Sidewalk certificate chain, the YubiHSM key, and an HSM provisioned with the product device attestation key (DAK).

Your device profile is also updated with the new device attestation key (DAK), and with factory support enabled. The DAK metadata information of the device profile provides details such as the DAK name, the certificate ID, the ApId (Advertised Product ID), whether factory support is enabled, and the maximum number of signatures that the DAK can sign.

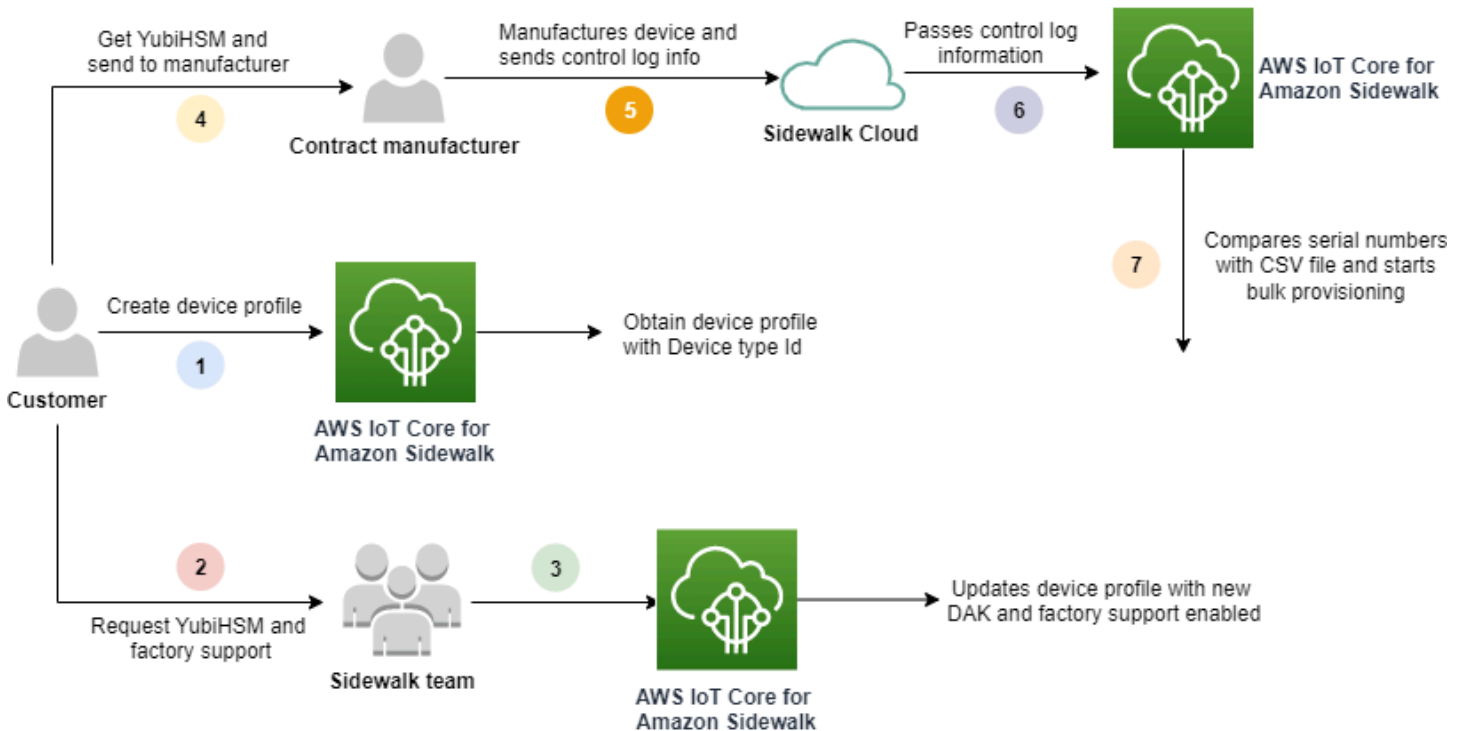
Advertised product ID (ApId)

The ApId parameter is an alphanumeric string that identifies the advertised product. This field must be specified when you want to use a given device profile for Sidewalk devices that you bulk provision. AWS IoT Core for Amazon Sidewalk then generates the DAK, and provides it to you through the YubiHSM key. The related DAK information will be presented in the device profile.

To obtain the ApId, after you retrieve information about the device profile that you created, contact the Amazon Sidewalk Support team. You can obtain the device profile information from the AWS IoT console, or using the [GetDeviceProfile](#) API operation, or the [get-device-profile](#) CLI command.

How bulk provisioning works

This flowchart shows how bulk provision works with AWS IoT Core for Amazon Sidewalk.



The following procedure illustrates the different steps in the bulk provisioning process.

1. Create device profile for Sidewalk device

Before you take your end device to the factory, first create a device profile. You can use this profile to provision individual devices as described in [Add your device profile and Sidewalk end device](#).

2. Request factory support for your profile

When you're ready to take your end device to factory, ask the Amazon Sidewalk team for the YubiHSM key and for factory support for your device profile.

3. Obtain DAK and factory supported profile

The Amazon Sidewalk Support team will then update your device profile with the product device attestation key (DAK) and factory support. Your device profile will be updated automatically with an advertised product ID (ApID), and a new DAK and certificate information, such as the certificate ID. Sidewalk devices that use this profile are qualified for use with bulk provisioning.

4. Send YubiHSM key to manufacturer (CM)

Your end device is now qualified, so you can send your YubiHSM key to the contract manufacturer (CM) to start the manufacturing process. For more information, see [Manufacturing Amazon Sidewalk devices](#) in the *Amazon Sidewalk documentation*.

5. Manufacture devices and send control logs and serial numbers

The CM manufactures the devices and generates control logs. The CM also provides you a CSV file that contains a list of devices to be manufactured and their Sidewalk manufacturing serial numbers (SMSN). The following code shows a sample control log. It contains the serial numbers of the device, the APID, and the public device certificates.

```
{
  "controlLogs": [
    {
      "version": "4-0-1",
      "device": {
        "serialNumber": "device1",
        "productIdentifier": {
          "advertisedProductId": "abCD"
        },
        "sidewalkData": {
          "SidewalkED25519CertificateChain": "...",
          "SidewalkP256R1CertificateChain": "..."
        }
      }
    }
  ]
}
```

6. Pass control log information to AWS IoT Core for Amazon Sidewalk

The Amazon Sidewalk cloud retrieves the control log information from the manufacturer and passes this information to AWS IoT Core for Amazon Sidewalk. The devices can then be created along with their serial numbers.

7. Check serial number match and start bulk provisioning

When you use the AWS IoT console or the AWS IoT Core for Amazon Sidewalk API operation `StartWirelessDeviceImportTask`, AWS IoT Core for Amazon Sidewalk compares the Sidewalk manufacturing serial number (SMSN) of each devices obtained from Amazon

Sidewalk with the corresponding serial numbers in your CSV file. If this information matches, it starts the bulk provisioning process and creates the devices to be imported to AWS IoT Core for Amazon Sidewalk.

Creating device profiles with factory support

Before you can bulk provision your Amazon Sidewalk devices, you must create a device profile and then contact the Amazon Sidewalk support team to request factory support for it. The Amazon Sidewalk team will then update your device profile with a new device attestation key (DAK) and add factory support to it. Sidewalk devices that use this profile are then qualified to be used with AWS IoT Core for Amazon Sidewalk, and can be onboarded for bulk provisioning.

The following steps show you how to create a factory-supported device profile.

1. Create a device profile

First create a device profile. When you create a profile, specify a name and optional tags as name-value pairs. For more information about the parameters required, and creating and using profiles, see [How to create and add your device](#).

2. Obtain factory support for the profile

Then obtain factory support for your device profile so that devices that use this profile can be qualified. For qualification, create a ticket with the Amazon Sidewalk team. Once confirmed by the team, you'll receive an Aprod (advertised product ID), and your profile will be updated with a factory-issued DAK. Sidewalk end devices that use this profile will be qualified.

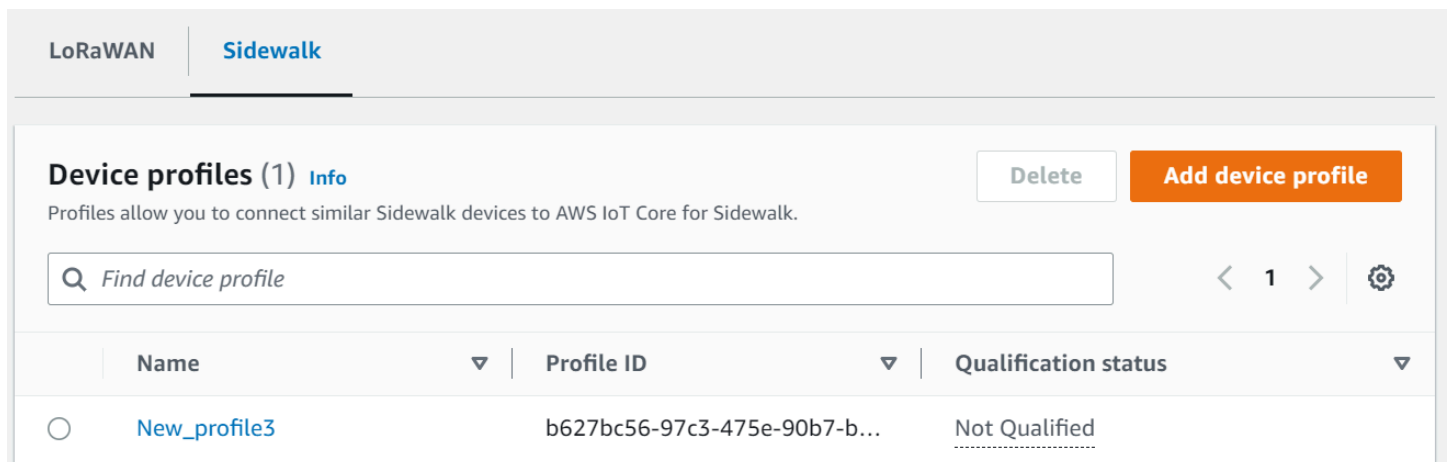
You can create a device profile either using the AWS IoT console, the AWS IoT Core for Amazon Sidewalk API operations, or the AWS CLI.

Topics

- [Create a profile \(console\)](#)
- [Create a profile \(CLI\)](#)
- [Next steps](#)

Create a profile (console)

To create a device profile using the AWS IoT console, go to the [Sidewalk tab of the Profiles hub](#) and choose **Create profile**.



The screenshot shows the AWS IoT console interface for the Profiles hub. At the top, there are two tabs: 'LoRaWAN' and 'Sidewalk', with 'Sidewalk' selected. Below the tabs, there is a header section for 'Device profiles (1)' with an 'Info' link. To the right of the header are 'Delete' and 'Add device profile' buttons. Below the header is a search bar with the placeholder text 'Find device profile'. Below the search bar is a table with the following columns: 'Name', 'Profile ID', and 'Qualification status'. The table contains one row with the following data: 'New_profile3', 'b627bc56-97c3-475e-90b7-b...', and 'Not Qualified'.

Name	Profile ID	Qualification status
New_profile3	b627bc56-97c3-475e-90b7-b...	Not Qualified

To create a profile, specify the following fields, and then choose **Submit**.

- **Name**

Enter a **Name** for your profile.

- **Tags**

Enter optional tags as name-value pairs to help you more easily identify your profile. Tags also make it easier to track billing charges.

View profile information and qualify profiles

You'll see the profile that you created in the [Profiles hub](#). Choose the profile to view its details. You'll see information about:

- The device profile name and unique identifier, and any optional tags you specified as name value pairs.
- The application server public key and device type ID of the profile.
- The qualification status, which indicates that you're using a device profile that is not factory supported. To qualify your device profile so that it's factory supported, contact Amazon Sidewalk Support.
- The device attestation key (DAK) information. Once your device profile is qualified, a new DAK will be issued and your profile will be updated automatically with the new DAK information.

Create a profile (CLI)

To create a device profile, use the [CreateDeviceProfile](#) API operation or the [create-device-profile](#) CLI command. For example, the following command creates a profile for your Sidewalk end device.

```
aws iotwireless create-device-profile \  
  --name sidewalk_device_profile --sidewalk {}
```

Running this command returns the profile details, which include the Amazon Resource Name (ARN) and the ID of the profile.

```
{  
  "DeviceProfileArn": "arn:aws:iotwireless:us-  
east-1:123456789012:DeviceProfile/12345678-a1b2-3c45-67d8-e90fa1b2c34d",  
  "DeviceProfileId": "12345678-a1b2-3c45-67d8-e90fa1b2c34d"  
}
```

View profile information and qualify profiles

Use the [GetDeviceProfile](#) API operation or the [get-device-profile](#) CLI command to get information about your device profile that you added to your account for AWS IoT Core for Amazon Sidewalk. To retrieve information about your device profile, specify the profile ID. The API will then return information about the device profile matching the specified identifier.

The following shows an example CLI command:

```
aws iotwireless get-device-profile \  
  --id "12345678-234a-45bc-67de-e8901234f0a1" > device_profile.json
```

Running this command returns the parameters of your device profile, the application server public key, the DeviceTypeId, ApId, qualification status, and the DAKCertificate information.

In this example, the qualification status and DAK information indicate that your device profile is not qualified. To qualify your profile, contact Amazon Sidewalk Support, and your profile will be issued a new DAK with no device limit.

```
{
```

```

    "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
    "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
    "Name": "Sidewalk_profile",
    "LoRaWAN": null,
    "Sidewalk":
    {
        "ApplicationServerPublicKey":
        "a123b45c6d78e9f012a34cd5e6a7890b12c3d45e6f78a1b234c56d7e890a1234",
        "DAKCertificateMetadata": [
            {
                "DeviceTypeId": "fe98",
                "CertificateId": "43564A6D2D50524F544F54595045",
                "FactorySupport": false,
                "MaxAllowedSignature": 1000
            }
        ],
        "QualificationStatus": false
    }
}

```

Once the Amazon Sidewalk Support team confirms this information, you'll receive the APID and a factory-supported DAK, as shown in the following example.

Note

The MaxAllowedSignature of -1 indicates that the DAK doesn't have any device limit. For information about the DAK parameters, see [DAKCertificateMetadata](#).

```

{
    "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d",
    "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
    "Name": "Sidewalk_profile",
    "LoRaWAN": null,
    "Sidewalk":
    {
        "ApplicationServerPublicKey":
        "a123b45c6d78e9f012a34cd5e6a7890b12c3d45e6f78a1b234c56d7e890a1234",
        "DAKCertificateMetadata": [
            {

```

```
        "ApId": "GZBd",
        "CertificateId": "43564A6D2D50524F544F54595045",
        "FactorySupport": true,
        "MaxAllowedSignature": -1
    }
],
"QualificationStatus": true
}
```

Next steps

Now that you've created a device profile which has a factory-supported DAK, provide the YubiHSM key that you obtain from the team to your manufacturer. Your devices will then be manufactured in the factory and a control log information will then be passed to Amazon Sidewalk, which contains the serial numbers (SMSN) of the devices. For more information about this workflow, see [Manufacturing Amazon Sidewalk devices](#) in the *Amazon Sidewalk documentation*.

You can then bulk provision your Sidewalk devices by providing AWS IoT Core for Amazon Sidewalk the serial numbers of the devices to be onboarded. When AWS IoT Core for Amazon Sidewalk receives the control log, it compares the serial numbers in the control log with the serial numbers that you provided. If the serial numbers match, the import task starts onboarding your devices to AWS IoT Core for Amazon Sidewalk. For more information, see [Provisioning Sidewalk devices using import tasks](#).

Provisioning Sidewalk devices using import tasks

This section shows how you can provision Sidewalk devices in bulk using the AWS IoT console, or the AWS IoT Core for Amazon Sidewalk API operations, or the AWS CLI. The following sections explain how to bulk provision your Sidewalk devices.

Topics

- [How Sidewalk bulk provisioning works](#)
- [Key considerations for Sidewalk bulk provisioning](#)
- [CSV file format](#)
- [How to use Sidewalk bulk provisioning](#)
- [Provision Sidewalk devices in bulk](#)
- [View import task and device onboarding status](#)

How Sidewalk bulk provisioning works

The following steps illustrate how bulk provisioning works.

1. Starting wireless device import task

To provision Sidewalk devices in bulk, you must create an import task and provide the Sidewalk manufacturing serial number (SMSN) of the devices to be onboarded to AWS IoT Core for Amazon Sidewalk. You obtained the Sidewalk manufacturing serial number (SMSN) of the devices as a CSV file in your email after the manufacturer uploaded the control logs to Amazon Sidewalk. For more information about the workflow and how you obtain the control log, see [Manufacturing Amazon Sidewalk devices](#) in the *Amazon Sidewalk documentation*.

2. Running import process in background

When AWS IoT Core for Amazon Sidewalk receives the import task request, it starts setting things up and starts a background process that polls the system frequently. Once the background process receives the import task instruction, it starts reading the CSV file. AWS IoT Core for Amazon Sidewalk simultaneously checks whether the control logs have been received from Amazon Sidewalk.

3. Creating wireless device records

When the control log is received from Amazon Sidewalk, AWS IoT Core for Amazon Sidewalk checks whether the serial numbers in the control log match the SMSN values in the CSV file. If the serial numbers match, AWS IoT Core for Amazon Sidewalk will start creating wireless device records for the Sidewalk devices that correspond to these serial numbers. Once all the devices have been onboarded, the import task is marked as *Completed*.

Key considerations for Sidewalk bulk provisioning

When provisioning your Sidewalk devices in bulk to AWS IoT Core for Amazon Sidewalk, following are some key considerations.

- You must perform bulk provisioning using the AWS IoT console or the AWS IoT Core for Amazon Sidewalk API operations in the same AWS account where you created the device profile.
- Before you bulk provision your Sidewalk devices, your device profile must already contain DAK information that indicates factory support. Otherwise, bulk provisioning using the AWS IoT console, or the bulk provisioning API operations can fail.

- After you start an import task, it can take at least 10 minutes or longer to process the CSV file, import the wireless devices, and onboard them to AWS IoT Core for Amazon Sidewalk.
- The wireless device import task will run for 90 days, once started. During this time, it checks whether the control logs have been received from Amazon Sidewalk. If the control log is not received from Amazon Sidewalk before 90 days, the task will be marked as *Completed* with a message indicating that it has expired when you view the task details. The onboarding status of the devices in the import task that were waiting for the control log will be marked as *Failed*.
- When you attempt to update an import task that you've already created, you can only add additional devices to the task. You can add new devices anytime after creating an import task and before the task has started on devices that were already added to the import task. If the update file contains serial numbers of devices that already exist in the original import task, these serial numbers will be ignored.
- When you request an update operation, the same IAM role as the one you used when creating the import task will be assumed to access the CSV file in the Amazon S3 bucket.
- An import task can be deleted only if the task has already completed successfully or if the task failed to update. A task might fail to update in cases such as when an incorrect IAM role was provided or when an Amazon S3 bucket file wasn't found. An import task cannot be updated or deleted if it is in the PENDING state.
- The CSV file that you import to the task must use the format described in the following section.

CSV file format

The CSV file that's contained in an Amazon S3 bucket that you specify for the import task must use the following format:

- Row 1 must use the keyword `smsn`, which indicates that the CSV file being imported contains the SMSN of the devices to be imported.
- Rows 2 and after must contain the SMSN of the devices to be onboarded. The device SMSN must be in the 64 hex character format.

This JSON file shows a sample CSV file format.

```
smsn
1C1A10B0AC0A200C012BBAC2CBB1B21CB12C0CA2AC1C1BB22CAA01C1B0B01122
B122C2B1121BACA2221001AC1B22012AAC11112C11C2A100C1C2B012A1100C10
02B222C110B0A210B0A0C2C112CCCAC21C1C0B0AA1221AB1022A2CC11B1B1122
```

```
C2C021CA1C111CCAB1221C0021C1C2AAA0AA1A2A01ABC10CBAACCA2A0121022A
0CB22C01BBC2CA2C0B11001121ACB2ABB0BB0121C2BA101C012CC2B20C011AC0
```

How to use Sidewalk bulk provisioning

The following steps show you how to use Amazon Sidewalk bulk provisioning.

1. Provide device serial numbers

To provision your Sidewalk devices, you must provide the serial numbers of the devices to be onboarded. You can provision your devices using either of the following methods.

- Provision each device individually using their Sidewalk manufacturing serial number (SMSN). This method is useful when you want to test the workflow and onboard your device faster without having to upload a CSV file with the appropriate IAM role, or waiting for the devices to be ready to be onboarded to the task.
- Provision devices in bulk by providing an Amazon S3 bucket URL that contains the SMSN of the devices to be provisioned in a CSV file. This method is especially useful when you have a large number of devices to be onboarded. In this case, onboarding each device individually can be tedious. Instead, you only need to provide the path to the CSV file that has been uploaded to an Amazon S3 bucket, and the IAM role to access the file.

2. Obtain import task and device onboarding status

For each import task that you create, you can retrieve information about the task onboarding status and the onboarding status of devices added to the task. You can also see additional status information, such as a reason as to why a task or device onboarding failed. For more information, see

3. (Optional) Update or delete import task

You can update or delete an import task that you've already created.

- You can update an import task and add additional devices to the task anytime before the task has started on devices that have already been added. AWS IoT Core for Amazon Sidewalk assumes the same IAM role as the one that you used when creating the import task. When you create the task, specify the new CSV file that contains the serial numbers of devices that you want to add to the task.

Note

When you're updating an existing import task, you can only add devices to the task. AWS IoT Core for Amazon Sidewalk performs a union operation between the devices that are already in the import task and the devices that you're attempting to add to the task. If the new file contains serial numbers of devices that already exist in the import task, these serial numbers will be ignored.

- You can delete an import task that has already completed successfully, or an import task that failed to update in cases such as when the IAM role information is incorrect, or when an S3 bucket file isn't available when creating or updating a task.

Topics

- [Provision Sidewalk devices in bulk](#)
- [View import task and device onboarding status](#)

Provision Sidewalk devices in bulk


This section shows how you can provision Sidewalk devices in bulk to AWS IoT Core for Amazon Sidewalk using the AWS IoT console and the AWS CLI.

Provision Sidewalk devices in bulk (console)

To add your Sidewalk device using the AWS IoT console, go to the [Sidewalk tab of the Devices hub](#), choose **Bulk provision devices**, and then perform the following steps.


LoRaWAN
Sidewalk

▼ How it works
 With AWS IoT Core for Sidewalk, you can add your Sidewalk device fleet to the AWS Cloud. Use the following steps to get started.




Step 1. Add your Sidewalk device

First, create a device profile and retrieve the application server public key. Next, create your Sidewalk device and retrieve information about it, including device certificates and private keys.



Step 2. Provision & register your Sidewalk device

Provision your hardware as a Sidewalk endpoint by flashing the device certificates and the application server public key that you have generated. Register your device so that it can connect to AWS IoT Core for Amazon Sidewalk.



Step 3. Connect your Sidewalk endpoint to the cloud

Create a destination and use [AWS IoT Rules](#) to process and route data to other AWS services. Your endpoint can now exchange messages with your cloud application.

Bulk provision (0) [Info](#)

Bulk provisioning table shows the task IDs, which includes tasks that are added for individual devices, and tasks that are linked with your [S3 CSV files](#).

Bulk provision devices

< 1 >
⚙️

Task ID	Creation date	S3 bucket	Success count	Pending count	Failed count
No bulk provisioning tasks are currently running at this time.					

1. Choose import method

Specify how you want to import the devices to be onboarded in bulk to AWS IoT Core for Amazon Sidewalk.

- To provision individual devices using their SMSN, choose **Provision individual factory supported device**.
- To provision devices in bulk by providing a CSV file that contains a list of devices and their SMS, choose **Use S3 bucket**.

2. Specify devices to be onboarded

Depending on the method that you chose to onboard your devices, add the device information and their serial numbers.

- a. If you chose **Provision individual factory supported device**, specify the following information:

- i. A **Name** for each device to be onboarded. The name must be unique in your AWS account and AWS Region.
 - ii. Their Sidewalk manufacturing serial number (SMSN) in the **Enter SMSN** field.
 - iii. A **Destination** that describes the IoT rule to route messages from the device to other AWS services.
- b. If you chose **Use S3 bucket**:
- i. Provide the **S3 Bucket destination** information, which consists of the S3 URL information. To provide your CSV file, choose **Browse S3**, and then choose the CSV file you want to use.

AWS IoT Core for Amazon Sidewalk automatically populates the S3 URL, which is the path to your CSV file in the S3 bucket. The format of the path is `s3://bucket_name/file_name`. To view the file in the [Amazon Simple Storage Service](#) console, choose **View**.

- ii. Provide the **S3 Provisioning role**, which allows AWS IoT Core for Amazon Sidewalk to access the CSV file in the S3 bucket on your behalf. You can either create a new service role or choose an existing role.

To create a new role, you can either provide a **Role name** or leave it blank to generate a random name automatically.

- iii. Provide a **Destination** that describes the IoT rule to route messages from the device to other AWS services.

3. Start import task

Provide any optional tags as name-value pairs and choose **Submit** to start your wireless device import task.

Provision Sidewalk devices in bulk (CLI)

To onboard your Sidewalk devices to your account for AWS IoT Core for Amazon Sidewalk, use any of the following API operations depending on whether you want to add devices individually or by providing the CSV file contained in an S3 bucket.

- **Upload devices in bulk using an S3 CSV file**

To upload devices in bulk by providing the CSV file in an S3 bucket, use the [StartWirelessDeviceImportTask](#) API operation, or the [start-wireless-device-import-task](#) AWS CLI command. When creating the task, specify the path to the CSV file in the Amazon S3 bucket and the IAM role that grants AWS IoT Core for Amazon Sidewalk permissions to access the CSV file.

Once the task starts to run, AWS IoT Core for Amazon Sidewalk will start reading the CSV file and compare the serial numbers (SMSN) in the file with the corresponding information in the control log received from Amazon Sidewalk. When the serial numbers match, it will start creating wireless device records corresponding to these serial numbers.

The following command shows an example of creating an import task:

```
aws iotwireless start-wireless-device-import-task \  
  --cli-input-json "file://task.json"
```

The following shows the contents of the file `task.json`.

Contents of `task.json`

```
{  
  "DestinationName": "Sidewalk_Destination",  
  "Sidewalk": {  
    "DeviceCreationFile": "s3://import_task_bucket/import_file1",  
    "Role": "arn:aws:iam::123456789012:role/service-role/ACF1zBEI"  
  }  
}
```

Running this command returns an ID and ARN for the import task.

```
{  
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ImportTask/a1b234c5-67ef-21a2-  
a1b2-3cd4e5f6789a"  
  "Id": "a1b234c5-67ef-21a2-a1b2-3cd4e5f6789a"  
}
```

- **Provision devices individually using their SMSN**

To provision devices individually using their SMSN, use the [StartSingleWirelessDeviceImportTask](#) API operation, or the [start-single-wireless-device-import-task](#) AWS CLI command. When creating the task, specify the Sidewalk destination and the serial number of the device that you want to onboard.

When the serial number matches the corresponding information in the control log received from Amazon Sidewalk, the task will run and create the wireless device record.

The following command shows an example of creating an import task:

```
aws iotwireless start-single-wireless-device-import-task \
  --destination-name sidewalk_destination \
  --sidewalk
  '{"SidewalkManufacturingSn": "82B83C8B35E856F43CE9C3D59B418CC96B996071016DB1C3BE5901F0F3071A"
```

Running this command returns an ID and ARN for the import task.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:ImportTask/e2a5995e-743b-41f2-a1e4-3ca6a5c5249f"
  "Id": "e2a5995e-743b-41f2-a1e4-3ca6a5c5249f"
}
```

Update or delete import tasks

If you want to add additional devices to an import task, you can update the task. You can also delete a task if you no longer require the task or if it failed. For information about when to update or delete a task, see [How to use Sidewalk bulk provisioning](#).

Warning

Deletion actions are permanent and can't be undone. Deleting an import task that has already completed successfully will not remove the end devices that have already been onboarded using the task.

To update or delete import tasks:

- **Using the AWS IoT console**

The following steps explain how to update or delete your import tasks using the AWS IoT console.

To update an import task:

1. Go to the [Sidewalk devices hub](#) of the AWS IoT console.
2. Choose the import task that you want to update and then choose **Edit**.
3. Provide another S3 file that contains the serial numbers of devices that you want to add to the task and then choose **Submit**.

To delete an import task:

1. Go to the [Sidewalk devices hub](#) of the AWS IoT console.
2. Choose the task that you want to delete and then choose **Delete**.

- **Using the AWS IoT Wireless API or AWS CLI**

Use the following AWS IoT Wireless API operations or CLI commands to update or delete your import task.

- [UpdateWirelessDeviceImportTask](#) API or [update-wireless-device-import-task](#) CLI

This API operation appends the contents of an Amazon S3 CSV file to an existing import task. You can only add serial numbers of devices that were not previously included in the task.

- [DeleteWirelessDeviceImportTask](#) API or [delete-wireless-device-import-task](#) CLI

This API operation deletes the import task that was marked for deletion using the import task ID.

View import task and device onboarding status

Your wireless device import tasks and Sidewalk devices that you've added to the task can have one of the following status messages. You'll see these messages displayed in the AWS IoT console, or when you use any of the AWS IoT Wireless API operations or AWS CLI commands to retrieve information about these tasks and their devices.

View import task status information

After you've created an import task, you can view the import task that you created and the onboarding status of devices added to the task. The onboarding status indicates the number of devices that are pending onboarding, the number of devices that have been onboarded successfully, and the number of devices that failed to onboard.

When an import task has just been created, the **Pending count** will display a value that corresponds to the number of devices that were added. Once the task starts and reads the CSV file to create the wireless device records, the **Pending count** will decrease, and the **Success count** will increase as the devices are onboarded successfully. If any device fails to onboard, the **Failed count** will increase.

To view the import task and device onboarding status:

- **Using the AWS IoT console**

In the [Sidewalk devices hub](#) of the AWS IoT console, you can see the import tasks that you created and a count of the summary of the onboarding status information of your devices. If you view the details of any of the import tasks that you created, you can see additional information about the device onboarding status.

- **Using the AWS IoT Wireless API or AWS CLI**

To view the device onboarding status, use any of the following AWS IoT Wireless API operations or the corresponding AWS CLI command. .

- [ListWirelessDeviceImportTasks](#) API or [list-wireless-device-import-tasks](#) CLI

This API operation returns information about all the import tasks that have been added to your account for AWS IoT Wireless and their status. It also returns a count of the summary of onboarding status of Sidewalk devices in these tasks.

- [ListDevicesForWirelessDeviceImportTask](#) API or [list-devices-for-wireless-device-import-task](#) CLI

This API operation returns information about the specified import task and its status, and information about all Sidewalk devices that have been added to the import task and their onboarding status information.

- [GetWirelessDeviceImportTask](#) API or [get-wireless-device-import-task](#) CLI

This API operation returns information about the specified import task and its status, and a count of the summary of onboarding status of Sidewalk devices in that task.

Import task status

The import tasks that you have created in your AWS account can have one of the following status messages. The status indicates whether your import task has started processing, or has completed, or failed. You can also use the AWS IoT console or the `StatusReason` parameter of any of the AWS IoT Wireless API operations to retrieve additional status details.

- **INITIALIZING**

AWS IoT Core for Amazon Sidewalk has received the wireless device import task request and is in the process of setting up the task.

- **INITIALIZED**

AWS IoT Core for Amazon Sidewalk has completed setting up the import task and is waiting for the control log to arrive so that it can import the devices using their serial numbers (SMSN) and continue processing the task.

- **PENDING**

The import task is waiting in the queue to be processed. AWS IoT Core for Amazon Sidewalk is evaluating other tasks that are in the processing queue.

- **COMPLETE**

The import task has been processed and completed.

- **FAILED**

The import task or device task failed. You can use the `StatusReason` parameter to identify why the import task failed, such as a validation exception.

- **DELETING**

The import task has been marked for deletion and is in the process of being deleted.

Device onboarding status

The Sidewalk devices that you added to your import task can have one of the following status messages. The status indicates whether your devices are ready to be onboarded, or has onboarded, or failed to onboard. You can also use the AWS IoT console or the `OnboardingStatusReason` parameter of the AWS IoT Wireless API operation, `ListDevicesForWirelessDeviceImportTask`, to retrieve additional status details.

- **INITIALIZED**

AWS IoT Core for Amazon Sidewalk has completed setting up the import task and is waiting for the control log to arrive so that it can import the devices using their serial numbers (SMSN) and continue processing the task.

- **PENDING**

The import task is waiting in the queue to be processed and to start onboarding your devices to the task. AWS IoT Core for Amazon Sidewalk is evaluating other tasks that are in the processing queue.

- **ONBOARDED**

The Sidewalk device has been successfully onboarded to the import task.

- **FAILED**

The import task or device task failed, and the Sidewalk device failed to onboard to the task. You can use the `OnboardingStatusReason` parameter to retrieve additional details about why the device onboarding failed.

Security in AWS IoT Wireless

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 Wireless, 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 Wireless. It shows you how to configure AWS IoT Wireless 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 Wireless resources.

Contents

- [Data protection in AWS IoT Wireless](#)
- [Identity and access management for AWS IoT Wireless](#)
- [Compliance validation for AWS IoT Wireless](#)
- [Resilience in AWS IoT Wireless](#)
- [Infrastructure security in AWS IoT Wireless](#)

Data protection in AWS IoT Wireless

The AWS [shared responsibility model](#) applies to data protection in AWS IoT Wireless. 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. You are also responsible for 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 users with AWS IAM Identity Center or 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 require TLS 1.2 and recommend TLS 1.3.
- 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 sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 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 \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT Wireless or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption in AWS IoT Wireless

By default, all AWS IoT Wireless data in transit and at rest is encrypted. AWS IoT Wireless doesn't support customer-managed AWS KMS keys from AWS KMS key. To encrypt the data, AWS IoT Wireless only uses an AWS owned key.

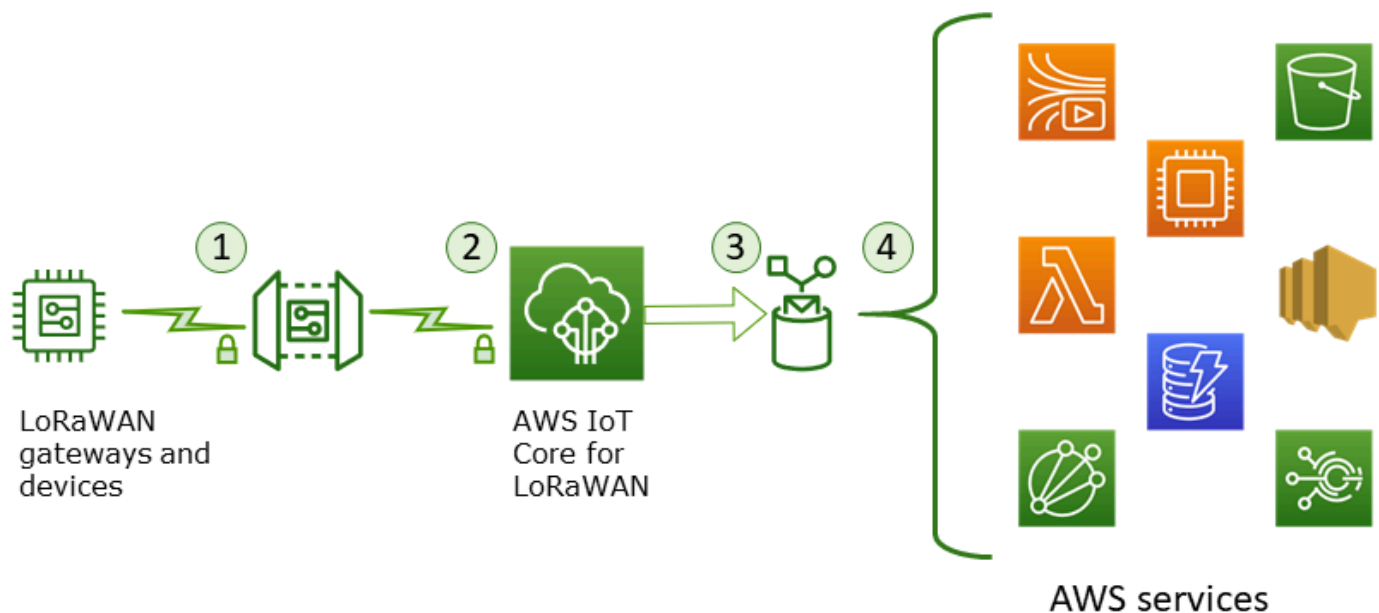
Data and transport security with AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN uses the following methods to secure the data and communication between LoRaWAN devices, gateways, and AWS IoT Core for LoRaWAN:

- The security best practices that devices follow when communicating with LoRaWAN gateways, as described in the whitepaper [LoRaWAN Security](#).
- The security that AWS IoT Core uses to connect gateways to AWS IoT Core for LoRaWAN and send the data to other AWS services. For more information, see [data protection in AWS IoT Core](#).

How data is secured throughout the system

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](#). 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](#) to authenticate the gateway. To ensure the confidentiality of the LoRaWAN protocol, [TLS version 1.2](#) is used. 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.

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.

Identity and access management for AWS IoT Wireless

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 Wireless resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS IoT Wireless works with IAM](#)
- [AWS IoT Wireless identity-based policy examples](#)
- [AWS managed policies for AWS IoT Wireless](#)
- [Troubleshooting AWS IoT Wireless identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS IoT Wireless.

Service user – If you use the AWS IoT Wireless service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT Wireless 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 Wireless, see [Troubleshooting AWS IoT Wireless identity and access](#).

Service administrator – If you're in charge of AWS IoT Wireless resources at your company, you probably have full access to AWS IoT Wireless. It's your job to determine which AWS IoT Wireless features and resources your service users 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 Wireless, see [How AWS IoT Wireless works with IAM](#).

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 Wireless. To view example AWS IoT Wireless identity-based policies that you can use in IAM, see [AWS IoT Wireless identity-based policy examples](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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 [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

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

Note

AWS IoT Wireless doesn't support service roles and service-linked 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:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **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 [Cross account resource access in IAM](#) 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.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. 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 AWS 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 AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. 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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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 an 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 [Service control policies](#) 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 Wireless works with IAM

Before you use IAM to manage access to AWS IoT Wireless, you should understand what IAM features are available to use with AWS IoT Wireless. To get a high-level view of how AWS IoT Wireless and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

IAM features you can use with AWS IoT Wireless

IAM feature	AWS IoT Wireless support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	No

Topics

- [AWS IoT Wireless Identity-based policies](#)

- [Resource-based policies within AWS IoT Wireless](#)
- [Policy actions](#)
- [Policy resources](#)
- [Condition keys](#)
- [Access control lists \(ACLs\)](#)
- [ABAC with AWS IoT Wireless](#)
- [Using temporary credentials with AWS IoT Wireless](#)
- [Cross-service principal permissions for AWS IoT Wireless](#)
- [Service roles](#)
- [Service-linked roles for AWS IoT Wireless](#)

AWS IoT Wireless Identity-based policies

Supports identity-based policies: Yes

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*.

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. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Examples

To view examples of AWS IoT Wireless identity-based policies, see [AWS IoT Wireless identity-based policy examples](#).

Resource-based policies within AWS IoT Wireless

Supports resource-based policies: No

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.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions

Supports policy actions: Yes

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.

Policy actions in AWS IoT Wireless use the following prefix before the action: `iotwireless:`. For example, to grant someone permission to list all wireless devices registered in their AWS account with the `ListWirelessDevices` API operation, you include the `iotwireless:ListWirelessDevices` action in their policy. Policy statements must include either an `Action` or `NotAction` element. AWS IoT Wireless 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": [
```

```
"iotwireless:ListMulticastGroups",  
"iotwireless:ListFuotaTasks"  
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Get, include the following action:

```
"Action": "iotwireless:Get*"
```

To see a list of AWS IoT Wireless actions, see [Actions Defined by AWS IoT Wireless](#) in the *IAM User Guide*.

Policy resources

Supports policy resources: Yes

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": "*"
```

The AWS IoT Wireless service has the following ARN:

```
arn:${Partition}:iotwireless:${Region}:${Account}:${Resource}/${Resource-id}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the network analyzer configuration, NAConfig1, in your statement, use the following ARN:

```
"Resource": "arn:aws:iotwireless:us-east-1:123456789012:NetworkAnalyzerConfiguration/  
NAConfig1"
```

To specify all FUOTA tasks that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:iotwireless:us-east-1:123456789012:FuotaTask/*"
```

Some AWS IoT Wireless actions, such as those for listing resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

Many AWS IoT Wireless API actions involve multiple resources. For example, `AssociateWirelessDeviceWithThing` associates a wireless device with an AWS IoT thing, so an IAM user must have permissions to use the device and an IoT thing. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "WirelessDevice",  
    "thing"
```

To see a list of AWS IoT Wireless resource types and their ARNs, see [Resources Defined by AWS IoT Wireless](#) in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS IoT Wireless](#).

Condition keys

Supports service-specific policy condition keys: Yes

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 Wireless 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*. To see a list of AWS IoT Wireless condition keys, see [Condition Keys for AWS IoT Wireless](#) in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see [Actions Defined by AWS IoT Wireless](#)

Access control lists (ACLs)

Supports ACLs: No

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.

ABAC with AWS IoT Wireless

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

You can attach tags to AWS IoT Wireless resources or pass tags in a request to AWS IoT Wireless. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `YOUR-SERVICE-PREFIX:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging AWS IoT Wireless resources, see [Tagging your AWS IoT Wireless resources](#).

Using temporary credentials with AWS IoT Wireless

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for AWS IoT Wireless

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with

the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles

Supports service roles: No

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. 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 roles for AWS IoT Wireless

Supports service-linked roles: No

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 AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

AWS IoT Wireless identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS IoT Wireless 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](#)
- [Using the AWS IoT Wireless console](#)
- [Allow users to view their own permissions](#)
- [Permissions required to perform AWS IoT Wireless wireless device actions](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS IoT Wireless 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the AWS IoT Wireless console

To access the AWS IoT Wireless console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT Wireless 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 Wireless console, also attach the following AWS managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*:

```
AWSIoTWirelessFullAccess
```

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": "*"
}
]
}

```

Permissions required to perform AWS IoT Wireless wireless device actions

You can use conditions in your identity-based policy to control access to AWS IoT Wireless actions. This example shows how you might create a policy that allows creating and managing devices. 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": "VisualEditor0",
    "Effect": "Allow",
    "Action": [
        "iotwireless:CreateWirelessDevice",
        "iotwireless:GetWirelessDevice",
        "iotwireless:ListWirelessDevices",
        "iotwireless:UpdateWirelessDevice",
        "iotwireless>DeleteWirelessDevice"
    ],
    "Resource": "*"
  }
]
}

```

The policy has one statement that grants permission to use the `CreateWirelessDevice`, `GetWirelessDevice`, `ListWirelessDevices`, `UpdateWirelessDevice`, and

`DeleteWirelessDevice` actions. AWS IoT Wireless calls these methods to create and manage your wireless devices.

The policy doesn't specify the Principal element because you don't specify the principal who gets the permission in an identity-based policy. When you attach a policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

AWS managed policies for AWS IoT Wireless

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

AWS managed policy: `AWSIoTWirelessDataAccess`

You can attach the `AWSIoTWirelessDataAccess` policy to your IAM identities.

This policy grants the associated identity permissions that allow access to send data to LoRaWAN and Sidewalk devices using the `SendDataToWirelessDevice` API. To view this policy in the AWS Management Console, see [AWSIoTWirelessDataAccess](#).

Permissions details

This policy includes the following permissions.

- `iotwireless` – Retrieve AWS IoT Wireless data.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotwireless:SendDataToWirelessDevice"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: `AWSIoTWirelessFullAccess`

You can attach the `AWSIoTWirelessFullAccess` policy to your IAM identities.

This policy grants the associated identity permissions that allow full access to all AWS IoT Wireless operations. To view this policy in the AWS Management Console, see [AWSIoTWirelessFullAccess](#).

Permissions details

This policy includes the following permissions.

- `iotwireless` – Retrieve AWS IoT Wireless data and perform all AWS IoT Wireless operations.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotwireless:*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: `AWSIoTWirelessFullPublishAccess`

You can attach the `AWSIoTWirelessFullPublishAccess` policy to your IAM identities.

This policy grants the associated identity permissions that allow limited access to publish to AWS IoT rules on your behalf. To view this policy in the AWS Management Console, see [AWSIoTWirelessFullPublishAccess](#).

Permissions details

This policy includes the following permissions.

- `iot` – Perform operations that gets the endpoint URL and publishes to AWS IoT rules engine.

```
{
```

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

AWS managed policy: AWSIoTWirelessLogging

You can attach the `AWSIoTWirelessLogging` policy to your IAM identities.

This policy grants the associated identity permissions that allow creation of Amazon CloudWatch Logs log groups and stream logs to the groups. This policy is attached to your CloudWatch logging role. To view this policy in the AWS Management Console, see [AWSIoTWirelessLogging](#).

Permissions details

This policy includes the following permissions.

- `logs` – Retrieve CloudWatch logs. Also allows creation of CloudWatch Logs groups and stream logs to the groups.

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

AWS managed policy: AWSIoTWirelessReadOnlyAccess

You can attach the AWSIoTLogging policy to your IAM identities.

This policy grants the associated identity permissions that allow read-only access to AWS IoT Wireless operations. To view this policy in the AWS Management Console, see [AWSIoTWirelessReadOnlyAccess](#).

Permissions details

This policy includes the following permissions.

- logs – Perform AWS IoT Wireless List and Get API operations.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotwireless:List*",
        "iotwireless:Get*"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

AWS managed policy: AWSIoTWirelessGatewayCertManager

You can attach the `AWSIoTWirelessGatewayCertManager` policy to your IAM identities.

This policy grants the associated identity permissions that allow access to create, list, and describe AWS IoT certificates. To view this policy in the AWS Management Console, see [AWSIoTWirelessGatewayCertManager](#).

Permissions details

This policy includes the following permissions.

- `iot` – Perform actions that create, describe, and list certificates.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "IoTWirelessGatewayCertManager",
      "Effect": "Allow",
      "Action": [
        "iot:CreateKeysAndCertificate",
        "iot:DescribeCertificate",
        "iot:ListCertificates"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS IoT Wireless; updates to AWS managed policies

View details about updates to AWS managed policies for AWS IoT Wireless since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [AWS IoT Wireless Document history page](#).

Change	Description	Date
AWS IoT Wireless started tracking changes	AWS IoT Wireless started tracking changes for its AWS managed policies.	May 18, 2022

Troubleshooting AWS IoT Wireless identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT Wireless and IAM.

Topics

- [I Am Not Authorized to Perform an Action in AWS IoT Wireless](#)
- [I Want to View My Access Keys](#)
- [I'm an Administrator and Want to Allow Others to Access AWS IoT Wireless](#)
- [I Want to Allow People Outside of My AWS Account to Access My AWS IoT Wireless Resources](#)

I Am Not Authorized to Perform an Action in AWS IoT Wireless

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 `WirelessDevice` but does not have `YOUR-SERVICE-PREFIX:GetWirelessDevice` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: YOUR-SERVICE-PREFIX:GetWirelessDevice on resource: my-LoRaWAN-device
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-LoRaWAN-device` resource using the `YOUR-SERVICE-PREFIX:GetWirelessDevice` 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 AWS 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 Wireless

To allow others to access AWS IoT Wireless, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in AWS IoT Wireless. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see [IAM Identities](#) and [Policies and permissions in IAM](#) in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My AWS IoT Wireless 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 Wireless supports these features, see [How AWS IoT Wireless works with IAM](#).
- 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 [Cross account resource access in IAM](#) in the *IAM User Guide*.

Compliance validation for AWS IoT Wireless

Third-party auditors assess the security and compliance of AWS IoT Wireless 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 Wireless 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.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – AWS Config; 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 Wireless

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between 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](#).

Infrastructure security in AWS IoT Wireless

As a managed service, AWS IoT Wireless 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 Wireless through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend 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.

Additionally, requests must be signed 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.

Monitoring your AWS IoT Wireless resources using Amazon CloudWatch Logs

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT Wireless and your other AWS solutions. You can use monitoring for both your LoRaWAN and Sidewalk devices and obtain informative messages and errors from when they are onboarded to AWS IoT Wireless.

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 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 and establish a baseline of normal AWS IoT Wireless performance in your environment by measuring performance at various times and under different load conditions. As you monitor AWS IoT Wireless, 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.

Monitoring tools

You can use the following monitoring tools to watch AWS IoT Wireless, report when something is wrong, and take automatic actions when appropriate:

- Amazon CloudWatch monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify.

For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).

- Network analyzer enables you to monitor your LoRaWAN resources, which includes LoRaWAN devices and gateways. reduces the time that it takes to set up a connection to start receiving trace messages, providing you with just-in-time log information. For more information, see [Monitoring your wireless resource fleet in real time using network analyzer](#).

How to monitor resources using Amazon CloudWatch

You can monitor AWS IoT Wireless using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

To log and monitor your AWS IoT Wireless resources, perform the following steps:

1. Create a logging role to log your AWS IoT Wireless resources, as described in [Create logging role and policy for AWS IoT Wireless](#).
2. Log messages in the CloudWatch Logs console have a default log level of ERROR, which is less verbose and contains only error information. If you want to view more verbose messages, we recommend that you use the CLI to configure logging first, as described in [Configure logging for AWS IoT Wireless resources](#).
3. Next, you can monitor your resources by viewing the log entries in the CloudWatch Logs console. For more information, see [View CloudWatch AWS IoT Wireless log entries](#).
4. You can create filter expressions by using **Logs groups** but we recommend that you first create simple filters and view log entries in the log groups, and then go to CloudWatch Insights to create queries to filter the log entries depending on the resource or event you're monitoring. For more information, see [Use CloudWatch Insights to filter logs for AWS IoT Wireless](#).

Configure Logging for AWS IoT Wireless

Before you can monitor and log AWS IoT activity, first enable logging for AWS IoT Wireless resources by using either the CLI or API.

When considering how to configure your AWS IoT Wireless logging, the default logging configuration determines how AWS IoT activity will be logged unless you specify otherwise. Starting out, you might want to obtain detailed logs with a default log level of INFO.

After reviewing the initial logs, you can change the default log level to ERROR, which is less verbose, and set a more verbose, resource-specific log level on resources that might need more attention. Log levels can be changed whenever you want.

The following topics show how to configure logging for AWS IoT Wireless resources.

Topics

- [Create logging role and policy for AWS IoT Wireless](#)
- [Configure logging for AWS IoT Wireless resources](#)

Create logging role and policy for AWS IoT Wireless

The following shows how to create a logging role for only AWS IoT Wireless resources. If you want to also create a logging role for AWS IoT Core, see <https://docs.aws.amazon.com/iot/latest/developerguide/create-logging-role.html>.

Create a logging role for AWS IoT Wireless

Before you can enable logging, you must create an IAM role and a policy that gives AWS permission to monitor AWS IoT Wireless activity on your behalf.

Create IAM role for logging

To create a logging role for AWS IoT Wireless, open the [Roles hub of the IAM console](#) and choose **Create role**.

1. Under **Select type of trusted entity**, choose **Another AWS account**.
2. In **Account ID**, enter your AWS account ID, and then choose **Next: Permissions**.
3. In the search box, enter **AWSIoTWirelessLogging**.
4. Select the box next to the policy named **AWSIoTWirelessLogging**, and then choose **Next: Tags**.
5. Choose **Next: Review**.
6. In **Role name**, enter **IoTWirelessLogsRole**, and then choose **Create role**.

Edit trust relationship of the IAM role

In the confirmation message displayed after you ran the previous step, choose the name of the role you created, **IoTWirelessLogsRole**. Next, you'll edit the role to add the following trust relationship.

1. In the **Summary** section of the role **IoTWirelessLogsRole**, choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
2. 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": {}
    }
  ]
}
```

3. To save your changes and exit, choose **Update Trust Policy**.

Logging policy for AWS IoT Wireless

The following policy document provides the role policy and trust policy that allows AWS IoT Wireless to submit log entries to CloudWatch on your behalf.

Note

This AWS managed policy document was automatically created for you when you created the logging role, **IoTWirelessLogsRole**.

Role policy

The following shows the role policy document.

```
{
  "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 Wireless activity

The following shows the trust policy for logging only AWS IoT Wireless activity.

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

```
        "Action": "sts:AssumeRole"
    }
]
}
```

If you created the IAM role to also log AWS IoT Core activity, then the policy documents allow you to log both activities. For information about creating a logging role for AWS IoT Core, see <https://docs.aws.amazon.com/iot/latest/developerguide/create-logging-role.html>.

Next steps

You've learned how to create a logging role to log your AWS IoT Wireless resources. By default, logs have a log level of ERROR, so if you want to see only error information, go to [View CloudWatch AWS IoT Wireless log entries](#) to monitor your wireless resources by viewing the log entries.

If you want more information in the log entries, you can configure the default log level for your resources or for different event types, such as setting the log level to INFO. For information about configuring logging for your resources, see [Configure logging for AWS IoT Wireless resources](#).

Configure logging for AWS IoT Wireless resources

To configure logging for AWS IoT Wireless resources, you can use either the API or the CLI. When starting to monitor AWS IoT Wireless resources, you can use the default configuration. To do this, you can skip this topic and proceed to [Monitor AWS IoT Wireless using CloudWatch Logs](#) to monitor your logs.

After you start monitoring the logs, you can use the CLI to change the log levels to a more verbose option, such as providing INFO and ERROR information and enabling logging for more resources.

AWS IoT Wireless resources and log levels

Before you use the API or CLI, use the following table to learn about the different log levels and the resources that you can configure logging for. The table shows parameters that you see in the CloudWatch logs when you monitor the resources. How you configure the logging for your resources will determine the logs you see in the console.

For information about what a sample CloudWatch logs looks like and how you can use these parameters to log useful information about the AWS IoT Wireless resources, see [View CloudWatch AWS IoT Wireless log entries](#).

Log levels and resources

Name	Possible values	Description
logLevel	INFO, ERROR, or DISABLED	<ul style="list-style-type: none"> ERROR: Displays any error that causes an operation to fail. Logs include only ERROR information. INFO: Provides high-level information about the flow of things. Logs include INFO and ERROR information. DISABLED: Disables all logging.
resource	WirelessGateway or WirelessDevice	The type of the resource, which can be WirelessGateway or WirelessDevice .
wirelessGatewayType	LoRaWAN	The type of the wireless gateway, when resource is WirelessGateway , which is always LoRaWAN.
wirelessDeviceType	LoRaWAN or Sidewalk	The type of the wireless device, when resource is WirelessDevice , which can be LoRaWAN or Sidewalk.
wirelessGatewayId	-	The identifier of the wireless gateway, when resource is WirelessGateway .
wirelessDeviceId	-	The identifier of the wireless device, when resource is WirelessDevice .
event	Join, Rejoin, Registration , Uplink_data , Downlink_data , CUPS_Request , and Certificate	The type of event being logged, which depends on whether the resource that you're logging is a wireless device or a wireless gateway. For more information, see View CloudWatch AWS IoT Wireless log entries .

AWS IoT Wireless logging API

You can use the following API actions to configure logging of resources. The table also shows a sample IAM policy that you must create for using the API actions. The following section describes how you can use the APIs to configure log levels of your resources.

Logging API actions

API name	Description	Sample IAM policy
GetLogLevelsByResourceTypes	Returns current default log levels, or log levels by resource types, which can include log options for wireless devices or wireless gateways.	<pre>{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["iotwireless:GetLogLevelsByResourceTypes"], "Resource": ["*"] }] }</pre>
GetResourceLogLevel	Returns the log-level override for a given resource identifier and resource type. The resource can be a wireless device or a wireless gateway.	<pre>{ "Version": "2012-10-17", "Statement": [{</pre>

API name	Description	Sample IAM policy
		<pre> "Effect": "Allow", "Action": ["iotwireless:GetResourceLogLevel"], "Resource": ["arn:aws:iotwireless:us-east-1:123456789012:WirelessDevice/012bc537-ab12-cd3a-d00e-1f0e20c1204a",] }] }</pre>

API name	Description	Sample IAM policy
PutResourceLogLevel	<p>Sets the log-level override for a given resource identifier and resource type. The resource can be a wireless gateway or a wireless device.</p> <div data-bbox="529 493 1029 760" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>This API has a limit of 200 log-level overrides per account.</p> </div>	<pre data-bbox="1068 226 1507 1411"> { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["iotwireless:PutResourceLogLevel"], "Resource": ["arn:aws:iotwireless:us-east-1:123456789012:WirelessDevice/012bc537-ab12-cd3a-d00e-1f0e20c1204a",] }] }</pre>

API name	Description	Sample IAM policy
ResetAllResourceLogLevels	<p>Removes the log-level overrides for all resources, which includes both wireless gateways and wireless devices.</p> <div data-bbox="529 447 1029 810" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; background-color: #e6f2ff;"> <p>Note</p> <p>This API doesn't affect the log levels that are set using the UpdateLogLevelsByResourceTypes API.</p> </div>	<pre data-bbox="1068 226 1507 1528"> { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["iotwireless:Reset AllResourceLogLevels"], "Resource": ["arn:aws:iotwireless:us-east-1:123456789012:WirelessDevice/*", "arn:aws:iotwireless:us-east-1:123456789012:WirelessGateway/*"] }] }</pre>

API name	Description	Sample IAM policy
ResetResourceLogLevel	Removes the log-level override for a given resource identifier and resource type. The resource can be a wireless gateway or a wireless device.	<pre>{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["iotwireless:Reset ResourceLogLevel"], "Resource": ["arn:aws:iotwirele ss:us-east-1:12345 6789012:WirelessDe vice/012bc537-ab12 -cd3a-d00e-1f0e20c 1204a",] }] } }</pre>

API name	Description	Sample IAM policy
UpdateLogLevelsByResourceTypes	<p>Set default log level, or log levels by resource types. You can use this API for log options for wireless devices or wireless gateways, and control the log messages that'll be displayed in CloudWatch.</p> <div data-bbox="529 541 1029 905" style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>Events are optional and the event type is tied to the resource type. For more information, see Events and resource types.</p> </div>	<pre data-bbox="1068 226 1505 1213"> { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["iotwireless:UpdateLogLevelsByResourceTypes"], "Resource": ["*"] }] }</pre>

Configure log levels of resources using the CLI

This section describes how to configure log levels for AWS IoT Wireless resources by using the API or AWS CLI.

Before you use the CLI:

- Make sure you created the IAM policy for the API for which you want to run the CLI command, as described previously.
- 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 logging role and policy for AWS IoT Wireless](#).

Why use the AWS CLI

By default, if you create the IAM role, `IoTWirelessLogsRole`, as described in [Create logging role and policy for AWS IoT Wireless](#), you'll see CloudWatch logs in the AWS Management Console that have a default log level of `ERROR`. To change the default log level for all your resources or for specific resources, use the AWS IoT Wireless logging API or CLI.

How to use the AWS CLI

The API actions can be categorized into the following types depending on whether you want to configure log levels for all resources or for specific resources:

- API actions `GetLogLevelsByResourceTypes` and `UpdateLogLevelsByResourceTypes` can retrieve and update the log levels for all resources in your account that are of a specific type, such as a wireless gateway, or a LoRaWAN or Sidewalk device.
- API actions `GetResourceLogLevel`, `PutResourceLogLevel`, and `ResetResourceLogLevel` can retrieve, update, and reset log levels of individual resources that you specify using a resource identifier.
- API action `ResetAllResourceLogLevels` resets the log-level override to `null` for all resources for which you specified a log-level override using the `PutResourceLogLevel` API.

To use the CLI to configure resource-specific logging for AWS IoT

Note

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.

1. By default, all resources have log level set to `ERROR`. To set the default log levels, or log levels by resource types for all resources in your account, use the [update-log-levels-by-resource-types](#) command. The following example shows how you can create a JSON file, `Input.json`, and provide it as an input to the CLI command. You can use this command to selectively disable logging or override the default log level for specific types of resources and events.

```
{
  "DefaultLogLevel": "INFO",
  "WirelessDeviceLogOptions":
  [
```

```
{
  "Type": "Sidewalk",
  "LogLevel": "INFO",
  "Events":
  [
    {
      "Event": "Registration",
      "LogLevel": "DISABLED"
    }
  ]
},
{
  "Type": "LoRaWAN",
  "LogLevel": "INFO",
  "Events":
  [
    {
      "Event": "Join",
      "LogLevel": "DISABLED"
    },
    {
      "Event": "Rejoin",
      "LogLevel": "ERROR"
    }
  ]
}
]
"WirelessGatewayLogOptions":
[
  {
    "Type": "LoRaWAN",
    "LogLevel": "INFO",
    "Events":
    [
      {
        "Event": "CUPS_Request",
        "LogLevel": "DISABLED"
      },
      {
        "Event": "Certificate",
        "LogLevel": "ERROR"
      }
    ]
  }
]
```



```
}  
  ]  
}
```

where:

WirelessDeviceLogOptions

The list of log options for a wireless device. Each log option includes the wireless device type (Sidewalk or LoRaWAN), and a list of wireless device event log options. Each wireless device event log option can optionally include the event type and its log level.

WirelessGatewayLogOptions

The list of log options for a wireless gateway. Each log option includes the wireless gateway type (LoRaWAN), and a list of wireless gateway event log options. Each wireless gateway event log option can optionally include the event type and its log level.

DefaultLogLevel

The log level to use for all your resources. Valid values are: ERROR, INFO, and DISABLED. The default value is INFO.

LogLevel

The log level you want to use for individual resource types and events. These log levels override the default log level, such as the log level INFO for the LoRaWAN gateway, and log levels DISABLED and ERROR for the two event types.

Run the following command to provide the `Input.json` file as input to the command. This command doesn't produce any output.


```
aws iotwireless update-log-levels-by-resource-types \  
  --cli-input-json Input.json
```

If you want to remove the log options for both wireless devices and wireless gateways, run the following command.

```
{  
  "DefaultLogLevel": "DISABLED",  
  "WirelessDeviceLogOptions": [],  
  "WirelessGatewayLogOptions": []  
}
```

```
}
```

2. The **update-log-levels-by-resource-types** command doesn't return any output. Use the [get-log-levels-by-resource-types](#) command to retrieve resource-specific logging information. The command returns the default log level, and the wireless device and wireless gateway log options.

 **Note**

The **get-log-levels-by-resource-types** command can't directly retrieve the log levels in the CloudWatch console. You can use the **get-log-levels-by-resource-types** command to get the latest log-level information that you've specified for your resources using the **update-log-levels-by-resource-types** command.

```
aws iotwireless get-log-levels-by-resource-types
```

When you run the following command, it returns the latest logging information you specified with **update-log-levels-by-resource-types**. For example, if you remove the wireless device log options, then running the **get-log-levels-by-resource-types** will return this value as `null`.

```
{
  "DefaultLogLevel": "INFO",
  "WirelessDeviceLogOptions": null,
  "WirelessGatewayLogOptions":
  [
    {
      "Type": "LoRaWAN",
      "LogLevel": "INFO",
      "Events":
      [
        {
          "Event": "CUPS_Request",
          "LogLevel": "DISABLED"
        },
        {
          "Event": "Certificate",
          "LogLevel": "ERROR"
        }
      ]
    }
  ]
}
```

```
    }  
  ]  
}
```

3. To control log levels for individual wireless gateways or wireless device resources, use the following CLI commands:

- [put-resource-log-level](#)
- [get-resource-log-level](#)
- [reset-resource-log-level](#)

For an example for when to use these CLIs, say that you have a large number of wireless devices or gateways in your account that are being logged. If you want to troubleshoot errors for only some of your wireless devices, you can disable logging for all wireless devices by setting the `DefaultLogLevel` to `DISABLED`, and use the **put-resource-log-level** to set the `LogLevel` to `ERROR` for only those devices in your account.

```
aws iotwireless put-resource-log-level \  
  --resource-identifier \  
  --resource-type WirelessDevice \  
  --log-level ERROR
```

In this example, the command sets the log level to `ERROR` only for the specified wireless device resource and the logs for all other resources are disabled. This command doesn't produce any output. To retrieve this information and verify that the log levels were set, use the **get-resource-log-level** command.

4. In the previous step, after you've debugged the issue and resolved the error, you can run the **reset-resource-log-level** command to reset the log level for that resource to `null`. If you used the `put-resource-log-level` command to set the log-level override for more than one wireless device or gateway resource, such as for troubleshooting errors for multiple devices, you can reset the log-level overrides back to `null` for all those resources using the [reset-all-resource-log-levels](#) command.

```
aws iotwireless reset-all-resource-log-levels
```

This command doesn't produce any output. To retrieve the logging information for the resources, run the **get-resource-log-level** command.

Next Steps

You've learned how to create the logging role and use the AWS IoT Wireless API to configure logging for your AWS IoT Core for LoRaWAN resources. Next, to learn about monitoring your log entries, go to [Monitor AWS IoT Wireless using CloudWatch Logs](#).

Monitor AWS IoT Wireless using CloudWatch Logs

AWS IoT Core for LoRaWAN has more than 50 CloudWatch log entries that are enabled by default. Each log entry describes the event type, log level, and the resource type. For more information, see [AWS IoT Wireless resources and log levels](#).

How to monitor your AWS IoT Wireless resources

When logging is enabled for AWS IoT Wireless, AWS IoT Wireless sends progress events about each message as it passes from your devices through AWS IoT and back. By default, AWS IoT Wireless log entries have a default log level of error. When you enable logging as described in [Create logging role and policy for AWS IoT Wireless](#), you'll see messages in the CloudWatch console that have a default log level of ERROR. By using this log level, the messages will show only error information for all wireless devices and gateway resources that you're using.

If you want the logs to show additional information, such as those that have a log level of INFO, or disable logs for some of your devices and show log messages for only some of your devices, you can use the AWS IoT Wireless logging API. For more information, see [Configure log levels of resources using the CLI](#).

You can also create filter expressions to display only the required messages.

Before you can view AWS IoT Wireless logs in the console

To make the `/aws/iotwireless` log group appear in the CloudWatch console, you must have done the following.

- Enabled logging in AWS IoT Wireless. For more information about how to enable logging in AWS IoT Wireless, see [Configure Logging for AWS IoT Wireless](#).
- Written some log entries by performing AWS IoT Wireless operations.

To create and use filter expressions more effectively, we recommend that you try using CloudWatch insights as described in the following topics. We also recommend that you follow the topics in the order they're presented here. This will help you use CloudWatch **Log groups** first to learn about the

different types of resources, its event types, and log levels that you can use to view log entries in the console. You can then learn how to create filter expressions by using CloudWatch Insights to get more helpful information from your resources.

Topics

- [View CloudWatch AWS IoT Wireless log entries](#)
- [Use CloudWatch Insights to filter logs for AWS IoT Wireless](#)

View CloudWatch AWS IoT Wireless log entries

After you've configured logging for AWS IoT Wireless as described in [Create logging role and policy for AWS IoT Wireless](#) and written some log entries, you can view the log entries in the CloudWatch console by performing the following steps.

Viewing AWS IoT logs in the CloudWatch Log groups console

In the [CloudWatch console](#), CloudWatch logs appear in a log group named `/aws/iotwireless`. For more information about CloudWatch Logs, see [CloudWatch Logs](#).

To view your AWS IoT logs in the CloudWatch console

Navigate to the [CloudWatch console](#) and choose **Log groups** in the navigation pane.

1. In the **Filter** text box, enter `/aws/iotwireless`, and then choose the `/aws/iotwireless` Log group.
2. To see a complete list of the AWS IoT Core for LoRaWAN logs generated for your account, choose **Search all**. To look at an individual log stream, choose the expand icon.
3. To filter the log streams, you can also enter a query in the **Filter events** text box. Here are some queries to try:

- `{ $.logLevel = "ERROR" }`

Use this filter to find all logs that have a log level of ERROR and you can expand the individual error streams to read the error messages, which will help you resolve them.

- `{ $.resource = "WirelessGateway" }`

Find all logs for the `WirelessGateway` resource regardless of the log level.

- `{ $.event = "CUPS_Request" && $.logLevel = "ERROR" }`

Find all logs that have an event type of CUPS_Request and a log level of ERROR.

Events and resource types

The following table shows the different types of events for which you'll see log entries. The event types also depend on whether the resource type is a wireless device or a wireless gateway. You can use the default log level for the resources and event types or override the default log level by specifying a log level for each of them.

Event types based on resources used

Resource	Resource type	Event type
Wireless gateway	LoRaWAN	<ul style="list-style-type: none"> CUPS_Request Certificate
Wireless device	LoRaWAN	<ul style="list-style-type: none"> Join Rejoin Uplink_Data Downlink_Data
Wireless device	Sidewalk	<ul style="list-style-type: none"> Registration Uplink_Data Downlink_Data

The following topic contains more information about these event types and the log entries for wireless gateways and wireless devices.

Topics

- [Log entries for wireless gateways and wireless device resources](#)

Log entries for wireless gateways and wireless device resources

After you've enabled logging, you can view log entries for your wireless gateways and wireless devices. The following section describes the various kinds of log entries based on your resource and event types.

Wireless gateway log entries

This section shows some of the sample log entries for your wireless gateway resources that you'll see in the [CloudWatch console](#). These log messages can have event type as CUPS_Request or Certificate, and can be configured to display a log level of INFO, ERROR, or DISABLED at the resource level or the event level. If you want to see only error information, set the log level to ERROR. The message in the ERROR log entry will contain information about why it failed.

The log entries for your wireless gateway resource can be classified based on the following event types:

- **CUPS_Request**

The LoRa Basics Station running on your gateway periodically sends a request to the Configuration and Update Server (CUPS) for updates. For this event type, if you set log level to INFO when configuring the CLI for your wireless gateway resource, then in the logs:

- If the event is successful, you'll see log messages that have a `LogLevel` of INFO. The messages will include details about the CUPS response sent to your gateway and the gateway details. Following shows an example of this log entry. For more information about the `LogLevel` and other fields in the log entry, see [AWS IoT Wireless resources and log levels](#).

```
{
  "timestamp": "2021-05-13T16:56:08.853Z",
  "resource": "WirelessGateway",
  "wirelessGatewayId": "5da85cc8-3361-4c79-8be3-3360fb87abda",
  "wirelessGatewayType": "LoRaWAN",
  "gatewayEui": "feffff00000000e2",
  "event": "CUPS_Request",
  "LogLevel": "INFO",
  "message": "Sending CUPS response of total length 3213 to GatewayEui:
feffff00000000e2 with TC Credentials,"
}
```

- If there is an error, you'll see log entries that have a `LogLevel` of ERROR, and the messages will include details about the error. Examples of when an error can occur for the CUPS_Request event include: missing CUPS CRC, mismatch in the gateway's TC Uri with AWS IoT Core for LoRaWAN, missing `IoTWirelessGatewayCertManagerRole`, or not able to obtain wireless gateway record. Following example shows a missing CRC log entry. To resolve the error, check your gateway setup to verify that you've entered the correct CUPS CRC.

```
{
  "timestamp": "2021-05-13T16:56:08.853Z",
  "resource": "WirelessGateway",
  "wirelessGatewayId": "5da85cc8-3361-4c79-8be3-3360fb87abda",
  "wirelessGatewayType": "LoRaWAN",
  "gatewayEui": "feffff00000000e2",
  "event": "CUPS_Request",
  "logLevel": "ERROR",
  "message": "The CUPS CRC is missing from the request. Check your gateway setup
and enter the CUPS CRC,"
}
```

• Certificate

These log entries will help you check whether your wireless gateway presented the correct certificate for authenticating connection to AWS IoT. For this event type, if you set log level to INFO when configuring the CLI for your wireless gateway resource, then in the logs:

- If the event is successful, you'll see log messages that have a `logLevel` of INFO. The messages will include details about the Certificate ID and the Wireless gateway identifier. Following shows an example of this log entry. For more information about the `logLevel` and other fields in the log entry, see [AWS IoT Wireless resources and log levels](#).

```
{
  "resource": "WirelessGateway",
  "wirelessGatewayId": "5da85cc8-3361-4c79-8be3-3360fb87abda",
  "wirelessGatewayType": "LoRaWAN",
  "event": "Certificate",
  "logLevel": "INFO",
  "message": "Gateway connection authenticated.
(CertificateId:
b5942a7aee973eda24314e416889227a5e0aa5ed87e6eb89239a83f515dea17c,
WirelessGatewayId: 5da85cc8-3361-4c79-8be3-3360fb87abda)"
}
```

- If there is an error, you'll see log entries that have a `logLevel` of ERROR, and the messages will include details about the error. Examples of when an error can occur for the Certificate event include an invalid Certificate ID, wireless gateway identifier, or a mismatch between the wireless gateway identifier and the Certificate ID. Following example shows an ERROR due to invalid wireless gateway identifier. To resolve the error, check the gateway identifiers.


```
{
  "resource": "WirelessGateway",
  "wirelessGatewayId": "5da85cc8-3361-4c79-8be3-3360fb87abda",
  "wirelessGatewayType": "LoRaWAN",
  "event": "Certificate",
  "logLevel": "INFO",
  "message": "The gateway connection couldn't be authenticated because a
  provisioned gateway associated with the certificate couldn't be found.
  (CertificateId:
  729828e264810f6fc7134daf68056e8fd848afc32bfe8082beeb44116d709d9e)"
}
```

Wireless device log entries

This section shows some of the sample log entries for your wireless device resources that you'll see in the [CloudWatch console](#). The event type for these log messages depend on whether you're using a LoRaWAN or a Sidewalk device. Each wireless device resource or event type can be configured to display a log level of INFO, ERROR, or DISABLED.

Note

Your request must not contain both LoRaWAN and Sidewalk wireless metadata at the same time. To avoid an ERROR log entry for this scenario, specify either LoRaWAN or Sidewalk wireless data.

LoRaWAN device log entries

The log entries for your LoRaWAN wireless device can be classified based on the following event types:

- **Join and Rejoin**

When you add a LoRaWAN device and connect it to AWS IoT Core for LoRaWAN, before your device can send uplink data, you must complete a process called activation or join procedure. For more information, see [Add your wireless device to AWS IoT Core for LoRaWAN](#).

For this event type, if you set log level to INFO when configuring the CLI for your wireless gateway resource, then in the logs:

- If the event is successful, you'll see log messages that have a `LogLevel` of `INFO`. The messages will include details about the status of your join or rejoin request. Following shows an example of this log entry. For more information about the `LogLevel` and other fields in the log entry, see [AWS IoT Wireless resources and log levels](#).

```
{
  "timestamp": "2021-05-13T16:56:08.853Z",
  "resource": "WirelessDevice",
  "wirelessDeviceType": "LoRaWAN",
  "WirelessDeviceId": "5da85cc8-3361-4c79-8be3-3360fb87abda",
  "devEui": "feffff00000000e2",
  "event": "Rejoin",
  "LogLevel": "INFO",
  "message": "Rejoin succeeded"
}
```

- If there is an error, you'll see log entries that have a `LogLevel` of `ERROR`, and the messages will include details about the error. Examples of when an error can occur for the `Join` and `Rejoin` events include invalid LoRaWAN region setting, or invalid Message Integrity Code (MIC) check. Following example shows a join error due to MIC check. To resolve the error, check whether you've entered the correct root keys.

```
{
  "timestamp": "2020-11-24T01:46:50.883481989Z",
  "resource": "WirelessDevice",
  "wirelessDeviceType": "LoRaWAN",
  "WirelessDeviceId": "cb4c087c-1be5-4990-8654-ccf543ee9fff",
  "devEui": "58a0cb000020255c",
  "event": "Join",
  "LogLevel": "ERROR",
  "message": "invalid MIC. It's most likely caused by wrong root keys."
}
```

- **Uplink_Data and Downlink_Data**

The event type `Uplink_Data` is used for messages that are generated by AWS IoT Wireless when the payload is sent from your LoRaWAN or Sidewalk device to AWS IoT. The event type `Downlink_Data` is used for messages that are related to downlink messages that are sent from AWS IoT to the wireless device.

For this event type, if you set log level to INFO when configuring the CLI for your wireless devices, then in the logs, you'll see:

- If the event is successful, you'll see log messages that have a `LogLevel` of INFO. The messages will include details about the status of the uplink or downlink message that was sent and the wireless device identifier. Following shows an example of this log entry for a Sidewalk device. For more information about the `LogLevel` and other fields in the log entry, see [AWS IoT Wireless resources and log levels](#).

```
{
  "resource": "WirelessDevice",
  "wirelessDeviceId": "5371db88-d63d-481a-868a-e54b6431845d",
  "wirelessDeviceType": "Sidewalk",
  "event": "Downlink_Data",
  "LogLevel": "INFO",
  "messageId": "8da04fa8-037d-4ae9-bf67-35c4bb33da71",
  "message": "Message delivery succeeded. MessageId: 8da04fa8-037d-4ae9-bf67-35c4bb33da71. AWS IoT Core: {\"message\": \"0K\", \"traceId\": \"038b5b05-a340-d18a-150d-d5a578233b09\"}"
}
```

- If there is an error, you'll see log entries that have a `LogLevel` of ERROR, and the messages will include details about the error, which will help you resolve it. Examples of when an error can occur for the Registration event include: authentication issues, invalid or too many requests, unable to encrypt or decrypt the payload, or unable to find the wireless device using the specified ID. Following example shows a permission error encountered while processing a message.

```
{
  "resource": "WirelessDevice",
  "wirelessDeviceId": "cb4c087c-1be5-4990-8654-ccf543ee9fff",
  "wirelessDeviceType": "LoRaWAN",
  "event": "Uplink_Data",
  "LogLevel": "ERROR",
  "message": "Cannot assume role MessageId: ef38877f-3454-4c99-96ed-5088c1cd8dee. Access denied: 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"
```

```
}
```

Sidewalk device log entries

The log entries for your Sidewalk device can be classified based on the following event types:

- **Registration**

These log entries will help you monitor the status of any Sidewalk devices that you're registering with AWS IoT Wireless. For this event type, if you set log level to INFO when configuring the CLI for your wireless device resource, then in the logs, you'll see log messages that have a `LogLevel` of INFO and ERROR. The messages will include details about the registration progress from start to completion. ERROR log messages will contain information about how to troubleshoot issues with registering your device.

Following shows an example for a log message with log level of INFO. For more information about the `LogLevel` and other fields in the log entry, see [AWS IoT Wireless resources and log levels](#).

```
{
  "resource": "WirelessDevice",
  "wirelessDeviceId": "8d0b2775-e19b-4b2a-a351-cb8a2734a504",
  "wirelessDeviceType": "Sidewalk",
  "event": "Registration",
  "logLevel": "INFO",
  "message": "Successfully completed device registration. Amazon SidewalkId =
2000000002"
}
```

- **Uplink_Data and Downlink_Data**

The event types `Uplink_Data` and `Downlink_Data` for Sidewalk devices are similar to the corresponding event types for LoRaWAN devices. For more information, refer to the **Uplink_Data and Downlink_Data** section described previously for LoRaWAN device log entries.

Next steps

You've learned how to view log entries for your resources and the different log entries that you can view in the CloudWatch console after enabling logging for AWS IoT Wireless. While you can create filter streams using **Log groups**, we recommend that you use CloudWatch Insights to create and

use filter streams. For more information, see [Use CloudWatch Insights to filter logs for AWS IoT Wireless](#).

Use CloudWatch Insights to filter logs for AWS IoT Wireless

While you can use CloudWatch Logs to create filter expressions, we recommend that you use CloudWatch insights to more effectively create and use filter expressions depending on your application.

We recommend that you first use CloudWatch **Log groups** to learn about the different types of resources, its event types, and log levels that you can use to view log entries in the console. You can then use the examples of some filter expressions on this page as a reference to create your own filters for your AWS IoT Wireless resources.

Viewing AWS IoT logs in the CloudWatch Logs insights console

In the [CloudWatch console](#), CloudWatch logs appear in a log group named `/aws/iotwireless`. For more information about CloudWatch Logs, see [CloudWatch Logs](#).

To view your AWS IoT logs in the CloudWatch console

Navigate to the [CloudWatch console](#) and choose **Logs Insights** in the navigation pane.

1. In the **Filter** text box, enter `/aws/iotwireless`, and then choose the `/aws/iotwireless` Logs Insights.
2. To see a complete list of log groups, choose **Select log group(s)**. To look at log groups for AWS IoT Wireless, choose `/aws/iotwireless`.

You can now start entering queries to filter the log groups. The following sections contain some useful queries that'll help you gain insights about your resource metrics.

Create useful queries to filter and gain insights for AWS IoT Wireless

You can use filter expressions to show additional helpful log information with CloudWatch Insights. Following shows some sample queries:

Show only logs for specific resource types

You can create a query that'll help you show logs for only specific resource types, such as a LoRaWAN gateway or a Sidewalk device. For example, to filter logs to show only messages for

Sidewalk devices, you can enter the following query and choose **Run query**. To save this query, choose **Save**.

```
fields @message
| filter @message like /Sidewalk/
```

After the query runs, you'll see the results in the **Logs** tab, which shows the timestamps for logs related to Sidewalk devices in your account. You'll also see a bar graph, which will show the time at which the events occurred, if there were such events that occurred previously related to your Sidewalk device. Following shows an example if you expand one of the results in the **Logs** tab. Alternatively, if you want to troubleshoot errors related to Sidewalk devices, you can add another filter that sets the log level to ERROR and show only error information.

Field	Value
@ingestionTime	1623894967640
@log	954314929104:/aws/iotwireless
@logStream	WirelessDevice-
Downlink_Data-715adccfb34170214ec2f6667ddfa13cb5af2c3ddfc52fbee0e554a2e780bed	
@message	{ "resource": "WirelessDevice", "wirelessDeviceId": "3b058d05-4e84-4e1a-b026-4932bddf978d", "wirelessDeviceType": "Sidewalk", "devEui": "feffff000000011a", "event": "Downlink_Data", "logLevel": "INFO", "messageId": "7e752a10-28f5-45a5-923f-6fa7133fedda", "message": "Successfully sent downlink message. Amazon SidewalkId = 2000000006, Sequence number = 0" }
@timestamp	1623894967640
devEui	feffff000000011a
event	Downlink_Data
logLevel	INFO
message	Successfully sent downlink message. Amazon SidewalkId = 2000000006, Sequence number = 0
messageId	7e752a10-28f5-45a5-923f-6fa7133fedda
resource	WirelessDevice
wirelessDeviceId	3b058d05-4e84-4e1a-b026-4932bddf978d
wirelessDeviceType	Sidewalk

Show specific messages or events

You can create a query that'll help you show specific messages and observe when the events occurred. For example, if you want to see when your downlink message was sent from your LoRaWAN wireless device, you can enter the following query and choose **Run query**. To save this query, choose **Save**.

```
filter @message like /Downlink message sent/
```

After the query runs, you'll see the results in the **Logs** tab, which shows the timestamps when the downlink message was successfully sent to your wireless device. You'll also see a bar graph, which will show the time at which a downlink message was sent, if there were downlink messages were previously sent to your wireless device. Following shows an example if you expand one of the results in the **Logs** tab. Alternatively, if a downlink message wasn't sent, you can modify the query to display only results for when the message wasn't sent so that you can debug the issue.

Field	Value
@ingestionTime	1623884043676
@log	954314929104:/aws/iotwireless
@logStream	WirelessDevice-
Downlink_Data-42d0e6d09ba4d7015f4e9756fcfdc616d401cd85fe3ac19854d9fbd866153c872	
@message	{ "timestamp": "2021-06-16T22:54:00.770493863Z", "resource": "WirelessDevice", "wirelessDeviceId": "3b058d05-4e84-4e1a-b026-4932bddf978d", "wirelessDeviceType": "LoRaWAN", "devEui": "feffff000000011a", "event": "Downlink_Data", "logLevel": "INFO", "messageId": "7e752a10-28f5-45a5-923f-6fa7133fedda", "message": "Downlink message sent. MessageId: 7e752a10-28f5-45a5-923f-6fa7133fedda" }
@timestamp	1623884040858
devEui	feffff000000011a
event	Downlink_Data
logLevel	INFO
message	Downlink message sent. MessageId: 7e752a10-28f5-45a5-923f-6fa7133fedda
messageId	7e752a10-28f5-45a5-923f-6fa7133fedda
resource	WirelessDevice
timestamp	2021-06-16T22:54:00.770493863Z

```
wirelessDeviceId    3b058d05-4e84-4e1a-b026-4932bddf978d
wirelessDeviceType  LoRaWAN
```

Next steps

You've learned how to use CloudWatch Insights to gain more helpful information by creating queries to filter log messages. You can combine some of the filters described previously and design your own filters depending on the resource you're monitoring. For more information about using CloudWatch Insights, see [Analyzing log data with CloudWatch Insights](#).

After you've created queries with CloudWatch Insights, if you've saved them, you can load and run the saved queries as needed. Alternatively, if you click the **History** button in the CloudWatch **Logs Insights** console, you can view the previously run queries and rerun them as needed, or further modify them by creating additional queries.

Event notifications for AWS IoT Wireless

AWS IoT Wireless can publish messages to notify you of events for your LoRaWAN and Sidewalk devices. For example, the device registration event notifies you when the Sidewalk devices in your account have been registered.

How your resources can be notified of events

When certain events occur, AWS IoT Wireless publishes event notifications. For example, when you've provisioned your Sidewalk device, it generates an event. Each event causes a single event notification to be sent. Event notifications are published over MQTT with a JSON payload. The content of the payload depends on the type of event.

Note

Event notifications are published at least once. It's possible for them to be published more than once. The ordering of event notifications is not guaranteed.

Events and resource types

The following table shows the different types of events for which you'll receive notifications. The event types depend on whether the resource type is a wireless device, a wireless gateway, or a Sidewalk account. You can also enable events for your resources at the resource level. This applies to all resources of a particular type, or for select resources, as described in the following section. For more information, see [Event notifications for LoRaWAN resources](#) and [Event notifications for Sidewalk resources](#).

Event types based on resources

Resource	Resource type	Event type	
Wireless device	LoRaWAN	Join	
	Sidewalk	<ul style="list-style-type: none">• Device registration state• Proximity	

Resource	Resource type	Event type
Wireless gateway	LoRaWAN	Connection status
Sidewalk account	Sidewalk	<ul style="list-style-type: none"> • Device registration state • Proximity

Policy for receiving wireless event notifications

To receive event notifications, your device must use an appropriate policy. The policy 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 policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iotwireless:region:account:$aws/iotwireless/events/join/*",
        "arn:aws:iotwireless:region:account:$aws/iotwireless/events/
connection_status/*",
        "arn:aws:iotwireless:region:account:$aws/iotwireless/events/
device_registration_state/*",
        "arn:aws:iotwireless:region:account:$aws/iotwireless/events/proximity/*"
      ]
    }
  ]
}
```

Format of MQTT topics for wireless events

To send you event notifications, AWS IoT uses MQTT reserved topics that begin with a dollar sign (\$). You can publish and subscribe to these reserved topics. However, you can't create new topics that begin with a dollar sign.

Note

MQTT topics are specific to your AWS account and use the format `arn:aws:iotwireless:aws-region:AWS-account-ID:topic/Topic`. For more information, see [MQTT topics](#) in the *AWS IoT developer guide*.

Reserved MQTT topics for wireless devices use the following format:

- **Resource-level topics**

These topics apply to all resources of a particular type in your AWS account.

```
$aws/iotwireless/events/{eventName}/{eventType}/{resourceType}/resources
```

- **Identifier-level topics**

These topics apply to select resources of a particular type, specified by the resource identifier.

```
$aws/iotwireless/events/{eventName}/{eventType}/{resourceType}/
{resourceIdentifierType}/{resourceID}/{id}
```

For more information about topics at resource and identifier levels, see [Event configurations](#).

The following table shows examples of MQTT topics for the various events:

Events and MQTT topics

Event	MQTT topic	Notes
Sidewalk device registration state	<ul style="list-style-type: none"> • Resource-level topic <code>\$aws/iotwireless/events/device_registration_state/{eventType}/sidewalk/wireless_devices</code> • Identifier-level topic 	<ul style="list-style-type: none"> • {eventType} can be registered or provisioned • {resourceType} can be sidewalk_accounts or wireless_devices • {resourceID} is the amazon_id for sidewalk_accounts and wireless_device_id for wireless_devices

Event	MQTT topic	Notes
	<pre>\$aws/iotwireless/ events/dev ice_regis tration_state/ {eventType}/ sidewalk/{ resourceType}/ {resourceID}/ {id}</pre>	
Sidewalk proximity	<ul style="list-style-type: none"> • Resource-level topic <pre>\$aws/iotwireless/ events/pro ximity/{e ventType}/ sidewalk/wireless _devices</pre> <ul style="list-style-type: none"> • Identifier-level topic <pre>\$aws/iotwireless/ events/pro ximity/{e ventType} /sidewalk/ {resourceType}/{r esourceID}/{id}</pre>	<ul style="list-style-type: none"> • {eventType} can be beacon_discovered or beacon_lost • {resourceType} can be sidewalk_accounts or wireless_devices • {resourceID} is the amazon_id for sidewalk_accounts and wireless_device_id for wireless_devices

Event	MQTT topic	Notes
LoRaWAN join	<ul style="list-style-type: none"> • Resource-level topic <code>\$aws/iotwireless/ events/join/ {eventType}/ lorawan/wirel ess_devices</code> • Identifier-level topic <code>\$aws/iotwireless/ events/join/ {eventType}/ lorawan/wirel ess_devices/ {resourceID}/{i d}</code> 	<ul style="list-style-type: none"> • {eventType} can be join_req_0_received or join_req_2_received or join_accepted • {resourceID} can be wireless_device_id or dev_eui
LoRaWAN gateway connection status	<ul style="list-style-type: none"> • Resource-level topic <code>\$aws/iotwireless/ events/join/ {eventType}/ lorawan/wirel ess_gateways</code> • Identifier-level topic <code>\$aws/iotwireless/ events/join/ {eventType}/ lorawan/wirel ess_gateways/ {resourceID}/{ id}</code> 	<ul style="list-style-type: none"> • {eventType} can be connected or disconnected • {resourceID} can be wireless_gateway_id or gateway_eui

For more information about the different events, see [Event notifications for LoRaWAN resources](#) and [Event notifications for Sidewalk resources](#).

If you've subscribed to these topics, you'll be notified when a message is published to one of the topics. For more information, see [MQTT reserved topics](#) in the *AWS IoT developer guide*.

Pricing for wireless events

For information about pricing for subscribing to events and for receiving notifications, see [AWS IoT Core pricing](#).

Enable events for wireless resources

Before subscribers to the reserved topics can receive messages, you must enable event notifications. To do this, you can use the AWS Management Console, or the AWS IoT Wireless API or AWS CLI.

Event configurations

You can configure events to send notifications to either all resources that belong to a particular type, or for individual wireless resources. The resource type can be a wireless gateway, Sidewalk partner account, or a wireless device, which can be a LoRaWAN or Sidewalk device. For information about the type of events that you can enable for your wireless devices, see [Event types for LoRaWAN resources](#) and [Event types for Sidewalk resources](#).

All resources

You can enable events such that all resources in your AWS account that belong to a particular resource type receives notifications. For example, you can enable an event that notifies you of changes to connection status for all LoRaWAN gateways that you've onboarded with AWS IoT Core for LoRaWAN. Monitoring these events will help you get notified in cases such as when certain LoRaWAN gateways in your resource fleet get disconnected or if a beacon is lost for a number of Sidewalk devices in your AWS account.

Individual resources

You can also add individual LoRaWAN and Sidewalk resources to your event configuration and enable notifications for them. This will help you monitor individual resources of a particular type. For example, you can add select LoRaWAN and Sidewalk devices to your configuration and receive notifications for join or device registration state events for these resources.

Prerequisites

Your LoRaWAN or Sidewalk resource must have an appropriate policy that allows it to receive event notifications. For more information, see [Policy for receiving wireless event notifications](#).

Enable notifications using the AWS Management Console

To enable event messages from the console, go to the [Settings](#) tab of the AWS IoT console, and then go to the **LoRaWAN and Sidewalk event notification** section.

You can enable notifications for all resources in your AWS account that belong to a particular resource type and monitor them.

To enable notifications for all resources

1. In the **LoRaWAN and Sidewalk event notification** section, go to the **All resources** tab, choose **Action**, and then choose **Manage events**.
2. Enable the events that you want to monitor, and then choose **Update events**. If you no longer want to monitor certain events, choose **Action** and choose **Manage events**, and then disable those events.

You can also enable notifications for individual resources in your AWS account that belong to a particular resource type and monitor them.

To enable notifications for individual resources

1. In the **LoRaWAN and Sidewalk event notification** section, choose **Action**, and then choose **Add resources**.
2. Select the resources and events for which you want to receive notifications:
 - a. Choose whether you want to monitor events for your **LoRaWAN resources** or **Sidewalk resources**.
 - b. Depending on the resource type, you can choose the events you want to enable for the resources. You can then subscribe to these events and receive notifications. If you choose:
 - **LoRaWAN resources**: You can enable **join** events for your LoRaWAN devices or **connection status** events for your LoRaWAN gateways.
 - **Sidewalk resources**: You can enable **device registration state** or **proximity** events or both for your Sidewalk partner accounts and Sidewalk devices.

3. Depending on the resource type and events that you chose, select the wireless devices or gateways that you want to monitor. You can select up to 250 resources for all resources combined.
4. Choose **Submit** to add your resources.

The resources that you add will appear with their MQTT topics in the tab for your resource type in the **LoRaWAN and Sidewalk event notification** section of the console.

- **LoRaWAN join** events and events for your Sidewalk devices will appear in the **Wireless devices** section of the console.
- **Connection status** events for your LoRaWAN gateways will appear in the **Wireless gateways** section.
- **Device registration state** and **proximity** events for your Sidewalk accounts will appear in the **Sidewalk accounts** tab.

Subscribe to topics using MQTT client

Depending on whether you enabled events for all resources or for individual resource types, the events that you enabled will appear in the console with their MQTT topics on the **All resources** tab or the tab for the specified resource type.

- If you choose one of the MQTT topics, you can go to the MQTT client to subscribe to these topics and receive messages.
- If you've added multiple events, you can subscribe to multiple event topics and receive notifications for them. To subscribe to multiple topics, choose your topics, and choose **Action** and then choose **Subscribe**.

Enable notifications using the AWS CLI

You can configure events and add resources to your configuration by using the AWS IoT Wireless API or the AWS CLI.

Enable notifications for all resources

You can enable notifications for all resources in your AWS account that belong to a particular resource type and monitor them by using the [UpdateEventConfigurationByResourceTypes](#) API or the [update-event-configuration-by-resource-types](#) CLI command. For example:


```
aws iotwireless update-event-configuration-by-resource-types \  
  --cli-input-json input.json
```

Contents of input.json

```
{  
  "DeviceRegistrationState": {  
    "Sidewalk": {  
      "AmazonIdEventTopic": "Enabled"  
    }  
  },  
  "ConnectionStatus": {  
    "LoRaWAN": {  
      "WirelessGatewayEventTopic": "Enabled"  
    }  
  }  
}
```

Note

All quotation marks (") are escaped with a backslash (\).

You can get the current event configuration by calling the [GetEventConfigurationByResourceTypes](#) API or by using the [get-event-configuration-by-resource-types](#) CLI command. For example:

```
aws iotwireless get-event-configuration-by-resource-types
```

Enable notifications for individual resources

To add individual resources to your event configuration and control which events are published by using the API or CLI, call the [UpdateResourceEventConfiguration](#) API or use the [update-resource-event-configuration](#) CLI command. For example:

```
aws iotwireless update-resource-event-configuration \  
  --identifer 1ffd32c8-8130-4194-96df-622f072a315f \  
  --identifier-type WirelessDeviceId \  
  --cli-input-json input.json
```

Contents of input.json

```
{
  "Join": {
    "LoRaWAN": {
      "DevEuiEventTopic": "Disabled"
    },
    "WirelessDeviceIdEventTopic": "Enabled"
  }
}
```

Note

All quotation marks (") are escaped with a backslash (\).

You can get the current event configuration by calling the [GetResourceEventConfiguration](#) API or by using the [get-resource-event-configuration](#) CLI command. For example:

```
aws iotwireless get-resource-event-configuration \
  --identifier-type WirelessDeviceId \
  --identifier 1ffd32c8-8130-4194-96df-622f072a315f
```

List event configurations

You can also use the AWS IoT Wireless API or the AWS CLI to list event configurations where at least one event topic has been enabled. To list configurations, use the [ListEventConfigurations](#) API operation or by using the [list-event-configurations](#) CLI command. For example:

```
aws iotwireless list-event-configurations --resource-type WirelessDevice
```

Event notifications for LoRaWAN resources

You can use the AWS Management Console or AWS IoT Wireless API operations to notify you of events for your LoRaWAN devices and gateways. For information about event notifications and how to enable them, see [Event notifications for AWS IoT Wireless](#) and [Enable events for wireless resources.d](#)

Event types for LoRaWAN resources

Events that you can enable for your LoRaWAN resources include:

- Join events that notify you of join events for your LoRaWAN device. You'll receive notifications when a device joins with AWS IoT Core for LoRaWAN, or when a rejoin request of type 0 or type 2 is received.
- Connection status events that notify you when the connection status of your LoRaWAN gateway changes to connected or disconnected.

The following sections contain more information about the events for your LoRaWAN resources:

Topics

- [LoRaWAN join events](#)
- [Connection status events](#)

LoRaWAN join events

AWS IoT Core for LoRaWAN can publish messages to notify you of join events for LoRaWAN devices that you onboard to AWS IoT. Join events notify you when a join or rejoin request of type 0 or type 2 is received, and the device has joined with AWS IoT Core for LoRaWAN.

How join events work

When you onboard your LoRaWAN devices with AWS IoT Core for LoRaWAN, AWS IoT Core for LoRaWAN performs a *join* procedure for your device with AWS IoT Core for LoRaWAN. Your device then becomes activated for use and can send an uplink message to indicate that it's available. After the device has joined, uplink and downlink messages can be exchanged between your device and AWS IoT Core for LoRaWAN. For more information about onboarding your device, see [Onboard your devices to AWS IoT Core for LoRaWAN](#).

You can enable events to notify you when your device has joined with AWS IoT Core for LoRaWAN. You'll also be notified if the join event fails and when a rejoin request of type 0 or type 2 is received, and when it's accepted.

Enable LoRaWAN join events

Before subscribers to the LoRaWAN join reserved topics can receive messages, you must enable event notifications for them from the AWS Management Console, or by using the API or CLI. You can enable these events for all LoRaWAN resources in your AWS account or for select resources. For information about how to enable these events, see [Enable events for wireless resources](#).

Format of MQTT topics for LoRaWAN events

Reserved MQTT topics for LoRaWAN devices use the following format. If you've subscribed to these topics, then all LoRaWAN devices that are registered to your AWS account can receive the notification:

- **Resource-level topics**

```
$aws/iotwireless/events/{eventName}/{eventType}/lorawan/wireless_devices
```

- **Identifier topics**

```
$aws/iotwireless/events/{eventName}/{eventType}/lorawan/wireless_devices/  
{resourceID}/{id}
```

Where:

{eventName}

{eventName} must be join.

{eventType}

{eventType} can be:

- join_req_received
- rejoin_req_0_received
- rejoin_req_2_received
- join_accepted

{resourceID}

{resourceID} can be dev_eui or wireless_device_id.

For example, you can subscribe to the following topics to receive an event notification when AWS IoT Core for LoRaWAN has accepted a join request from your devices.

```
$aws/iotwireless/events/join/join_accepted/lorawan/wireless_devices/  
wireless_device_id/{id}
```

You can also use the + wildcard character to subscribe to multiple topics at the same time. The + wildcard character matches any string in the level that contains the character, such as the following topic:

```
$aws/iotwireless/events/join/join_req_received/lorawan/wireless_devices/  
wireless_device_id/+
```

Note

You can't use the wildcard character # to subscribe to the reserved topics.

For more information about using the + wildcard when subscribing to topics, see [MQTT topic filters](#) in the *AWS IoT developer guide*.

Message payload for LoRaWAN join event

The following shows the message payload for the LoRaWAN join event.

```
{  
  // General fields  
  "eventId": "string",  
  "eventType": "join_req_received|rejoin_req_0_received|rejoin_req_2_received|  
join_accepted",  
  "WirelessDeviceId": "string",  
  "timestamp": "timestamp",  
  
  // Event-specific fields  
  "LoRaWAN": {  
    "DevEui": "string",  
  
    // The fields below are optional indicating that it can be a null value.  
    "DevAddr": "string",  
    "JoinEui": "string",  
    "AppEui": "string",  
  }  
}
```

```
}
```

The payload contains the following attributes:

eventId

A unique event ID that's generated by AWS IoT Core for LoRaWAN (string).

eventType

The type of event that occurred. Can be one of the following values:

- `join_req_received`: This field will show the EUI parameters `JoinEui` or `AppEui`
- `rejoin_req_0_received`
- `rejoin_req_2_received`
- `join_accepted`: This field will show the `NetId` and `DevAddr`.

wirelessDeviceId

The ID of the LoRaWAN device.

timestamp

The Unix timestamp of when the event occurred.

DevEui

The unique identifier of the device found on the device label or device documentation.

DevAddr and EUIs (optional)

These fields are the optional device address and the EUI parameters `JoinEUI` or `AppEUI`.

Connection status events

AWS IoT Core for LoRaWAN can publish messages to notify you of connection status events for LoRaWAN gateways that you onboard to AWS IoT. Connection status events notify you when the connection status of a LoRaWAN gateway changes to connected or disconnected.

How connection status events work

After you've onboarded your gateway to AWS IoT Core for LoRaWAN, you can connect your gateway to AWS IoT Core for LoRaWAN and verify its connection status. This event notifies you when your gateway connection status changes to connected or disconnected. For more information

about onboarding and connecting your gateway to AWS IoT Core for LoRaWAN, see [Onboard your gateways to AWS IoT Core for LoRaWAN](#) and [Connect your LoRaWAN gateway and verify its connection status](#).

Format of MQTT topics for LoRaWAN gateways

Reserved MQTT topics for LoRaWAN gateways use the following format. If you've subscribed to these topics, then all LoRaWAN gateways that are registered to your AWS account can receive the notification:

- For resource-level topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/lorawan/wireless_gateways
```

- For identifier topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/lorawan/wireless_gateways/{resourceID}/{id}
```

Where:

{eventName}

{eventName} must be `connection_status`.

{eventType}

{eventType} can be `connected` or `disconnected`.

{resourceID}

{resourceID} can be `gateway_eui` or `wireless_gateway_id`.

For example, you can subscribe to the following topics to receive an event notification when all your gateways have connected to AWS IoT Core for LoRaWAN:

```
$aws/iotwireless/events/connection_status/connected/lorawan/wireless_gateways/wireless_gateway_id/{id}
```

You can also use the `+` wildcard character to subscribe to multiple topics at the same time. The `+` wildcard character matches any string in the level that contains the character, such as the following topic:

```
$aws/iotwireless/events/connection_status/connected/lorawan/  
wireless_gateways/wireless_gateway_id/+
```

Note

You can't use the wildcard character # to subscribe to the reserved topics.

For more information about using the + wildcard when subscribing to topics, see [MQTT topic filters](#) in the *AWS IoT developer guide*.

Message payload for connection status events

The following shows the message payload for the connection status event.

```
{  
  // General fields  
  "eventId": "string",  
  "eventType": "connected|disconnected",  
  "WirelessGatewayId": "string",  
  "timestamp": "timestamp",  
  
  // Event-specific fields  
  "LoRaWAN": {  
    "GatewayEui": "string"  
  }  
}
```

The payload contains the following attributes:

eventId

A unique event ID that's generated by AWS IoT Core for LoRaWAN (string).

eventType

The type of event that occurred. Can be connected or disconnected.

wirelessGatewayId

The ID of the LoRaWAN gateway.

timestamp

The Unix timestamp of when the event occurred.

GatewayEui

The unique identifier of the gateway found on the gateway label or gateway documentation.

Event notifications for Sidewalk resources

You can use the AWS Management Console or AWS IoT Wireless API operations to notify you of events for your Sidewalk devices and partner accounts. For information about event notifications and how to enable them, see [Event notifications for AWS IoT Wireless](#) and [Enable events for wireless resources](#).

Event types for Sidewalk resources

Events that you can enable for your Sidewalk resources include:

- Device events that notify you of changes to the state of your Sidewalk device, such as when the device has been registered and is ready to use.
- Proximity events that notify you when AWS IoT Wireless receives a notification from Amazon Sidewalk that a beacon has been discovered or lost.

The following sections contain more information about the events for your Sidewalk resources:

Topics

- [Device registration state events](#)
- [Proximity events](#)
- [Message delivery status events](#)

Device registration state events

Device registration state events publish event notifications when there is a change in the device registration state, such as when a Sidewalk device has been provisioned or registered. The events provide you information about the different states that the device goes through from when it's provisioned to when it has been registered.

How device registration state events work

When you onboard your Sidewalk device with Amazon Sidewalk and AWS IoT Wireless, AWS IoT Wireless performs a `create` operation and adds your Sidewalk device to your AWS account. Your device then enters the provisioned state and the `eventType` becomes `provisioned`. For more information about onboarding your device, see [Getting started with AWS IoT Core for Amazon Sidewalk](#).

After the device has been provisioned, Amazon Sidewalk performs a `register` operation to register your Sidewalk device with AWS IoT Wireless. The registration process starts, where the encryption and session keys are set up with AWS IoT. When the device is registered, the `eventType` becomes `registered`, and your device is ready to use.

After the device has been registered, Sidewalk can send a request to `deregister` your device. AWS IoT Wireless then fulfills the request and changes the device state back to `provisioned`. For more information about the device states, see [DeviceState](#).

Enable notifications for device registration state events

Before subscribers to the device registration state reserved topics can receive messages, you must enable event notifications for them from the AWS Management Console, or by using the API or CLI. You can enable these events for all Sidewalk resources in your AWS account or for select resources. For information about how to enable these events, see [Enable events for wireless resources](#).

Format of MQTT topics for device registration state events

To notify you of device registration state events, you can subscribe to MQTT reserved topics that begin with a dollar (\$) sign. For more information, see [MQTT topics](#) in the *AWS IoT developer guide*.

Reserved MQTT topics for Sidewalk device registration state events use the following format:

- For resource-level topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/wireless_devices
```

- For identifier topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/{resourceType}/  
{resourceID}/{id}
```

Where:

{eventName}

{eventName} must be `device_registration_state`.

{eventType}

{eventType} can be `provisioned` or `registered`.

{resourceType}

{resourceType} can be `sidewalk_accounts` or `wireless_devices`.

{resourceID}

{resourceID} is `amazon_id` for {resourceType} of `sidewalk_accounts` and `wireless_device_id` for {resourceType} of `wireless_devices`.

You can also use the `+` wildcard character to subscribe to multiple topics at the same time. The `+` wildcard character matches any string in the level that contains the character. For example, if you want to be notified of all possible event types (`provisioned` and `registered`) and for all devices registered to a particular Amazon ID, you can use the following topic filter:

```
$aws/iotwireless/events/device_registration_state/+/sidewalk/  
sidewalk_accounts/amazon_id/+
```

Note

You can't use the wildcard character `#` to subscribe to the reserved topics. For more information about topic filters, see [MQTT topic filters](#) in the *AWS IoT developer guide*.

Message payload for device registration state events

After you enable notifications for device registration state events, event notifications are published over MQTT with a JSON payload. These events contain the following example payload:

```
{  
  "eventId": "string",  
  "eventType": "provisioned|registered",  
  "WirelessDeviceId": "string",  
  "timestamp": "timestamp",  
  
  // Event-specific fields
```

```
"operation": "create|deregister|register",
"Sidewalk": {
  "AmazonId": "string",
  "SidewalkManufacturingSn": "string"
}
}
```

The payload contains the following attributes:

eventId

A unique event ID (string).

eventType

The type of event that occurred. Can be provisioned or registered.

wirelessDeviceId

The identifier of the wireless device.

timestamp

The Unix timestamp of when the event occurred.

operation

The operation that triggered the event. Valid values are create, register, and deregister.

sidewalk

The Sidewalk Amazon ID or SidewalkManufacturingSn for which you want to receive event notifications.

Proximity events

Proximity events publish event notifications when AWS IoT receives a beacon from the Sidewalk device. When your Sidewalk device approaches Amazon Sidewalk, beacons that are sent from your device are filtered by Amazon Sidewalk at regular intervals and received by AWS IoT Wireless. AWS IoT Wireless then notifies you of these events when a beacon is received.

How proximity events work

Proximity events notify you when AWS IoT receives a beacon, Your Sidewalk devices can emit beacons any time. When your device is near Amazon Sidewalk, Sidewalk receives the beacons and

forwards them to AWS IoT Wireless at regular time intervals. Amazon Sidewalk has configured this time interval as 10 minutes. When AWS IoT Wireless receives the beacon from Sidewalk, you'll be notified of the event.

Proximity events will notify you when a beacon is discovered or when a beacon is lost. You can configure the intervals at which you're notified of the proximity event.

Enable notifications for proximity events

Before subscribers to the Sidewalk proximity reserved topics can receive messages, you must enable event notifications for them from the AWS Management Console, or by using the API or CLI. You can enable these events for all Sidewalk resources in your AWS account or for select resources. For information about how to enable these events, see [Enable events for wireless resources](#).

Format of MQTT topics for proximity events

To notify you of proximity events, you can subscribe to MQTT reserved topics that begin with a dollar (\$) sign. For more information, see [MQTT topics](#) in the *AWS IoT developer guide*.

Reserved MQTT topics for Sidewalk proximity events use the format:

- For resource-level topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/wireless_devices
```

- For identifier topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/{resourceType}/  
{resourceID}/{id}
```

Where:

{eventName}

{eventName} must be `proximity`.

{eventType}

{eventType} can be `beacon_discovered` or `beacon_lost`.

{resourceType}

{resourceType} can be `sidewalk_accounts` or `wireless_devices`.

{resourceID}

{resourceID} is `amazon_id` for {resourceType} of `sidewalk_accounts` and `wireless_device_id` for {resourceType} of `wireless_devices`.

You can also use the `+` wildcard character to subscribe to multiple topics at the same time. The `+` wildcard character matches any string in the level that contains the character. For example, if you want to be notified of all possible event types (`beacon_discovered` and `beacon_lost`) and for all devices registered to a particular Amazon ID, you can use the following topic filter:

```
$aws/iotwireless/events/proximity/+sidewalk/sidewalk_accounts/amazon_id/+
```

Note

You can't use the wildcard character `#` to subscribe to the reserved topics. For more information about topic filters, see [MQTT topic filters](#) in the *AWS IoT developer guide*.

Message payload for proximity events

After you enable notifications for proximity events, event messages are published over MQTT with a JSON payload. These events contain the following example payload:

```
{
  "eventId": "string",
  "eventType": "beacon_discovered|beacon_lost",
  "WirelessDeviceId": "string",
  "timestamp": "1234567890123",

  // Event-specific fields
  "Sidewalk": {
    "AmazonId": "string",
    "SidewalkManufacturingSn": "string"
  }
}
```

The payload contains the following attributes:

eventId

A unique event ID, which is a string.

eventType

The type of event that occurred. Can be `beacon_discovered` or `beacon_lost`.

WirelessDeviceId

The identifier of the wireless device.

timestamp

The Unix timestamp of when the event occurred.

sidewalk

The Sidewalk Amazon ID or `SidewalkManufacturingSn` for which you want to receive event notifications.

Message delivery status events

Message delivery status events publish event notifications about the status of messages that are exchanged between your Sidewalk devices and AWS IoT Wireless. Event notifications are published for both downlink messages that are sent from AWS IoT Wireless to the Sidewalk device, and uplink messages that are sent from your device to AWS IoT Wireless.

How message delivery status events work

After you've onboarded your Sidewalk device to AWS IoT Wireless and connected your device, messages can be exchanged between your device and AWS IoT Wireless. The events publish notifications about the message delivery status that indicate whether these messages were successfully delivered to your device or to AWS IoT Wireless.

For example, if an uplink message is received from the device with an acknowledge (ACK) flag, a notification is published indicating that the message was delivered successfully. When you send downlink messages from AWS IoT Wireless to the Sidewalk device, the `SendDataToWirelessDevice` API returns a `MessageId` for the downlink message even if packets have dropped or the message wasn't delivered. In this case, the message delivery status events return an error indicating that the message failed to deliver to the device.

Enable notifications for message delivery status events

Before subscribers to the Sidewalk message delivery status reserved topics can receive messages, you must enable event notifications for them using the AWS IoT Wireless API or the AWS CLI. You can enable these events for all Sidewalk resources in your AWS account or for select resources.

Note

The Sidewalk message delivery status event configuration isn't available in the console.

For information about how to enable these events, see [Enable notifications using the AWS CLI](#).

Format of MQTT topics for message delivery status events

To receive notifications about message delivery status events, you can subscribe to MQTT reserved topics that begin with a dollar (\$) sign. For more information, see [topics](#).

Reserved MQTT topics for Sidewalk proximity events use the format:

- For resource-level topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/wireless_devices
```

- For identifier topics:

```
$aws/iotwireless/events/{eventName}/{eventType}/sidewalk/{resourceType}/  
{resourceID}/{id}
```

Where:

{eventName}

{eventName} must be `message_delivery_status`.

{eventType}

{eventType} can be `success` or `error`.

{resourceType}

{resourceType} can be `sidewalk_accounts` or `wireless_devices`.

{resourceID}

{resourceID} is `amazon_id` for {resourceType} of `sidewalk_accounts` and `wireless_device_id` for {resourceType} of `wireless_devices`.

You can also use the `+` wildcard character to subscribe to multiple topics at the same time. The `+` wildcard character matches any string in the level that contains the character. For example, if you want to be notified of all possible event types (`success` and `error`) and for all devices registered to a particular Amazon ID, you can use the following topic filter:

```
$aws/iotwireless/events/message_delivery_status/+/sidewalk/  
sidewalk_accounts/amazon_id/+
```

Note

You can't use the wildcard character `#` to subscribe to the reserved topics. For more information about topic filters, see [topicfilters](#).

Message payload for message delivery status events

After you enable notifications for message delivery status events, event messages are published over MQTT with a JSON payload. These events contain the following example payload depending on whether the event is a success, indicating that the device successfully received the message, or an error.

Success events

The following shows the payload format when the event is a success.

```
{  
  "eventId": "string",  
  "eventType": "success",  
  "WirelessDeviceId": "string",  
  "timestamp": "timestamp",  
  "Sidewalk": {  
    "Seq": "Integer",  
    "MsgType": "CUSTOM_COMMAND_ID_RESP",  
    "CmdExStatus": "COMMAND_EXEC_STATUS_SUCCESS"  
  }  
}
```

```
}
```

The payload contains the following attributes:

eventId

A unique event ID, which is a string.

eventType

The type of event that occurred. Can be `success` or `error`. In this case, the `eventType` is `error`.

WirelessDeviceId

The identifier of the wireless device.

timestamp

The Unix timestamp of when the event occurred.

sidewalk

The Sidewalk wrapper that contains the status code for success messages, sequence number of message, and the message type.

Error events

The following shows the payload format when the event indicates that an error occurred.

```
{
  "eventId": "string",
  "eventType": "error" ,
  "WirelessDeviceId": "string",
  "timestamp": "timestamp",
  "Sidewalk": {
    "Seq": "Integer",
    "Status": "DeviceNotReachable" | "RADIO_TX_ERROR" | "MEMORY_ERROR"
  }
}
```

The payload contains similar attributes as when the `eventType` is a success. Following are some differences or additional attributes:

eventType

The type of event that occurred. In this case, the eventType is an `error`.

sidewalk

The Sidewalk wrapper that contains the sequence number and the status code that indicates why the downlink message wasn't sent successfully.

AWS IoT Wireless API operations

You can perform the following additional API operations when onboarding your LoRaWAN or Sidewalk end devices, or when creating an import task for provisioning Sidewalk end devices in bulk.

The following sections contain additional information about these API operations.

Topics

- [AWS IoT Wireless API operations for device profiles](#)
- [AWS IoT Wireless API operations for LoRaWAN and Sidewalk devices](#)
- [AWS IoT Wireless API operations for destinations for wireless devices](#)
- [AWS IoT Core for Amazon Sidewalk API operations for bulk provisioning](#)

AWS IoT Wireless API operations for device profiles

You can perform the following API operations for your LoRaWAN and Sidewalk device profiles:

- [CreateDeviceProfile](#) API or the [create-device-profile](#) CLI
- [GetDeviceProfile](#) API or the [get-device-profile](#) CLI
- [ListDeviceProfiles](#) API or the [list-device-profiles](#) CLI
- [DeleteDeviceProfile](#) API or the [delete-device-profile](#) CLI

The following sections show you how to list and delete profiles. For information about creating and retrieving device profiles, see:

- [Add device profiles](#)
- [Step 1: Create a device profile](#)

List device profiles in your AWS account

You can use the [ListDeviceProfiles](#) API operation to list device profiles in your AWS account that you added to AWS IoT Wireless. You can use this information to identify the devices that you want to associate this profile to.

To filter the list to display only LoRaWAN or Sidewalk device profiles, set the Type when running the API. Following shows an example CLI command:

```
aws iotwireless list-device-profiles --wireless-device-type "Sidewalk"
```

Running this command returns a list of device profiles that you added, including their profile identifier and Amazon Resource Name (ARN). To retrieve additional details about a specific profile, use the `GetDeviceProfile` API.

```
{
  "DeviceProfileList": [
    {
      "Name": "SidewalkDeviceProfile1",
      "Id": "12345678-a1b2-3c45-67d8-e90fa1b2c34d",
      "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/12345678-
a1b2-3c45-67d8-e90fa1b2c34d"
    },
    {
      "Name": "SidewalkDeviceProfile2",
      "Id": "a1b2c3d4-5678-90ab-cdef-12ab345c67de",
      "Arn": "arn:aws:iotwireless:us-east-1:123456789012:DeviceProfile/
a1b2c3d4-5678-90ab-cdef-12ab345c67de"
    }
  ]
}
```

Delete device profiles from your AWS account

You can delete your device profiles using the [DeleteDeviceProfile](#) API operation. The following shows an example CLI command:

Warning

Deletion actions can't be undone. The device profile will be permanently removed from your AWS account.

```
aws iotwireless delete-device-profile --name "SidewalkProfile"
```

This command doesn't produce any output. You can use the `GetDeviceProfile` API or the `ListDeviceProfiles` API operation to verify that the profile has been removed from your account.

AWS IoT Wireless API operations for LoRaWAN and Sidewalk devices

You can perform the following API operations for your LoRaWAN and Sidewalk devices:

- [CreateWirelessDevice](#) API or the [create-wireless-device](#) CLI
- [GetWirelessDevice](#) API or the [get-wireless-device](#) CLI
- [ListWirelessDevices](#) API or the [list-wireless-devices](#) CLI
- [DeleteWirelessDevice](#) API or the [delete-wireless-device](#) CLI
- [UpdateWirelessDevice](#) API or the [update-wireless-device](#) CLI
- [AssociateWirelessDeviceWithThing](#) API or the [associate-wireless-device-with-thing](#) CLI
- [DisassociateWirelessDeviceFromThing](#) API or the [disassociate-wireless-device-from-thing](#) CLI

The following sections show you how to list and delete devices. For information about creating wireless devices and retrieving device information, see:

- [Add your wireless device to AWS IoT Core for LoRaWAN](#)
- [Step 2: Add your Sidewalk device](#)

Associate wireless devices in your AWS account to an IoT thing

To associate your LoRaWAN and Sidewalk devices with an AWS IoT thing, use the `AssociateWirelessDeviceWithThing` API operation.

Things in AWS IoT make it easier to search and manage your devices. Associating a thing with your device lets the device access other AWS IoT Core features. For more information about using this API, see [AssociateWirelessDeviceWithThing](#).

The following shows an example of running this command. Running this command doesn't produce any output.

```
aws iotwireless associate-wireless-device-with-thing \  
  --id "12345678-a1b2-3c45-67d8-e90fa1b2c34d" \  
  --thing-arn "arn:aws:iot:us-east-1:123456789012:thing/MySidewalkThing"
```

To disassociate your wireless device from an AWS IoT thing, use the [DisassociateWirelessDeviceFromThing](#) API operation, as shown in the following example.

```
aws iotwireless disassociate-wireless-device-from-thing \  
  --id "12345678-a1b2-3c45-67d8-e90fa1b2c34d"
```

List wireless devices in your AWS account

To list wireless devices in your AWS account that you added to AWS IoT Wireless, use the [ListWirelessDevices](#) API operation. To filter the list to return only LoRaWAN or Sidewalk devices, set the `WirelessDeviceType`.

The following shows an example of running this command:

```
aws iotwireless list-wireless-devices --wireless-device-type Sidewalk
```

Running this command returns a list of devices that you added, including their profile identifier and the Amazon Resource Name (ARN). To retrieve additional details about a specific device, use the [GetWirelessDevice](#) API operation.

```
{  
  "WirelessDeviceList": [  
    {  
      "Name": "mySidewalkDevice",  
      "DestinationName": "SidewalkDestination",  
      "Id": "1ffd32c8-8130-4194-96df-622f072a315f",  
      "Type": "Sidewalk",  
      "Sidewalk": {  
        "SidewalkId": "1234567890123456"  
      },  
      "Arn": "arn:aws:iotwireless:us-  
east-1:123456789012:WirelessDevice/1ffd32c8-8130-4194-96df-622f072a315f"  
    }  
  ]  
}
```

Delete wireless devices from your AWS account

To delete your wireless devices, pass the `WirelessDeviceID` of the devices you want to delete to the [DeleteWirelessDevice](#) API operation.

The following shows an example command:

```
aws iotwireless delete-wireless-device --id "23456789-abcd-0123-bcde-fabc012345678"
```

This command doesn't produce any output. You can use the `GetWirelessDevice` API or the `ListWirelessDevices` API operation to verify that the device has been removed from your account.

AWS IoT Wireless API operations for destinations for wireless devices

You can perform the following API operations for destinations for your LoRaWAN and Sidewalk devices:

- [CreateDestination](#) API or the [create-destination](#) CLI
- [GetDestination](#) API or the [get-destination](#) CLI
- [UpdateDestination](#) API or the [update-destination](#) CLI
- [ListDestinations](#) API or the [list-destinations](#) CLI
- [DeleteDestination](#) API or the [delete-destination](#) CLI

The following sections show you how to get, list, update, and delete destinations. For information about creating destinations, see [Add a destination for your Sidewalk end device](#).

Get information about your destination

You can use the [GetDestination](#) API operation to get information about the destination that you added to your account for AWS IoT Wireless. Provide the destination name as input to the API. The API will return information about the destination matching the specified identifier.

The following shows an example CLI command:

```
aws iotwireless get-destination --name SidewalkDestination
```


Running this command returns the parameters of your destination.

```
{
  "Arn": "arn:aws:iotwireless:us-east-1:123456789012:Destination/
IoTWirelessDestination",
  "Name": "SidewalkDestination",
  "Expression": "IoTWirelessRule",
  "ExpressionType": "RuleName",
  "RoleArn": "arn:aws:iam::123456789012:role/IoTWirelessDestinationRole"
}
```

Update properties of your destination

Use the [UpdateDestination](#) API operation to update properties of your destination that you added to your account for AWS IoT Wireless. The following shows an example CLI command that updates the description property:

```
aws iotwireless update-destination --name SidewalkDestination \
  --description "Destination for messages processed using IoTWirelessRule"
```

List destinations in your AWS account

Use the [ListDestinations](#) API operation to list destinations in your AWS account that you added to AWS IoT Wireless. To filter the list to return only destinations for LoRaWAN and Sidewalk end devices, use the `WirelessDeviceType` parameter.

The following shows an example CLI command:

```
aws iotwireless list-destinations --wireless-device-type "Sidewalk"
```

Running this command returns a list of destinations that you added, including their Amazon Resource Name (ARN). To retrieve additional details about a specific destination, use the `GetDestination` API.

```
{
  "DestinationList": [
    {
      "Arn": "arn:aws:iotwireless:us-
east-1:123456789012:Destination/IoTWirelessDestination",
```

```

    "Name": "IoTWirelessDestination",
    "Expression": "IoTWirelessRule",
    "Description": "Destination for messages processed using IoTWirelessRule",
    "RoleArn": "arn:aws:iam::123456789012:role/IoTWirelessDestinationRole"
  },
  {
    "Arn": "arn:aws:iotwireless:us-east-1:123456789012:Destination/IoTWirelessDestination2",
    "Name": "IoTWirelessDestination2",
    "Expression": "IoTWirelessRule2",
    "RoleArn": "arn:aws:iam::123456789012:role/IoTWirelessDestinationRole"
  }
]
}

```

Delete destinations from your AWS account

To delete your destination, pass the name of the destination to be deleted as input to the [DeleteDestination](#) API operation. The following shows an example CLI command:

Warning

Deletion actions can't be undone. The destination will be permanently removed from your AWS account.

```
aws iotwireless delete-destination --name "SidewalkDestination"
```

This command doesn't produce any output. You can use the [GetDestination](#) API or the [ListDestinations](#) API operation to verify that the destination has been removed from your account.

AWS IoT Core for Amazon Sidewalk API operations for bulk provisioning

You can perform the following API operations for bulk provisioning your Sidewalk end devices:

- [StartWirelessDeviceImportTask](#) API or the [start-wireless-device-import-task](#) CLI

- [StartSingleWirelessDeviceImportTask](#) API or the [start-single-wireless-device-import-task](#) CLI
- [ListWirelessDeviceImportTasks](#) API or the [list-wireless-device-import-tasks](#) CLI
- [ListDevicesForWirelessDeviceImportTask](#) API or the [list-devices-for-wireless-device-import-task](#) CLI
- [GetWirelessDeviceImportTask](#) API or the [get-wireless-device-import-task](#) CLI
- [UpdateWirelessDeviceImportTask](#) API or the [update-wireless-device-import-task](#) CLI
- [DeleteWirelessDeviceImportTask](#) API or the [delete-wireless-device-import-task](#) CLI

The following sections show you how to get, list, update, and delete import tasks. For information about creating import tasks, see [Provisioning Sidewalk devices using import tasks](#).

Get information about your import task

You can use the [ListDevicesForWirelessDeviceImportTask](#) API operation to retrieve information about a particular import task and the onboarding status of devices in that task. As input to the API operation, specify the import task ID that you obtained from either the [StartWirelessDeviceImportTask](#) or [StartSingleWirelessDeviceImportTask](#) API operations. The API will then return information about the import task matching the specified identifier.

The following shows an example CLI command:

```
aws iotwireless list-devices-for-wireless-device-import-task --id e2a5995e-743b-41f2-a1e4-3ca6a5c5249f
```

Running this command returns your import task information and device onboarding status.

```
{
  "DestinationName": "SidewalkDestination",
  "ImportedWirelessDeviceList": [
    {
      "Sidewalk": {
        "OnboardingStatus": "ONBOARDED",
        "LastUpdateTime": "2023-02021T06:11:09.151Z",
```

```
    "SidewalkManufacturingSn":  
      "82B83C8B35E856F43CE9C3D59B418CC96B996071016DB1C3BE5901F0F3071A4A"  
    },  
    "Sidewalk": {  
      "OnboardingStatus": "PENDING",  
      "LastUpdateTime": "2023-02021T06:22:12.061Z",  
      "SidewalkManufacturingSn":  
        "12345ABCDE6789FABDESBDEF123456789012345FEABC0123679AFEB01234EF"  
    },  
  }  
]  
}
```

Get import task device summary

To get a count of summary information of the onboarding status of devices that you added to a particular import task, use the [GetWirelessDeviceImportTask](#) API operation. The following shows an example CLI command.

```
aws iotwireless get-wireless-device-import-task --Id "e2a5995e-743b-41f2-  
a1e4-3ca6a5c5249f"
```

The following code shows a sample response from the command.

```
{  
  "NumberOfFailedImportedDevices": 2,  
  "NumberOfOnboardedImportedDevices": 4,  
  "NumberOfPendingImportedDevices": 1  
}
```

Add devices to import task

Use the `UpdateWirelessDeviceImportTask` API operation to add devices to an existing import task that you added. You can use this API operation to add the serial numbers (SMSN) of devices that were not previously included the task that you created using the `StartWirelessDeviceImportTask` API operation.

To append devices to the import task, as part of the API request, specify a new CSV file in an Amazon S3 bucket that contains the serial numbers of devices to be added. The request will be accepted only if the onboarding process hasn't already started for devices that

are currently in the import task. If the onboarding process has already started, then the `UpdateWirelessDeviceImportTask` API request will fail.

If you still want to append devices to the import task, you can perform the `UpdateWirelessDeviceImportTask` API operation a second time. Before you perform this API operation, the first `UpdateWirelessDeviceImportTask` API request must have completed processing the CSV file in the S3 bucket.

Note

When you perform a `ListImportedWirelessDeviceTasks` API request, the S3 URL of the new CSV file specified using the `UpdateWirelessDeviceImportTask` API operation is currently not returned. Instead, the API operation returns the S3 URL of the request sent originally using the `StartWirelessDeviceImportTask` API request.

The following shows an example CLI command.

```
aws iotwireless update-wireless-device-import task \  
  --Id "e2a5995e-743b-41f2-a1e4-3ca6a5c5249f" \  
  --sidewalk '{"FileForCreateDevices": "s3://import_task_bucket/import_file3"}'
```

List import tasks in your AWS account

Use the `ListWirelessDeviceImportTasks` API or the `list-imported-wireless-device-tasks` CLI command to list import tasks in your AWS account. The following shows an example CLI command.

```
aws iotwireless list-wireless-device-import-tasks
```

Running this command returns a list of import tasks that you created. The list includes their Amazon S3 CSV files and the IAM role that was specified, the import task ID, and summary information of the device onboarding status.

```
{  
  "ImportWirelessDeviceTaskList": [  
    {  
      "FileForCreateDevices": "s3://import_task_bucket/import_file1",  
      "ImportTaskId": "e2a5995e-743b-41f2-a1e4-3ca6a5c5249f",
```

```
    "NumberOfFailedImportedDevices": 1,
    "NumberOfOnboardedImportedDevices": 3,
    "NumberOfPendingImportedDevices": 2,
    "Role": "arn:aws:iam::123456789012:role/service-role/ACF1zBEI",
    "TimeStamp": "1012202218:23:55"
  },
  {
    "FileForCreateDevices": "s3://import_task_bucket/import_file2",
    "ImportTaskId": "a1b234c5-67ef-21a2-a1b2-3cd4e5f6789a",
    "NumberOfFailedImportedDevices": 2,
    "NumberOfOnboardedImportedDevices": 4,
    "NumberOfPendingImportedDevices": 1,
    "Role": "arn:aws:iam::123456789012:role/service-role/CDEFaBC1",
    "TimeStamp": "1201202210:12:20"
  }
]
```

Delete import tasks from your AWS account

To delete an import task, pass the import task ID to the `DeleteWirelessDeviceImportTask` API operation or the `delete-wireless-device-import-task` CLI command.

Warning

Deletion actions can't be undone. The import task will be permanently removed from your AWS account.

When you perform the `DeleteWirelessDeviceImportTask` API request, a background process starts deleting the import task. When the request is in progress, the serial numbers (SMSN) of devices in the import tasks are in the process of deletion. Only after the deletion has completed, you'll be able to see this information using the `ListImportedWirelessDeviceTasks` or the `GetImportedWirelessDeviceTasks` API operations.

If an import task still contains devices that are waiting to be onboarded, the `DeleteWirelessDeviceImportTask` API request will be processed only after all the devices in the import task have either onboarded or failed to onboard. An import task expires after 90 days, and once the task has expired, it can be deleted from your account. However, devices that were onboarded successfully using the import task will not be deleted.

Note

If you attempt to create another import task that includes the serial number of a device that's pending deletion using the `DeleteWirelessDeviceImportTask` API request, then the `StartWirelessDeviceImportTask` API operation will return an error.

The following shows an example CLI command:

```
aws iotwireless delete-import-task --Id "e2a5995e-743b-41f2-a1e4-3ca6a5c5249f"
```

This command doesn't produce any output. After the task has been deleted, to verify that the import task has been removed from your account, you can use the `GetWirelessDeviceImportTask` API operation or the `ListWirelessDeviceImportTasks` API operation.

Creating AWS IoT Wireless resources with AWS CloudFormation

AWS IoT Wireless is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your AWS IoT Wireless resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

AWS IoT Wireless and AWS CloudFormation templates

To provision and configure resources for AWS IoT Wireless and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the *AWS CloudFormation User Guide*.

AWS IoT Wireless supports creating your wireless resources in AWS CloudFormation. For more information, including examples of JSON and YAML templates for your AWS IoT Wireless resources, see [AWS IoT Wireless resource type reference](#) in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

Quotas for AWS IoT Wireless

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is Region-specific. You can request increases for some quotas, and other quotas cannot be increased.

To view the quotas for AWS IoT Wireless, open the [Service Quotas console](#). In the navigation pane, choose **AWS services** and select **AWS IoT Wireless**.

To request a quota increase, see [Requesting a Quota Increase](#) in the *Service Quotas User Guide*. If the quota is not yet available in Service Quotas, use the [limit increase form](#).

AWS IoT Wireless has quotas for:

- AWS IoT Core for LoRaWAN quotas that apply to device data that is transmitted between the devices
- AWS IoT Wireless API operations that apply to both LoRaWAN and Sidewalk devices.

For more information, see [AWS IoT Core for LoRaWAN quotas](#) in the *AWS General Reference*.

Tagging your AWS IoT Wireless resources

To help you manage and organize your devices, gateways, destinations, and 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. AWS IoT Wireless doesn't have billing groups, and uses the same billing groups as AWS IoT Core. For more information, see [Billing groups](#) in the *AWS IoT Core documentation*.

Tag basics

When you've several AWS IoT Wireless resources of the same type, you can use tags to categorize your resources in different ways (for example, by purpose, owner, or environment). This helps you 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 a group of LoRaWAN devices for which the device firmware is being updated. To more easily manage your resources, we recommend that you create a consistent set of tag keys that meets your needs for each kind of resource.

You can search for and filter resources based on the tags you add or apply. You can also use tags to control access to your resources by using IAM policies and billing group tags to categorize and track your costs.

Create and manage tags

You can create and manage tags using the Tag Editor in the AWS Management Console, the AWS IoT Wireless, or the AWS CLI

Using the console

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](#).

Using the API or CLI

You can also use the API or CLI, and associate tags with wireless devices, gateways, profiles, and destinations when you create them by using the Tags field in the following commands:

- [AssociateAwsAccountWithPartnerAccount](#)
- [CreateDestination](#)
- [CreateDeviceProfile](#)
- [CreateFuotaTask](#)
- [CreateMulticastGroup](#)
- [CreateServiceProfile](#)
- [CreateWirelessGateway](#)
- [CreateWirelessGatewayTaskDefinition](#)
- [CreateWirelessDevice](#)
- [API_StartBulkAssociateWirelessDeviceWithMulticastGroup](#)

Update tags or list tags for resources

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

To specify what resources a user can create, modify, or use, you can apply tag-based resource-level permissions in the IAM policies you use for AWS IoT Wireless API actions. To control user access (permissions) based on a resource's tags, use the `Condition` element (also called the `Condition` block) with the following condition context keys and values in an IAM policy.

- 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 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:

- Can't give a resource the tag "env=prod" (in the example, see the line "aws:RequestTag/env" : "prod").
- Can't 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:CreateMulticastGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:CreateMulticastGroup",
        "iot:UpdateMulticastGroup",
        "iot:GetMulticastGroup",
        "iot:ListMulticastGroups"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateMulticastGroup",
        "iot:UpdateMulticastGroup",
        "iot:GetMulticastGroup",
        "iot:ListMulticastGroups"
      ],
      "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.

Document history for the AWS IoT Wireless User Guide

The following table describes the documentation releases for AWS IoT Wireless.

Change	Description	Date
Initial release	Initial release of the AWS IoT Wireless User Guide	December 31, 2020