# Table of Contents

What is AWS IoT SiteWise? ....................................................................................................................... 1
  How AWS IoT SiteWise works .................................................................................................................. 1
  Why use AWS IoT SiteWise? .................................................................................................................... 2
    Benefits .................................................................................................................................................. 2
  Use cases ................................................................................................................................................. 3
  Are you new to AWS IoT SiteWise? .......................................................................................................... 3
Related services .............................................................................................................................................. 4
  Concepts ................................................................................................................................................... 4
Key components ............................................................................................................................................. 7
  We want to hear from you ....................................................................................................................... 8
Getting started .............................................................................................................................................. 9
  Requirements ........................................................................................................................................... 9
  Setting up an AWS account ..................................................................................................................... 9
  Using the quick start demo ..................................................................................................................... 10
    Creating the AWS IoT SiteWise demo .................................................................................................. 11
    Deleting the AWS IoT SiteWise demo ................................................................................................12
Tutorials ....................................................................................................................................................... 13
  Calculating OEE ..................................................................................................................................... 13
    Prerequisites ........................................................................................................................................ 13
    How to calculate OEE ............................................................................................................................ 13
  Ingesting data to AWS IoT SiteWise from AWS IoT things ................................................................. 15
    Prerequisites ........................................................................................................................................ 16
    Creating an AWS IoT policy .................................................................................................................. 16
    Creating and configuring an AWS IoT thing ......................................................................................... 17
    Creating a device asset model ............................................................................................................. 20
    Creating a device fleet asset model .................................................................................................... 22
    Creating and configuring a device asset .............................................................................................. 23
    Creating and configuring a device fleet asset ..................................................................................... 24
    Creating a rule in AWS IoT Core to send data to device assets .......................................................... 26
    Running the device client script ......................................................................................................... 30
    Cleaning up resources ....................................................................................................................... 34
    Troubleshooting a rule ....................................................................................................................... 35
  Visualizing and sharing wind farm data in SiteWise Monitor ............................................................ 38
    Prerequisites ........................................................................................................................................ 39
    Creating a portal .................................................................................................................................... 39
    Signing in to a portal ............................................................................................................................. 42
    Creating a wind farm project ............................................................................................................... 44
    Creating dashboards to visualize wind farm data .............................................................................. 46
    Exploring the portal ............................................................................................................................. 52
    Cleaning up resources ....................................................................................................................... 53
  Publishing property value updates to Amazon DynamoDB .............................................................. 55
    Prerequisites ........................................................................................................................................ 56
    Configuring AWS IoT SiteWise to publish property value updates .................................................... 56
    Creating a rule in AWS IoT Core .......................................................................................................... 58
    Creating a DynamoDB table ............................................................................................................... 60
    Configuring the DynamoDB rule action .............................................................................................. 61
    Exploring data in DynamoDB ............................................................................................................. 61
    Cleaning up resources ....................................................................................................................... 63
    Troubleshooting a rule ....................................................................................................................... 65
  Ingesting data to AWS IoT SiteWise ................................................................................................... 69
    Using AWS IoT Core rules .................................................................................................................. 69
    Granting AWS IoT the required access ............................................................................................... 69
    Configuring the AWS IoT SiteWise rule action .................................................................................. 70
    Reducing costs with basic ingest ....................................................................................................... 75
### Table of Contents

- Monitoring data with alarms ................................................................. 220
- Using AWS IoT SiteWise gateways ........................................................ 79
- Configuring alarms on assets ............................................................... 247
- Defining alarms on asset models ......................................................... 223
- Alarm state properties ........................................................................ 221
- Alarm types ....................................................................................... 220
- Deleting assets and models ................................................................. 215
- Updating assets and models ................................................................. 210
- Associating assets ............................................................................. 207
- Mapping industrial data streams to asset properties ............................ 202
- Creating assets .................................................................................. 199
- Asset and model states ..................................................................... 140
  - Checking the status of an asset ....................................................... 140
  - Checking the status of an asset model ......................................... 142
- Creating asset models ....................................................................... 143
  - Creating an asset model (console) ................................................ 143
  - Creating an asset model (CLI) ....................................................... 144
  - Example asset models .................................................................. 145
  - Defining data properties .............................................................. 150
  - Defining relationships between assets ......................................... 198
- Creating assets .................................................................................. 199
  - Creating an asset (console) ........................................................... 200
  - Creating an asset (CLI) ................................................................. 201
  - Configuring a new asset ............................................................... 202
- Mapping industrial data streams to asset properties ............................ 202
  - Setting a property alias (console) ................................................ 203
  - Setting a property alias (CLI) ....................................................... 203
- Updating attribute values ................................................................... 204
  - Updating an attribute value (console) .......................................... 205
  - Updating an attribute value (CLI) ................................................. 205
- Associating assets ............................................................................. 207
  - Associating and disassociating assets (console) ............................ 207
  - Associating and disassociating assets (CLI) ................................. 209
- Updating assets and models ............................................................... 210
  - Updating assets ........................................................................... 211
  - Updating asset models ................................................................. 212
- Deleting assets and models ............................................................... 215
  - Deleting assets ............................................................................ 215
  - Deleting asset models ................................................................. 217
- Monitoring data with alarms ............................................................. 220
- Alarm types ....................................................................................... 220
- Alarm states ..................................................................................... 221
- Alarm state properties ....................................................................... 221
- Defining alarms on asset models ...................................................... 223
  - Defining AWS IoT Events alarms ............................................... 225
  - Defining external alarms ............................................................ 245
- Configuring alarms on assets ............................................................ 247
  - Configuring threshold values ....................................................... 247

---

**AWS IoT SiteWise User Guide**
What is AWS IoT SiteWise?

AWS IoT SiteWise is a managed service that lets you collect, model, analyze, and visualize data from industrial equipment at scale. With AWS IoT SiteWise Monitor, you can quickly create web applications for non-technical users to view and analyze your industrial data in real time. You can gain insights about your industrial operations by configuring and monitoring metrics such as mean time between failures and overall equipment effectiveness (OEE). With AWS IoT SiteWise Edge, you can view and process your data on your local devices.

The following diagram shows the basic architecture of AWS IoT SiteWise.

How AWS IoT SiteWise works

AWS IoT SiteWise provides an asset modeling framework that you can use to build representations of your industrial devices, processes, and facilities. With asset models, you define what raw data to consume and how to process your raw data into complex metrics. You can build and visualize assets and models for your industrial operation in the AWS IoT SiteWise console. You can configure asset models to collect and process data at the edge, or process data in the AWS Cloud.

You can upload industrial data to AWS IoT SiteWise in the following ways:

- Use AWS IoT SiteWise gateway software that runs on any platform that supports AWS IoT Greengrass, such as common industrial gateways or virtual servers. This software can read data directly from on-site servers over protocols such as the OPC-UA protocol. You can connect up to 100 OPC-UA servers to a single AWS IoT SiteWise gateway. You can also read data over the Modbus TCP and Ethernet/IP (EIP) protocol. For more information, see Ingesting data using a gateway (p. 87).

  **Note**
  You can add packs to your gateway to enable edge capability. With Sitewise Edge, you can read and process data directly on-site and send it to the AWS Cloud using a AWS IoT Greengrass stream. For more information, see Enabling edge data processing (p. 130).

- Use AWS IoT Core rules. If you have devices connected to AWS IoT Core sending MQTT messages, you can use the AWS IoT Core rules engine to route those messages to AWS IoT SiteWise. For more information, see Ingesting data using AWS IoT Core rules (p. 69).

- Use AWS IoT Events actions. You can configure the IoT SiteWise action in AWS IoT Events to send data to AWS IoT SiteWise when events occur. For more information, see Ingesting data from AWS IoT Events (p. 76).
• Use AWS IoT Greengrass stream manager. You can configure solutions on the edge that send high-volume IoT data to AWS IoT SiteWise. For more information, see Ingesting data using AWS IoT Greengrass stream manager (p. 76).
• Use the AWS IoT SiteWise API. Your applications at the edge or in the cloud can directly send data to AWS IoT SiteWise. For more information, see Ingesting data using the AWS IoT SiteWise API (p. 77).

You can set up SiteWise Monitor to create web applications for your non-technical employees to visualize your operations. With AWS SSO or IAM, you can configure unique logins and permissions for each employee to view specific subsets of an entire industrial operation. AWS IoT SiteWise provides an application guide for these employees to learn how to use SiteWise Monitor.

Why use AWS IoT SiteWise?

Benefits

Collect data consistently from all your sources

With AWS IoT SiteWise, you can gather data reliably from multiple facilities, structure it, and make it accessible and understandable – without developing additional software. You can query information and metrics about equipment or processes across multiple facilities, so it’s readily available for applications. AWS IoT SiteWise as the data collection, management and visualization capabilities you need built right in. So, you can invest your development resources on new applications that help you learn more from your data.

Identify issues quickly with remote monitoring

Assess the performance of your industrial equipment remotely, across locations, with AWS IoT SiteWise. Before, you had to dispatch a technician to diagnose a problem and then send another technician to fix the problem. Now, you can remotely diagnose a problem and only dispatch technicians when needed to fix issues. You can spend less time coordinating on-site diagnostic activities and let your engineers focus on what they do best: understanding your operations and designing better systems.

Improve cross-facility processes with a central data source

Visibility across industrial facilities allows you to streamline operations, as well as identify gaps in production and waste. With AWS IoT SiteWise, you can create models of industrial processes and equipment across multiple facilities, and then automatically discover and visualize live and historical asset data through customizable charts and dashboards. Through SiteWise Monitor, you have the ability to launch a web application with your asset data in minutes and give industrial engineers the visibility to react to issues or identify differences across facilities. SiteWise Monitor makes it easy to create a centralized, authoritative source of information to better understand your operations, improve processes, and reduce waste across your entire organization.

Process and monitor data on-premises for shop floor applications

AWS IoT SiteWise includes software (in preview), Sitewise Edge, that runs on-premises, securely connecting to and reading data from equipment or local historian databases. Once you have modeled your equipment and environment in the cloud, Sitewise Edge uses the same models locally to maintain consistency across both cloud and on-premise environments, reducing duplication, effort, and development costs. You can choose where to use and store your data across multiple locations such as keeping data on-premises for data residency requirements or for use by local edge applications. You can also send data to AWS IoT SiteWise or other AWS services in the cloud for additional storage and further analysis. With Sitewise Edge, you can deploy SiteWise Monitor web applications locally so users like process engineers can visualize equipment data in near real time on the shop floor. Sitewise Edge continues to operate even when connectivity to the cloud is not available, for on-premises scenarios.
Use cases

Manufacturing

With AWS IoT SiteWise, you can easily collect and use data from your equipment to identify and reduce inefficiencies and improve industrial operations. AWS IoT SiteWise helps you collect data from manufacturing lines and assembly robots, transfer it to the AWS Cloud, and structure performance metrics for your specific equipment and processes. You can use these metrics to understand the overall effectiveness of your operations and identify opportunities for innovation and improvement. You can also view your manufacturing process and identify equipment and process deficiencies, production gaps, or product defects.

Food and beverage

Food and beverage industry facilities handle a wide variety of food processing, including grinding grain to flour, butchering and packing meat, and assembling, cooking, and freezing microwaveable meals. These processing plants often span multiple locations with plant and equipment operators in a centralized location monitoring processes and equipment. For example, they may be monitoring refrigeration units, assessing ingredient handling and expiration, or they may be monitoring waste creation across facilities to ensure operational efficiency. With AWS IoT SiteWise, you can group sensor data streams from multiple locations by production line and facility so your process engineers can better understand and improve processes across facilities.

Energy and utilities

Companies often deploy their power-generation assets in remote areas, far from the technicians who are trained to fix the equipment. When an issue arises, the technicians receive a notification, travel to the site to diagnose the problem, and then make another trip to fix it. With AWS IoT SiteWise, you can resolve equipment issues easier and more efficiently. You can monitor asset performance remotely in real time and access historical equipment data from anywhere to pinpoint potential problems, dispatch the right resources, and both prevent and fix issues faster.

Are you new to AWS IoT SiteWise?

If you’re a first-time user of AWS IoT SiteWise, we recommend that you read about the components and concepts of AWS IoT SiteWise and set up the AWS IoT SiteWise demo (p. 9).

• Key components of AWS IoT SiteWise (p. 7)
• AWS IoT SiteWise concepts (p. 4)

You can complete the following tutorials to explore certain features of AWS IoT SiteWise:

• Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 15)
• Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 38)
• Publishing property value updates to Amazon DynamoDB (p. 55)

See the following topics to learn more about AWS IoT SiteWise:

• Ingesting data to AWS IoT SiteWise (p. 69)
• Modeling industrial assets (p. 139)
• Monitoring data with AWS IoT SiteWise Monitor (p. 257)
• Querying asset property values and aggregates (p. 285)
• Interacting with other AWS services (p. 291)
Related services

AWS IoT SiteWise integrates with the following AWS services so that you can develop a complete AWS IoT solution in the AWS Cloud:

- **AWS IoT Core** – Register and control AWS IoT devices that upload sensor data to AWS IoT SiteWise. You can configure AWS IoT SiteWise to publish notifications to the AWS IoT message broker, which lets you send AWS IoT SiteWise data to other AWS services. For more information, see the following topics:
  - Ingesting data using AWS IoT Core (p. 69)
  - Interacting with other AWS services (p. 291)
  - What is AWS IoT? in the AWS IoT Developer Guide

- **AWS IoT Greengrass** – Deploy edge devices that have AWS Cloud capabilities and can communicate with local AWS IoT devices. AWS IoT SiteWise gateways run on AWS IoT Greengrass to collect data from local servers and publish data to the AWS Cloud. For more information, see the following topics:
  - Ingesting data using a gateway (p. 87)
  - What is AWS IoT Greengrass? in the AWS IoT Greengrass Version 1 Developer Guide

- **AWS IoT Events** – Monitor your IoT data for process failures or changes in operation, and trigger actions when such events occur. For more information, see the following topics:
  - Monitoring data with alarms (p. 220)
  - Monitoring with alarms in the AWS IoT SiteWise Monitor Application Guide
  - What is AWS IoT Events? in the AWS IoT Events Developer Guide

- **AWS Single Sign-On (AWS SSO) and AWS Identity and Access Management (IAM)** – Create and manage user identities and permissions. SiteWise Monitor users sign in to web portals with AWS SSO or IAM credentials, and you can define which users have access to which assets' data. For more information, see the following topics:
  - Monitoring data with AWS IoT SiteWise Monitor (p. 257)
  - What is SiteWise Monitor? in the AWS IoT SiteWise Monitor Application Guide
  - What is IAM? in the IAM User Guide

AWS IoT SiteWise concepts

The following are the core concepts of AWS IoT SiteWise:

**Gateway**

A gateway resides on the customer premises to collect, process, and route data. A gateway connects to industrial data sources by using OPC-UA, Modbus TCP, or Ethernet/IP protocols to collect data when processing or routing the data to the AWS cloud. Gateways use packs to collect data, process data at the edge, and more. For more information about available packs, see the section called “Using packs” (p. 87).

You can create a gateway on any device or platform that can run AWS IoT Greengrass. The gateway software is made up of connectors that you can add to your AWS IoT Greengrass group. For more information, see Ingesting data using a gateway (p. 87).

**Packs**

Gateways use packs to decide how to collect, process, and route data. Currently, AWS IoT SiteWise supports the data collection pack and the data processing pack. For more information about the available packs for your gateway, see the section called “Using packs” (p. 87).
Data collection pack

Use the data collection pack so that your gateway can collect your industrial data and route it to the AWS destination of your choice. This pack is automatically added to your gateway and can't be removed.

Data processing pack

Use the data processing pack so that your gateway can communicate with edge-configured asset models and assets. Gateways with the data processing pack automatically periodically sync with all asset models in your AWS account that are configured for the edge.

Asset

When you ingest data into AWS IoT SiteWise from your industrial equipment, your devices, equipment, and processes are each represented as assets. Each asset has data associated with it. For example, a piece of equipment might have a serial number, a location, a make and model, and an install date. It might also have time series values for availability, performance, quality, temperature, pressure, and so on. You can organize assets into hierarchies, where assets have access to the data stored in its child assets. For more information, see Modeling industrial assets (p. 139).

Asset model

Every asset is created from an asset model. Asset models are declarative structures that standardize the format of your assets. Asset models enforce consistent information across multiple assets of the same type, so that you can process data in assets that represent groups of devices. In each asset model, you can define attributes (p. 5), time series inputs (measurements (p. 5)), time series transformations (transforms (p. 6)), time series aggregations (metrics (p. 6)), and asset hierarchies (p. 6). For more information, see Modeling industrial assets (p. 139).

You can control where your asset model's properties are processed by configuring your asset model for the edge. Use this feature to process and monitor asset data on your local devices.

Source destination

You can use a source destination to control where to send the incoming data from your source server. You can either send your data to AWS IoT SiteWise, or you can use a AWS IoT Greengrass stream to send your data to a different location. You can configure your AWS IoT Greengrass stream to send your data to an on-premises application, or to the AWS Cloud. You configure a source destination for each source server in your gateway.

Asset property

Asset properties are the structures within each asset that contain industrial data. Each property has a data type and can have a unit. A property can be an attribute (p. 5), a measurement (p. 5), a transform (p. 6), or a metric (p. 6). For more information, see Defining data properties (p. 150).

You can configure asset properties to compute at the edge. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

Attribute

Attributes are asset properties that represent information that generally doesn't change, such as device manufacturer or device location. Attributes can have default values. Each asset that you create from an asset model contains the default values of the attributes of that model. For more information, see Defining static data (attributes) (p. 151).

Measurement

Measurements are asset properties that represent a device or equipment's raw sensor time series data streams. For more information, see Defining data streams from equipment (measurements) (p. 152).
Transform

Transforms are asset properties that represent transformed time series data. Every transform has a mathematical expression (formula (p. 6)) that defines how to transform data points from one form to another. The transformed data points hold a one-to-one relationship with the input data points. For more information, see Transforming data (transforms) (p. 154).

Metric

Metrics are asset properties that represent aggregated time series data. Every metric has a mathematical expression (formula (p. 6)) that defines how to aggregate data points, and a time interval over which to compute that aggregation. Metrics output a single data point per given time interval. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).

Aggregate

Aggregates are basic metrics that AWS IoT SiteWise automatically computes for all time series data. For more information, see Querying asset property aggregates (p. 288).

Asset hierarchy

You can define asset hierarchies to create logical representations of your industrial operations. To create a hierarchy, you define a hierarchy definition in an asset model, and then you associate assets created from that model and the model specified in the hierarchy definition. Metrics in parent assets can aggregate data from child assets' properties, so you can calculate statistics that provide insight to your operation or a subset of your operation. For more information, see Defining relationships between assets (hierarchies) (p. 198).

Formula

Every transform (p. 6) and metric (p. 6) property has a formula that defines how that property transforms or aggregates data. Formulas consist of property inputs, operators, and functions offered by AWS IoT SiteWise. For more information, see Using formula expressions (p. 163).

Property alias

You can define aliases on asset properties to easily identify an asset property when you ingest or retrieve asset data. When you use a gateway (p. 4) to ingest data from servers, your property aliases must match the paths of your raw data streams. For more information, see Mapping industrial data streams to asset properties (p. 202).

Property notification

When you enable property notifications for an asset property, AWS IoT SiteWise publishes an MQTT message to AWS IoT Core each time that property receives a new value. The message payload contains information about that property value update. You can use property value notifications to create solutions that connect your industrial data in AWS IoT SiteWise with other AWS services. For more information, see Interacting with other AWS services (p. 291).

Portal

An SiteWise Monitor portal is a web application that you can use to visualize and share your AWS IoT SiteWise data. A portal has one or more administrators and contains zero or more projects.

Portal administrator

Each SiteWise Monitor portal has one or more portal administrators. Portal administrators use the portal to create projects that contain collections of assets and dashboards. The portal administrator then assigns assets and owners to each project. By controlling access to the project, portal administrators specify which assets that project owners and viewers can see.

Project

Each SiteWise Monitor portal contains a set of projects. Each project has a subset of your AWS IoT SiteWise assets associated with it. Project owners create one or more dashboards to provide a
consistent way to view the data associated with those assets. Project owners can invite viewers to the project to allow them to view the assets and dashboards in the project. The project is the basic unit of sharing within SiteWise Monitor. Project owners can invite users who were given access to the portal by the AWS administrator. A user must have access to a portal before a project in that portal can be shared with that user.

**Project owner**

Each SiteWise Monitor project has owners. Project owners create visualizations in the form of dashboards to represent operational data in a consistent manner. When dashboards are ready to share, the project owner can invite viewers to the project. Project owners can also assign other owners to the project. Project owners can configure thresholds and notification settings for alarms.

**Project viewer**

Each SiteWise Monitor project has viewers. Project viewers can connect to the portal to view the dashboards that project owners created. In each dashboard, project viewers can adjust the time range to better understand operational data. Project viewers can only view dashboards in the projects to which they have access. Project viewers can acknowledge and snooze alarms.

**Dashboard**

Each project contains a set of dashboards. Dashboards provide a set of visualizations for the values of a set of assets. Project owners create the dashboards and the visualizations that it contains. When a project owner is ready to share the set of dashboards, the owner can invite viewers to the project, which gives them access to all dashboards in the project. If you want a different set of viewers for different dashboards, you must divide the dashboards between projects. When viewers look at dashboards, they can adjust the time range.

**Visualization**

In each dashboard, project owners decide how to display the properties and alarms of the assets associated with the project. Availability might be represented as a line chart, while other values might be displayed as bar charts or key performance indicators (KPIs). Alarms are best displayed as status grids and status timelines. Project owners customize each visualization to provide the best understanding of the data for that asset.

### Key components of AWS IoT SiteWise

The following are the core components of AWS IoT SiteWise:

- **Gateway software** – The gateway software comprises several AWS IoT Greengrass connectors provided by AWS IoT SiteWise. These connectors run on any platform that supports AWS IoT Greengrass (version 1.10.2 or later). The gateway software polls data streams over the OPC-UA, Modbus TCP, or Ethernet/IP protocol and uploads the data to AWS IoT SiteWise over a secured internet connection. The gateway software also caches data locally in case internet connectivity is interrupted. To learn how to configure the connectors in your gateway, see [Ingesting data using a gateway](#)

  **Note**
  
  These connectors don't support AWS IoT Greengrass version 2.

  With the data processing pack enabled, the gateway software can process data locally and then send the data to the AWS Cloud. For more information about processing data locally, see the section called “Enabling edge data processing” (p. 130).

- **Local gateway application** – An application that you can use to monitor your local edge-configured gateways, asset models, and assets on your local device. For more information about processing data locally, see the section called “Enabling edge data processing” (p. 130).

- **Resources** – Asset models and monitor dashboards that are configured for the edge.
• **Gateway management** – Tools that the AWS IoT SiteWise console provides so that you can configure your edge gateways and update existing gateway configurations. To learn how to configure gateways and sources, see Adding the gateway to AWS IoT SiteWise (p. 100).

• **Asset modeling** – Tools provided in the AWS IoT SiteWise console to model assets, processes, and facilities. You can create multiple assets from asset models and uploaded data streams. You can define assets as children of other assets to represent processes and entire facilities, and you can define transforms and metrics that AWS IoT SiteWise computes from the uploaded data streams. AWS IoT SiteWise also automatically computes a set of basic aggregates for your data. To learn how to create models and assets, see Modeling industrial assets (p. 139). To learn how to query aggregated asset property values, see Querying asset property aggregates (p. 288).

• **AWS IoT SiteWise Monitor** – A data visualization and monitoring web application that you and others can access from any device. You can configure portals and dashboards with unique subsets of your industrial operation for your non-technical employees to gain insights from. To learn how to configure SiteWise Monitor, see Monitoring data with AWS IoT SiteWise Monitor (p. 257). To learn how to use the SiteWise Monitor web application, see the AWS IoT SiteWise Monitor Application Guide.

• **AWS IoT Core integration** – An AWS IoT Core rules action to ingest data to AWS IoT SiteWise and an option to publish asset property values to MQTT topics for integration with other AWS services. To learn how to ingest data from AWS IoT Core, see Ingesting data using AWS IoT Core rules (p. 69). To learn how to send asset property data to AWS IoT Core, see Interacting with other AWS services (p. 291).

• **CLI and SDKs** – An API that lets you create, manage, and update your assets, portals, and dashboards. You can use the API to read assets' current, historical, and aggregated property values. You can also use the API to upload data to asset properties. The AWS IoT SiteWise API is available in the AWS Command Line Interface (CLI) and in SDKs for various programming languages.

---

**We want to hear from you**

We welcome your feedback. To contact us, visit the AWS IoT SiteWise Discussion Forums or use one of the feedback links:

- **Provide feedback** at the bottom of the page.
- **Feedback** at the top right of the page.
getting started with AWS IoT SiteWise

With AWS IoT SiteWise, you can collect, organize, and analyze your data.

AWS IoT SiteWise provides a demo that you can use to explore the service without configuring a real data source. For more information, see Using the AWS IoT SiteWise demo (p. 10).

You can complete the following tutorials to explore certain features of AWS IoT SiteWise:

- Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 15)
- Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 38)
- Publishing property value updates to Amazon DynamoDB (p. 55)

See the following topics to learn more about AWS IoT SiteWise:

- Ingesting data to AWS IoT SiteWise (p. 69)
- Modeling industrial assets (p. 139)
- Enabling edge data processing (p. 130)
- Monitoring data with AWS IoT SiteWise Monitor (p. 257)
- Querying asset property values and aggregates (p. 285)
- Interacting with other AWS services (p. 291)

Requirements

You must have an AWS account to get started with AWS IoT SiteWise. If you don't have one, see Setting up an AWS account (p. 9).

Use a Region where AWS IoT SiteWise is available. For more information, see AWS IoT SiteWise endpoints and quotas. You can use the Region selector in the AWS Management Console to switch to one of these Regions.

Setting up 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

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

1. Sign in to the IAM console as the account owner by choosing Root user and entering your AWS account email address. On the next page, enter your password.
   
   **Note**
   
   We strongly recommend that you adhere to the best practice of using the Administrator IAM user that follows and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

2. In the navigation pane, choose Users and then choose Add user.

3. For User name, enter Administrator.

4. Select the check box next to AWS Management Console access. Then select Custom password, and then enter your new password in the text box.

5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to User must create a new password at next sign-in to allow the new user to reset their password after they sign in.

6. Choose Next: Permissions.

7. Under Set permissions, choose Add user to group.

8. Choose Create group.

9. In the Create group dialog box, for Group name enter Administrators.

10. Choose Filter policies, and then select AWS managed - job function to filter the table contents.

11. In the policy list, select the check box for AdministratorAccess. Then choose Create group.
   
   **Note**
   
   You must activate IAM user and role access to Billing before you can use the AdministratorAccess permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.

12. Back in the list of groups, select the check box for your new group. Choose Refresh if necessary to see the group in the list.

13. Choose Next: Tags.

14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM entities in the IAM User Guide.

15. Choose Next: Review to see the list of group memberships to be added to the new user. When you are ready to proceed, choose Create user.

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

**Using the AWS IoT SiteWise demo**

You can easily explore AWS IoT SiteWise by using the AWS IoT SiteWise demo. AWS IoT SiteWise provides the demo as an AWS CloudFormation template that you can deploy to create assets and generate sample data for up to a week.

**Important**

You will be charged for the resources that this demo creates and consumes.

**Topics**

- Creating the AWS IoT SiteWise demo (p. 11)
Creating the AWS IoT SiteWise demo

You can create the AWS IoT SiteWise demo from the AWS IoT SiteWise or AWS CloudFormation consoles.

Note
The demo AWS CloudFormation template creates three Lambda functions, one CloudWatch Events rule, and the IAM roles required for the demo. You might see these resources in your AWS account. We recommend that you keep these resources until you're done with the demo. If you delete the resources, the demo might stop working correctly.

To create the demo in the AWS IoT SiteWise console

1. Navigate to the AWS IoT SiteWise console and find the SiteWise demo in the upper-right corner of the page.
2. (Optional) Under SiteWise demo, change the Days to keep demo assets field to specify how many days to keep the demo before deleting it.
3. Choose Create demo.

The demo takes around 3 minutes to create. If the demo fails to create, your account might have insufficient permissions. Switch to an account that has administrative permissions, or use the following steps to delete the demo and try again:

   a. Choose Delete demo.

      The demo takes around 15 minutes to delete.

   b. If the demo doesn't delete, open the AWS CloudFormation console, choose the stack named IoTSiteWiseDemoAssets, and choose Delete in the upper-right corner.

   c. If the demo fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.

4. After the demo creates successfully, you can explore the demo assets and data in the AWS IoT SiteWise console.

To create the demo in AWS CloudFormation

1. Open the AWS CloudFormation template for the demo.
2. On the Create stack page, choose Next at the bottom of the page.
3. On the Specify stack details page, choose Next.
4. (Optional) On the Configure stack options page, change the DemoDurationDays field to specify how many days to keep the demo before deleting it.
5. Choose Next.
6. At the bottom of the page, select the check box that says I acknowledge that AWS CloudFormation might create IAM resources.
7. Choose Create stack.

The stack takes around 3 minutes to create. If the stack fails to create, your account might have insufficient permissions. Switch to an account that has administrative permissions, or use the following steps to delete the demo and try again:

   a. Choose Delete in the upper-right corner.

      The stack takes around 15 minutes to delete.

   b. If the demo fails to delete, choose Delete in the upper right corner again.
Deleting the AWS IoT SiteWise demo

The AWS IoT SiteWise demo deletes itself after a week, or the number of days you chose if you created the demo stack from the AWS CloudFormation console. You can delete the demo before if you're done using the demo resources. You can also delete the demo if the demo fails to create. Use the following steps to delete the demo manually.

To delete the AWS IoT SiteWise demo

1. Navigate to the AWS CloudFormation console.
2. Choose IoTSiteWiseDemoAssets from the list of Stacks.
3. Choose Delete.

   When you delete the stack, all of the resources created for the demo are deleted.
4. In the confirmation dialog, choose Delete stack.

   The stack takes around 15 minutes to delete. If the demo fails to delete, choose Delete in the upper-right corner again. If the demo fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.
AWS IoT SiteWise tutorials

You can use the following tutorials to work with AWS IoT SiteWise.

Topics
- Calculating OEE in AWS IoT SiteWise (p. 13)
- Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 15)
- Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 38)
- Publishing property value updates to Amazon DynamoDB (p. 55)

Calculating OEE in AWS IoT SiteWise

This tutorial provides an example of how to calculate overall equipment effectiveness (OEE) for a manufacturing process. As a result, your OEE calculations or formulas might differ from those shown here. In general, OEE is defined as Availability \* Quality \* Performance. To learn more about calculating OEE, see Overall equipment effectiveness on Wikipedia.

Prerequisites

To complete this tutorial, you must configure data ingestion for a device that has the following three data streams:

- Equipment_State – A numerical code that represents the state of the machine, such as idle, fault, planned stop, or normal operation.
- Good_Count – A data stream where each data point contains the number of successful operations since the last data point.
- Bad_Count – A data stream where each data point contains the number of unsuccessful operations since the last data point.

To configure data ingestion, see Ingesting data to AWS IoT SiteWise (p. 69). If you don’t have an available industrial operation, you can write a script that generates and uploads sample data through the AWS IoT SiteWise API.

How to calculate OEE

In this tutorial, you create an asset model that calculates OEE from three data input streams: Equipment_State, Good_Count, and Bad_Count. In this example, consider a generic packaging machine, such as one that’s used for packaging sugar, potato chips, or paint. In the AWS IoT SiteWise console, create an AWS IoT SiteWise asset model with the following measurements, transforms, and metrics. Then, you can create an asset to represent the packaging machine and observe how AWS IoT SiteWise calculates OEE.

Define the following measurements (p. 152) to represent the raw data streams from the packaging machine.
Measurements

- **Equipment_State** – A data stream (or measurement) that provides the current state of the packaging machine in numerical codes:
  - 1024 – The machine is idle.
  - 1020 – A fault, such as an error or delay.
  - 1000 – A planned stop.
  - 1111 – A normal operation.
- **Good_Count** – A data stream where each data point contains the number of successful operations since the last data point.
- **Bad_Count** – A data stream where each data point contains the number of unsuccessful operations since the last data point.

Using the **Equipment_State** measurement data stream and the codes it contains, define the following transforms (p. 154) (or derived measurements). Transforms have a one-to-one relationship with raw measurements.

**Transforms**

- **Idle** = eq(Equipment_State, 1024) – A transformed data stream that contains the machine's idle state.
- **Fault** = eq(Equipment_State, 1020) – A transformed data stream that contains the machine's fault state.
- **Stop** = eq(Equipment_State, 1000) – A transformed data stream that contains the machine's planned stop state.
- **Running** = eq(Equipment_State, 1111) – A transformed data stream that contains the machine's normal operational state.

Using the raw measurements and the transformed measurements, define the following metrics (p. 157) that aggregate machine data over specified time intervals. Choose the same time interval for each metric when you define the metrics in this section.

**Metrics**

- **Successes** = sum(Good_Count) – The number of successfully filled packages over the specified time interval.
- **Failures** = sum(Bad_Count) – The number of unsuccessfully filled packages over the specified time interval.
- **Idle_Time** = statetime(Idle) – The machine's total idle time (in seconds) per specified time interval.
- **Fault_Time** = statetime(Fault) – The machine's total fault time (in seconds) per specified time interval.
- **Stop_Time** = statetime(Stop) – The machine's total planned stop time (in seconds) per specified time interval.
- **Run_Time** = statetime(Running) – The machine's total time (in seconds) running without issue per specified time interval.
- **Down_Time** = Idle_Time + Fault_Time + Stop_Time – The machine's total downtime (in seconds) over the specified time interval, calculated as the sum of the machine states other than Run_Time.
- **Availability** = Run_Time / (Run_Time + Down_Time) – The machine's uptime or percentage of scheduled time that the machine is available to operate over the specified time interval.
• **Quality** = \(\frac{\text{Successes}}{(\text{Successes} + \text{Failures})}\) – The machine's percentage of successfully filled packages over the specified time intervals.

• **Performance** = \(\frac{((\text{Successes} + \text{Failures}) / \text{Run\_Time})}{\text{Ideal\_Run\_Rate}}\) – The machine's performance over the specified time interval as a percentage out of the ideal run rate (in seconds) for your process.

For example, your **Ideal\_Run\_Rate** might be 60 packages per minute (1 package per second). If your **Ideal\_Run\_Rate** is per minute or per hour, you need to divide it by the appropriate unit conversion factor because **Run\_Time** is in seconds.

• **OEE** = Availability * Quality * Performance – The machine's overall equipment effectiveness over the specified time interval. This formula calculates OEE as a fraction out of 1.

### Ingesting data to AWS IoT SiteWise from AWS IoT things

You can easily ingest data to AWS IoT SiteWise from a fleet of AWS IoT things by using device shadows. Device shadows are JSON objects that store current state information for an AWS IoT device. For more information, see Device shadow service in the AWS IoT Developer Guide.

After you complete this tutorial, you can set up an operation in AWS IoT SiteWise based on AWS IoT things. By using AWS IoT things, you can also easily integrate your operation with other useful features of AWS IoT. For example, you can configure AWS IoT features to do the following tasks:

- Configure additional rules to stream data to AWS IoT Events, Amazon DynamoDB, and other AWS services. For more information, see Rules in the AWS IoT Developer Guide.
- Index, search, and aggregate your device data with the AWS IoT Fleet Indexing service. For more information, see Fleet indexing service in the AWS IoT Developer Guide.
- Audit and secure your devices with AWS IoT Device Defender. For more information, see AWS IoT Device Defender in the AWS IoT Developer Guide.

In this tutorial, you learn how to ingest data from AWS IoT things' device shadows to assets in AWS IoT SiteWise. To do so, you create one or more AWS IoT things and run a script that updates each thing's device shadow with CPU and memory usage data. You use CPU and memory usage data in this tutorial to imitate realistic sensor data. Then, you create a rule with an AWS IoT SiteWise action that sends this data to an asset in AWS IoT SiteWise every time a thing's device shadow updates. For more information, see Ingesting data using AWS IoT Core rules (p. 69).

### Topics

- Prerequisites (p. 16)
- Creating an AWS IoT policy (p. 16)
- Creating and configuring an AWS IoT thing (p. 17)
- Creating a device asset model (p. 20)
- Creating a device fleet asset model (p. 22)
- Creating and configuring a device asset (p. 23)
- Creating and configuring a device fleet asset (p. 24)
- Creating a rule in AWS IoT Core to send data to device assets (p. 26)
- Running the device client script (p. 30)
- Cleaning up resources after the tutorial (p. 34)
Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don’t have one, see Setting up an AWS account (p. 9).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- Python 3 installed on your development computer or installed on the device that you want to register as an AWS IoT thing.

Creating an AWS IoT policy

In this procedure, you create an AWS IoT policy that allows your AWS IoT things to access the resources used in this tutorial.

To create an AWS IoT policy

1. Sign in to the AWS Management Console.
2. Review the AWS Regions (p. 9) where AWS IoT SiteWise is supported. Switch to one of these supported Regions, if necessary.
3. Navigate to the AWS IoT console. If a Get started button appears, choose it.
4. In the left navigation pane, choose Secure and then choose Policies.
5. If a You don’t have any policies yet dialog box appears, choose Create a policy. Otherwise, choose Create.
6. Enter a name for the AWS IoT policy (for example, SiteWiseTutorialDevicePolicy).
7. Under Add statements, choose Advanced mode to enter the following policy in JSON form. Replace region and account-id with your Region and account ID, such as us-east-1 and 123456789012.

```json
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": "iot:Connect",
    },
    {
        "Effect": "Allow",
        "Action": "iot:Publish",
        "Resource": [
        ]
    },
    {
        "Effect": "Allow",
        "Action": "iot:Subscribe",
    }
    ]
}
```
Creating and configuring an AWS IoT thing

In this procedure, you create and configure an AWS IoT thing. You can register your development computer as an AWS IoT thing to easily complete this tutorial. When you apply this tutorial's concepts to a real-world application, you can create and configure AWS IoT things on any device that can run an AWS

This policy allows your AWS IoT things to connect and interact with device shadows through MQTT messages. To interact with device shadows, your AWS IoT things publish and receive MQTT messages on topics that start with $aws/things/thing-name/shadow/. This policy uses a thing policy variable ${iot:Connection.Thing.ThingName}, which substitutes the connected thing's name in each topic. The iot:Connect statement limits which things can connect, so the thing policy variable can only substitute to names that start with SiteWiseTutorialDevice.

For more information, see Thing policy variables in the AWS IoT Developer Guide.

Note
This policy applies to things whose names start with SiteWiseTutorialDevice. To use a different name for your things, you must update the policy accordingly.

8. Choose Create.
IoT SDK, including AWS IoT Greengrass and FreeRTOS. For more information, see AWS IoT SDKs in the AWS IoT Developer Guide.

To create and configure an AWS IoT thing

1. Open a command line and run the following command to create a directory for this tutorial.

   ```bash
   mkdir iot-sitewise-rule-tutorial
   cd iot-sitewise-rule-tutorial
   ```

2. Run the following command to create a directory for your thing's certificates.

   ```bash
   mkdir device1
   ```

   If you're creating additional things, increment the number in the directory name accordingly to keep track of which certificates belong to which thing.

3. Navigate to the AWS IoT console.

4. In the left navigation pane, choose Manage and then choose Things.

5. If a You don't have any things yet dialog box appears, choose Create a thing. Otherwise, choose Create.

6. On the Creating AWS IoT things page, choose Create a single thing.

7. On the Add your device to the device registry page, enter a name for your AWS IoT thing (for example, SiteWiseTutorialDevice1) and then choose Next. If you're creating additional things, increment the number in the thing name accordingly.

   **Important**
   The thing name must match the name used in the policy that you created earlier in this tutorial. Otherwise, your device can't connect to AWS IoT.

8. On the Add a certificate for your thing page, choose Create certificate. Certificates enable AWS IoT to securely identify your devices.
9. Choose the **Download** links to download your thing's certificate, public key, and private key. Save all three files to the directory that you created for your thing's certificates (for example, `iot-sitewise-rule-tutorial/device1`).

   **Important**
   This is the only time that you can download your thing's certificate and keys, which you need for your device to successfully connect to AWS IoT.

10. Choose the root CA **Download** link to open a documentation page where you choose and download a root CA certificate. Save the root CA certificate to the `iot-sitewise-rule-tutorial`. We recommend downloading Amazon Root CA 1.

11. Choose **Activate**.

12. Choose **Attach a policy**.

13. On the **Add a policy for your thing page**, choose the policy that you created earlier in this tutorial (**SiteWiseTutorialDevicePolicy**), then choose **Register Thing**.
You have now registered an AWS IoT thing on your computer. You can now take one of the following next steps:

- Continue to the next section without creating additional AWS IoT things. You can complete this tutorial with only one thing.
- Repeat the steps in this section on another computer or device to create more AWS IoT things. For this tutorial, we recommend that you follow this option so that you can ingest unique CPU and memory usage data from multiple devices.
- Repeat the steps in this section on the same device (your computer) to create more AWS IoT things. Each AWS IoT thing receives similar CPU and memory usage data from your computer, so use this approach to demonstrate ingesting non-unique data from multiple devices.

Creating a device asset model

In this procedure, you create an asset model in AWS IoT SiteWise to represent your devices that stream CPU and memory usage data. Asset models enforce consistent information across multiple assets of the same type, so that you can process data in assets that represent groups of devices. For more information, see Modeling industrial assets (p. 139).

To create an asset model that represents a device

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Models.
3. Choose Create model.
4. Enter a name under Asset model information (for example, SiteWise Tutorial Device Model).
5. Under Measurement definitions, do the following:
   a. In Name, enter CPU Usage.
   b. In Unit, enter %.
   c. Leave the Data type as Double.

Measurement properties represent a device's raw data streams. For more information, see Defining data streams from equipment (measurements) (p. 152).
6. Choose Add measurement to add a second measurement property.
7. In the second row under Measurement definitions, do the following:
a. In Name, enter **Memory Usage**.

b. In Unit, enter \%.

c. Leave the Data type as Double.

---

8. Under **Metric definitions**, do the following:

a. In Name, enter **Average CPU Usage**.

b. In Formula, enter `avg(CPU Usage)`. Choose **CPU Usage** from the autocomplete list when it appears.

c. In Time interval, enter 5 minutes.

Metric properties define aggregation calculations that process all input data points over an interval and output a single data point per interval. This metric property calculates each device's average CPU usage every 5 minutes. For more information, see *Aggregating data from properties and other assets (metrics)* (p. 157).

9. Choose **Add metric** to add a second metric property.

10. In the second row under **Metric definitions**, do the following:

a. In Name, enter **Average Memory Usage**.

b. In Formula, enter `avg(Memory Usage)`. Choose **Memory Usage** from the autocomplete list when it appears.

c. In Time interval, enter 5 minutes.

This metric property calculates each device's average memory usage every 5 minutes.

---

11. (Optional) Add other additional metrics that you're interested in calculating per device. Some interesting functions include `min` and `max`. For more information, see *Using formula expressions* (p. 163). In the next section, you create a parent asset that can calculate metrics using data from your entire fleet of devices.

12. Choose **Create model**.
Creating a device fleet asset model

In this procedure, you create an asset model in AWS IoT SiteWise to represent your fleet of devices. In this asset model, you define a hierarchy, which lets you associate many device assets to a single fleet asset. Then, you define metrics in the fleet asset model that aggregate data from all associated device assets to gain insights about your fleet as a whole.

To create an asset model that represents a device fleet

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Models.
3. Choose Create model.
4. Enter a name under Asset model information (for example, SiteWise Tutorial Device Fleet Model).
5. Under Hierarchy definitions, do the following:
   a. Choose Add hierarchy.
   b. In Hierarchy name, enter Device.
   c. In Hierarchy model, choose your device asset model (SiteWise Tutorial Device Model).

A hierarchy defines a relationship between a parent (fleet) asset model and a child (device) asset model. Parent assets can access child assets’ property data. When you create assets later, you need to associate child assets to parent assets according to a hierarchy definition in the parent asset model. For more information, see Defining relationships between assets (hierarchies) (p. 198).

6. Under Metric definitions, do the following:
   a. In Name, enter Average CPU Usage.
   b. In Formula, enter \( \text{avg(Device | Average CPU Usage)} \). When the autocomplete list appears, choose Device to choose a hierarchy, then choose Average CPU Usage to choose the metric from the device asset that you created earlier.
   c. In Time interval, enter 5 minutes.

This metric property calculates the average CPU usage of all device assets associated to a fleet asset through the Device hierarchy.

7. Choose Add metric to add a second metric property.
8. In the second row under Metric definitions, do the following:
   a. In Name, enter Average Memory Usage.
   b. In Formula, enter \( \text{avg(Device | Average Memory Usage)} \). When the autocomplete list appears, choose Device to choose a hierarchy, then choose Average Memory Usage to choose the metric from the device asset that you created earlier.
   c. In Time interval, enter 5 minutes.

This metric property calculates the average memory usage of all device assets associated to a fleet asset through the Device hierarchy.
9. (Optional) Add other additional metrics that you're interested in calculating across your fleet of devices.

10. Choose **Create model**.

---

### Creating and configuring a device asset

In this procedure, you create a device asset from your device asset model. Then, you define property aliases for each measurement property. A property alias is a string that uniquely identifies an asset property. You can later use these aliases, rather than asset ID and property ID, to identify a property to which to upload data. For more information, see [Mapping industrial data streams to asset properties](p. 202).

#### To create a device asset and define property aliases

1. Navigate to the [AWS IoT SiteWise console](https://aws.amazon.com/iot-sitewise/).
2. In the left navigation pane, choose **Assets**.
3. Choose **Create asset**.
4. In **Asset model**, choose your device asset model, **SiteWise Tutorial Device Model**.
5. In **Name**, enter **SiteWise Tutorial Device 1**.
6. Choose **Create asset**.

7. For your new device asset, choose **Edit**.
8. Under CPU Usage, enter /tutorial/device/SiteWiseTutorialDevice1/cpu as the property alias. You include the AWS IoT thing's name in the property alias, so that you can ingest data from all of your devices using a single AWS IoT rule.

9. Under Memory Usage, enter /tutorial/device/SiteWiseTutorialDevice1/memory as the property alias.

10. Choose Save asset.

11. If you created multiple AWS IoT things earlier, repeat steps 3 through 10 for each device, and increment the number in the asset name and property aliases accordingly. For example, the second device asset's name should be SiteWise Tutorial Device 2, and its property aliases should be /tutorial/device/SiteWiseTutorialDevice2/cpu, and /tutorial/device/SiteWiseTutorialDevice2/memory.

Creating and configuring a device fleet asset

In this procedure, you create a device fleet asset from your device fleet asset model. Then, you associate your device assets to the fleet asset so that the fleet asset's metric properties can aggregate data from many devices.

To create a device fleet asset and associate device assets

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Assets.
3. Choose Create asset.
4. In Asset model, choose your device fleet asset model, SiteWise Tutorial Device Fleet Model.
5. In Name, enter SiteWise Tutorial Device Fleet 1.
6. Choose Create asset.
7. For your new device fleet asset, choose **Edit**.

8. Under **Assets associated to this asset**, choose **Add associated asset**.

9. Under **Hierarchy**, choose **Device**. This hierarchy identifies the hierarchical relationship between device and device fleet assets. You defined this hierarchy in the device fleet asset model earlier in this tutorial.

10. Under **Asset**, choose your device asset, **SiteWise Tutorial Device 1**.

11. If you created multiple device assets earlier, repeat steps 8 through 10 for each device asset that you created.

12. Choose **Save asset**.

You should now see your device assets organized as a hierarchy.
Creating a rule in AWS IoT Core to send data to device assets

In this procedure, you create a rule in AWS IoT Core that parses device shadow notification messages and sends data to your device assets in AWS IoT SiteWise. Each time your device's shadow updates, AWS IoT sends an MQTT message. You can create a rule that takes action when device shadows change based on the MQTT message. In this case, you want to process the update message to extract the property values and send them to your device assets in AWS IoT SiteWise.

To create a rule with an AWS IoT SiteWise action

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose Act and then choose Rules.
3. If a You don't have any rules yet dialog box appears, choose Create a rule. Otherwise, choose Create.
4. Enter a name and description for your rule.
5. Enter the following rule query statement.

```sql
SELECT * FROM 'aws/things/*/shadow/update/accepted' WHERE startsWith(topic(3), "SiteWiseTutorialDevice")
```
This rule query statement works because the device shadow service publishes shadow updates to `@aws/things/thingName/shadow/update/accepted`. For more information about device shadows, see "Device shadow service" in the "AWS IoT Developer Guide".

In the WHERE clause, this rule query statement uses the `topic(3)` function to get the thing name from the third segment of the topic. Then, the statement filters out devices that have names that don't match those of the tutorial devices. For more information about AWS IoT SQL, see "AWS IoT SQL reference" in the "AWS IoT Developer Guide".

6. Under Set one or more actions, choose Add action.

7. On the Select an action page, choose Send message data to asset properties in AWS IoT SiteWise to create an AWS IoT SiteWise rule action.

8. Choose Configure action at the bottom of the page.

9. On the Configure action page, complete the following steps to set up the AWS IoT SiteWise rule action:

   a. Choose By property alias.
b. In **Property alias**, enter `/tutorial/device/${topic(3)}/cpu`.

The `$...` syntax is a substitution template. AWS IoT evaluates the contents within the braces. This substitution template pulls the thing name from the topic to create an alias unique to each thing. For more information, see Substitution templates in the *AWS IoT Developer Guide*.

**Note**

Because an expression in a substitution template is evaluated separately from the SELECT statement, you can't use a substitution template to reference an alias created using an AS clause. You can reference only information present in the original payload, in addition to supported functions and operators.

c. In **Entry ID**, enter `$(concat(topic(3), "-cpu-", floor(state.reported.timestamp)))`.

Entry IDs uniquely identify each value entry attempt. If an entry returns an error, you can find the entry ID in the error output to troubleshoot the issue. The substitution template in this entry ID combines the thing name and the device's reported timestamp. For example, the resulting entry ID might look like `SiteWiseTutorialDevice1-cpu-1579808494`.

d. In **Time in seconds**, enter `$(floor(state.reported.timestamp))`.
This substitution template calculates the time in seconds from the device's reported timestamp. In this tutorial, devices report timestamp in seconds in Unix epoch time as a floating point number.

e. **In Offset in nanos**, enter \( \text{\$\{floor((state.reported.timestamp \% 1) * 1E9)\}} \).

This substitution template calculates the nanosecond offset from the time in seconds by converting the decimal portion of the device's reported timestamp.

**Note**
AWS IoT SiteWise requires that your data has a current timestamp in Unix epoch time. If your devices don't report time accurately, you can get the current time from the AWS IoT rules engine with `timestamp()`. This function reports time in milliseconds, so you must update your rule action's time parameters to the following values:

- **In Time in seconds**, enter \( \text{\$\{floor(timestamp() / 1E3)\}} \).
- **In Offset in nanos**, enter \( \text{\$\{(timestamp() \% 1E3) * 1E6\}} \).

f. In **Value**, enter \( \text{\$\{state.reported.cpu\}} \). In substitution templates, you use the . operator to retrieve a value from within a JSON structure.

g. In **Data type**, choose **Double**.

This data type must match the data type of the asset property you defined in the asset model.

h. Choose **Add entry** to add a new entry for the memory usage property, and complete the following steps again for that property:

i. **In Property alias**, enter `/tutorial/device/$\{topic(3)\}/memory`.

ii. **In Entry ID**, enter \( \text{\$\{concat(topic(3), "-memory-", floor(state.reported.timestamp))\}} \).

iii. **In Time in seconds**, enter \( \text{\$\{floor(state.reported.timestamp)\}} \).

iv. **In Offset in nanos**, enter \( \text{\$\{floor((state.reported.timestamp \% 1) * 1E9)\}} \).

v. **In Value**, enter \( \text{\$\{state.reported.memory\}} \).

vi. **In Data type**, choose **Double**.

i. Under **Root asset name**, choose **Select** to expand the list, then choose your device fleet asset (SiteWise Tutorial Device Fleet 1).

j. Under **Role**, choose **Create Role** to create an IAM role for this rule action. This role allows AWS IoT to push data to properties in your device fleet asset and its asset hierarchy.

k. Enter a role name and choose **Create role**.

![Create a new role](image)

A new IAM role will be created in your account. An inline policy will be attached to the role providing scoped-down permissions allowing AWS IoT to access resources on your behalf.

**Name**

SiteWiseTutorialDeviceRuleRole

l. Choose **Add action**.
10. (Optional) Configure an error action that you can use to troubleshoot your rule. For more information, see Troubleshooting a rule (p. 35).

11. Choose Create rule at the bottom of the page to finish creating the rule.

Running the device client script

Because you aren't using an actual device to report data, you run a script to update your AWS IoT thing's device shadow with CPU and memory usage to imitate real sensor data. To run the script, you must first install required Python packages. In this procedure, you install the required Python packages and then run the device client script.

To configure and run the device client script

1. Navigate to the AWS IoT console.
2. At the bottom of the left navigation pane, choose Settings.
3. Save your custom endpoint for use with the device client script. You use this endpoint to interact with your thing's shadows. This endpoint is unique to your account in the current Region.

   Your custom endpoint should look like the following example.

   ```
   identifier.iot.region.amazonaws.com
   ```

4. Open a command line and run the following command to navigate to the tutorial directory you created earlier.

   ```
   cd iot-sitewise-rule-tutorial
   ```

5. Run the following command to install the AWS IoT Device SDK for Python.

   ```
   pip3 install AWSIoTPythonSDK
   ```

   For more information, see AWS IoT Device SDK for Python in the AWS IoT Developer Guide

6. Run the following command to install psutil, a cross-platform process and system utilities library.

   ```
   pip3 install psutil
   ```

   For more information, see psutil in the Python Package Index.

7. Create a file called thing_performance.py in the iot-sitewise-rule-tutorial directory and then copy the following Python code into the file.

   ```python
   from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
   import json
   import psutil
   import argparse
   import logging
   import time

   # Configures the argument parser for this program.
   def configureParser():
       parser = argparse.ArgumentParser()
       parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
                        help="Your AWS IoT custom endpoint")
       parser.add_argument("-r", "--rootCA", action="store", required=True, dest="rootCAPath",
                        help="Root CA file path")
   ```
Running the device client script

```python
parser.add_argument("-c", "--cert", action="store", required=True,
dest="certificatePath",
    help="Certificate file path")
parser.add_argument("-k", "--key", action="store", required=True,
dest="privateKeyPath",
    help="Private key file path")
parser.add_argument("-p", "--port", action="store", dest="port", type=int,
default=8883,
    help="Port number override")
parser.add_argument("-n", "--thingName", action="store", required=True,
dest="thingName",
    help="Targeted thing name")
parser.add_argument("-d", "--requestDelay", action="store", dest="requestDelay",
type=float, default=1,
    help="Time between requests (in seconds")
parser.add_argument("-v", "--enableLogging", action="store_true",
dest="enableLogging",
    help="Enable logging for the AWS IoT Device SDK for Python")
return parser
```

# An MQTT shadow client that uploads device performance data to AWS IoT at a regular interval.
class PerformanceShadowClient:
    def __init__(self, thingName, host, port, rootCAPath, privateKeyPath,
                 certificatePath, requestDelay):
        self.thingName = thingName
        self.host = host
        self.port = port
        self.rootCAPath = rootCAPath
        self.privateKeyPath = privateKeyPath
        self.certificatePath = certificatePath
        self.requestDelay = requestDelay

    # Updates this thing's shadow with system performance data at a regular interval.
    def run(self):
        print("Connecting MQTT client for {}...".format(self.thingName))
        mqttClient = self.configureMQTTClient()
        mqttClient.connect()
        print("MQTT client for {} connected".format(self.thingName))
        deviceShadowHandler = mqttClient.createShadowHandlerWithName(self.thingName,
                                                                       True)
        print("Running performance shadow client for {}...".format(self.thingName))
        while True:
            performance = self.readPerformance()
            print("{}\nCPU:	{}% Memory:	{}%".format(json.dumps(performance),
                                                  performance["cpu"],
                                                  performance["memory"])
            payload = { "state": { "reported": performance } }
            deviceShadowHandler.shadowUpdate(json.dumps(payload),
                                               self.shadowUpdateCallback, 5)
            time.sleep(args.requestDelay)

    # Configures the MQTT shadow client for this thing.
    def configureMQTTClient(self):
        mqttClient = AWSIoTMQTTShadowClient(self.thingName)
        mqttClient.configureEndpoint(self.host, self.port)
        mqttClient.configureCredentials(self.rootCAPath, self.privateKeyPath,
                                         certificatePath)
        mqttClient.configureAutoReConnectBackoffTime(1, 32, 20)
        mqttClient.configureConnectDisconnectTimeout(10)
        mqttClient.configureMQTTOperationTimeout(5)
        return mqttClient
```

# Returns the local device’s CPU usage, memory usage, and timestamp.
```
def readPerformance(self):
    cpu = psutil.cpu_percent()
    memory = psutil.virtual_memory().percent
    timestamp = time.time()
    return { "cpu": cpu, "memory": memory, "timestamp": timestamp }

# Prints the result of a shadow update call.
def shadowUpdateCallback(self, payload, responseStatus, token):
    print("[{}]".format(self.thingName))
    print("Update request {} {}
".format(token, responseStatus))

# Configures debug logging for the AWS IoT Device SDK for Python.
def configureLogging():
    logger = logging.getLogger("AWSIoTPythonSDK.core")
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)

# Runs the performance shadow client with user arguments.
if __name__ == "__main__":
    parser = configureParser()
    args = parser.parse_args()
    if (args.enableLogging):
        configureLogging()
    thingClient = PerformanceShadowClient(args.thingName, args.host, args.port,
                                           args.rootCAPath, args.privateKeyPath,
                                           args.certificatePath, args.requestDelay)
    thingClient.run()

8. Run thing_performance.py from the command line with the following parameters:
   • -n, --thingName – Your thing name, such as SiteWiseTutorialDevice1.
   • -e, --endpoint – Your custom AWS IoT endpoint that you saved earlier in this procedure.
   • -r, --rootCA – The path to your AWS IoT root CA certificate.
   • -c, --cert – The path to your AWS IoT thing certificate.
   • -k, --key – The path to your AWS IoT thing certificate private key.
   • -d, --requestDelay – (Optional) The time in seconds to wait between each device shadow update. Defaults to 1 second.
   • -v, --enableLogging – (Optional) If this parameter is present, the script prints debug messages from the AWS IoT Device SDK for Python.

Your command should look similar to the following example.

```
python3 thing_performance.py
   --thingName SiteWiseTutorialDevice1
   --endpoint identifier.iot.region.amazonaws.com
   --rootCA AmazonRootCA1.pem
   --cert device1/thing-id-certificate.pem.crt
   --key device1/thing-id-private.pem.key
```

If you're running the script for additional AWS IoT things, update the thing name and certificate directory accordingly.
9. Try opening and closing programs on your device to see how the CPU and memory usages change. The script prints each CPU and memory usage reading. If the script uploads data to the device shadow service successfully, the script’s output should look like the following example.

```
[SiteWiseTutorialDevice1]
CPU: 24.6%
Memory: 85.2%
[SiteWiseTutorialDevice1]
Update request e6686e44-fca0-44db-aa48-3ca81726f3e3 accepted
```

10. Follow these steps to verify that the script is updating the device shadow:
   a. Navigate to the AWS IoT console.
   b. In the left navigation pane, choose Manage and then choose Things.
   c. Choose your thing, SiteWiseTutorialDevice1.
   d. In the left navigation pane on your thing’s page, choose Shadow.
   e. Verify that the Shadow state looks like the following example.

```
{
    "reported": {
        "cpu": 24.6,
        "memory": 85.2,
        "timestamp": 1579567542.2835066
    }
}
```
   f. If your thing’s shadow state is empty or doesn’t look like the previous example, check that the script is running and successfully connected to AWS IoT. If the script continues to time out when connecting to AWS IoT, check that your thing policy (p. 16) is configured according to this tutorial.

11. Follow these steps to verify that the rule action is sending data to AWS IoT SiteWise:
   a. Navigate to the AWS IoT SiteWise console.
   b. In the left navigation pane, choose Assets.
   c. Choose the arrow next to your device fleet asset (SiteWise Tutorial Device Fleet 1) to expand its asset hierarchy, and then choose your device asset (SiteWise Tutorial Device 1).
   d. Choose Measurements.
   e. Verify that the Latest value cells have values for the CPU Usage and Memory Usage properties.

   ![Table of Measurements](image)

   f. If the CPU Usage and Memory Usage properties don’t have the latest values, refresh the page. If values don’t appear after a few minutes, see Troubleshooting a rule (p. 35).

12. You have completed this tutorial. If you want to explore live visualizations of your data, you can configure a portal in AWS IoT SiteWise Monitor. For more information, see Monitoring data with AWS IoT SiteWise Monitor (p. 257). Otherwise, you can press CTRL+C in your command prompt to stop the device client script. It’s unlikely the Python program will send enough messages to incur charges, but it’s a best practice to stop the program when you’re done.
Cleaning up resources after the tutorial

After you complete the tutorial, clean up your resources to avoid incurring additional charges.

To delete hierarchical assets in AWS IoT SiteWise

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Assets**.
3. When you delete assets in AWS IoT SiteWise, you must first disassociate them.

   Complete the following steps to disassociate your device assets from your device fleet asset:

   a. Choose your device fleet asset (**SiteWise Tutorial Device Fleet 1**).
   b. Choose **Edit**.
   c. Under **Assets associated to this asset**, choose **Disassociate** for each device asset associated to this device fleet asset.
   d. Choose **Save asset**.

   You should now see your device assets no longer organized as a hierarchy.

4. Choose your device asset (**SiteWise Tutorial Device 1**).
5. Choose **Delete**.
6. In the confirmation dialog, enter **Delete** and then choose **Delete**.

   When you delete an asset, AWS IoT SiteWise discards all data from that asset's properties.

7. Repeat steps 4 through 6 for each device asset and the device fleet asset (**SiteWise Tutorial Device Fleet 1**).

To delete hierarchical asset models in AWS IoT SiteWise

1. Navigate to the AWS IoT SiteWise console.
2. If you haven't already, delete your device and device fleet assets. For more information, see the previous procedure (p. 34). You can't delete a model if you have assets that were created from that model.
3. In the left navigation pane, choose **Models**.
4. Choose your device fleet asset model (**SiteWise Tutorial Device Fleet Model**).
When you delete hierarchical asset models, you must delete the parent asset model first.

5. Choose **Delete**.

6. In the confirmation dialog, enter **Delete** and then choose **Delete**.

![Delete model SiteWise Tutorial Device Fleet Model dialog](image)

7. Repeat steps 4 through 6 for your device asset model (**SiteWise Tutorial Device Model**).

**To disable or delete a rule in AWS IoT Core**

1. Navigate to the **AWS IoT console**.
2. In the left navigation pane, choose **Act** and then choose **Rules**.
3. Choose the menu on your rule and choose **Disable** or **Delete**.

![Rules menu](image)

**Troubleshooting a rule**

Follow the steps in this procedure to troubleshoot your rule if the CPU and memory usage data isn't appearing in AWS IoT SiteWise as expected. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client. You can also configure logging to CloudWatch Logs to troubleshoot. For more information, see **Troubleshooting an AWS IoT SiteWise rule action** (p. 374).
To add a republish error action to a rule

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose Act and then choose Rules.
3. Choose the rule that you created earlier.
4. Under Error action, choose Add action.
5. Choose Republish a message to an AWS IoT topic.
6. Choose Configure action at the bottom of the page.
7. In Topic, enter sitewise/rule/tutorial/error. AWS IoT Core will republish error messages to this topic.
8. Choose **Select** to grant AWS IoT Core access to perform the error action.

9. Choose **Select** next to the role that you created earlier (for example, **SiteWiseTutorialDeviceRuleRole**).

10. Choose **Update Role** to add the additional permissions to the role.

11. Choose **Add action**.
12. Choose the back arrow in the upper left of the console to return to the AWS IoT console home.

After you set up the republish error action, you can view the error messages in the MQTT test client in AWS IoT Core.

In the following procedure, you subscribe to the error topic in the MQTT test client.

**To subscribe to the error action topic**

1. Navigate to the AWS IoT console.
2. In the left navigation page, choose Test to open the MQTT test client.
3. In the Subscription topic field, enter `sitewise/rule/tutorial/error` and choose Subscribe to topic.

   ![MQTT client](image)

   Connected as console-1570894263709-0

4. When error messages appear, view the `failures` array in any error message to diagnose issues. For more information about possible issues and how to resolve them, see Troubleshooting an AWS IoT SiteWise rule action (p. 374).

If errors don't appear, check that your rule is enabled and that you subscribed to the same topic that you configured in the republish error action. If errors still don't appear after you do that, check that the device script is running and updating the device's shadow successfully.

**Note**

You can also subscribe to your device's shadow update topic to view the payload that your AWS IoT SiteWise action parses. To do so, subscribe to the following topic.

```
#aws/things/+shadow/update/accepted
```

---

**Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor**

You can configure AWS IoT SiteWise Monitor to visualize and share your industrial data through managed web applications. Each web application is called a portal. Each portal contains projects, and you choose which data is available in each project.

You can then specify people in your company that can access each portal. Your users sign in to portals using AWS Single Sign-On accounts, so you can use your existing identity store or an AWS-managed one.

You, and your users with sufficient permissions, can create dashboards in each project to visualize your industrial data in meaningful ways. Then, your users can view these dashboards to quickly gain insights into your data and monitor your operation. You can configure administrative or read-only permissions...
Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don’t have one, see Setting up an AWS account (p. 9).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- A running AWS IoT SiteWise wind farm demo. When you set up the demo, it defines models and assets in AWS IoT SiteWise and streams data to them to represent a wind farm. For more information, see Using the AWS IoT SiteWise demo (p. 10).
- If you enabled AWS SSO in your account, sign in to your AWS Organizations management account. For more information, see AWS Organizations terminology and concepts. If you haven’t enabled AWS SSO, you will enable it in this tutorial and set your account as the management account.

If you can’t sign in to your AWS Organizations management account, you can partially complete the tutorial as long as you have an AWS SSO user in your organization. In this case, you can create the portal and dashboards, but you can’t create new AWS SSO users to assign to projects.

Creating a portal in SiteWise Monitor

In this procedure, you create a portal in SiteWise Monitor. Each portal is a managed web application that you and your users can sign in to with AWS Single Sign-On accounts. AWS SSO lets you use your company’s existing identity store or create one managed by AWS. Your company’s employees can sign in without creating separate AWS accounts.

To create a portal

1. Sign in to the AWS IoT SiteWise console.
2. Review the AWS Regions (p. 9) where AWS IoT SiteWise is supported and switch Regions, if needed. You must run the AWS IoT SiteWise demo in the same Region.

3. In the left navigation pane, choose Portals.

4. Choose Create portal.

5. If you already enabled AWS SSO, skip to step 6. Otherwise, complete the following steps to enable AWS SSO:

   a. On the Enable AWS Single Sign-On (SSO) page, enter your Email address, First name, and Last name to create an AWS SSO user for yourself to be the portal administrator. Use an email address you can access so that you can receive an email to set a password for your new AWS SSO user.

   In a portal, the portal administrator creates projects and assigns users to projects. You can create more users later.

   b. Choose Create user.

6. On the Portal configuration page, complete the following steps:

   a. Enter a name for your portal, such as WindFarmPortal.

   b. (Optional) Enter a description for your portal. If you have multiple portals, use meaningful descriptions to keep track of what each portal contains.

   c. (Optional) Upload an image to display in the portal.

   d. Enter an email address that portal users can contact when they have an issue with the portal and need help from your company's AWS administrator to resolve it.

   e. Choose Create portal.

7. On the Invite administrators page, you can assign AWS SSO users to the portal as administrators. Portal administrators manage permissions and projects within a portal. On this page, do the following:

   a. Select a user to be the portal administrator. If you enabled AWS SSO earlier in this tutorial, select the user that you created.
b. (Optional) Choose **Send invite to selected users**. Your email client opens, and an invitation appears in the message body. You can customize the email before you send it to your portal administrators. You can also send the email to your portal administrators later. If you're trying SiteWise Monitor for the first time and will be the portal administrator, you don't need to email yourself.

c. Choose **Next**.

8. On the **Assign users** page, you can assign AWS SSO users to the portal. Portal administrators can later assign these users as project owners or viewers. Project owners can create dashboards in projects. Project viewers have read-only access to the projects that they're assigned. On this page, you can create AWS SSO users to add to the portal.

**Note**

If you aren't signed in to your AWS Organizations management account, you can't create AWS SSO users. Choose **Assign users** to create the portal without portal users, and then skip this step.

On this page, do the following:

a. Complete the following steps twice to create two AWS SSO users:

i. Choose **Create user** to open a dialog box where you enter details for the new user.

ii. Enter an **Email address**, **First name**, and **Last name** for the new user. AWS SSO sends the user an email for them to set their password. If you want to sign in to the portal as these users, choose an email address that you can access. Each email address must be unique. Your users sign in to the portal using their email address as their usernames.
 iii. Choose **Create user**.

 b. Select the two AWS SSO users that you created in the previous step.

 c. Choose **Assign users** to add these users to the portal.

 The portals page opens with your new portal listed.

**Signing in to a portal**

In this procedure, you sign in to your new portal using the AWS SSO user that you added to the portal.

**To sign in to a portal**

1. On the **Portals** page, choose your new portal's **Link** to open your portal in a new tab.
2. If you created your first AWS SSO user earlier in the tutorial, use the following steps to create a password for your user:
   a. Check your email for the subject line Invitation to join AWS Single Sign-On.
   b. Open that invitation email and choose Accept invitation.
   c. In the new window, set a password for your AWS SSO user.

   If you want to sign in later to the portal as the second and third AWS SSO users that you created earlier, you can also complete these steps to set passwords for those users.

   Note
   If you didn’t receive an email, you can generate a password for your user in the AWS SSO console. For more information, see Reset a user password in the AWS Single Sign-On User Guide.

3. Enter your AWS SSO Username and Password. If you created your AWS SSO user earlier in this tutorial, your Username is the email address of the portal administrator user that you created.

   All portal users, including the portal administrator, must sign in with their AWS SSO user credentials. These credentials are typically not the same credentials that you use to sign in to the AWS Management Console.
4. Choose Sign in.

Your portal opens.

Creating a wind farm project

In this procedure, you create a project in your portal. Projects are resources that define a set of permissions, assets, and dashboards, which you can configure to visualize asset data in that project. With projects, you define who has access to which subsets of your operation and how those subsets' data is visualized. You can assign portal users as owners or viewers of each project. Project owners can create dashboards to visualize data and share the project with other users. Project viewers can view dashboards but not edit them. For more information about roles in SiteWise Monitor, see SiteWise Monitor roles (p. 257).

To create a wind farm project

1. In the left navigation pane in your portal, choose the Assets tab. On the Assets page, you can explore all assets available in the portal and add assets to projects.

2. In the asset browser, choose Demo Wind Farm Asset. When you choose an asset, you can explore that asset's live and historical data. You can also press Shift to select multiple assets and compare their data side-by-side.

3. Choose Add asset to project in the upper left. Projects contain dashboards that your portal users can view to explore your data. Each project has access to a subset of your assets in AWS IoT SiteWise. When you add an asset to a project, all users with access to that project can also access data for that asset and its children.

   ![Add asset to project dialog box](image)

4. In the Add asset to project dialog box, choose Create new project, and then choose Next.
5. In the **Create new project** dialog box, enter a **Project name** and **Project description** for your project, and then choose **Add asset to project**.

Your new project's page opens.

6. On the project's page, you can add portal users as owners or viewers of this project.

   **Note**
   If you aren't signed in to your AWS Organizations management account, you might not have portal users to assign to this project, so you can skip this step.

   On this page, do the following:
   a. Under **Project owners**, choose **Add owners** or **Edit users**.
Creating dashboards to visualize wind farm data

In this procedure, you create dashboards to visualize the demo wind farm data. Dashboards contain customizable visualizations of your project's asset data. Each visualization can have a different type, such
as a line chart, bar chart, or KPI display. You can choose the visualization type that works best for your data. Project owners can edit dashboards, while project viewers can only view them to gain insights.

**To create a dashboard with visualizations**

1. On your new project's page, choose **Create dashboard** to create a dashboard and open its edit page.

   In a dashboard's edit page, you can drag asset properties from the asset hierarchy to the dashboard to create visualizations. Then, you can edit each visualization's title, legend titles, type, size, and location in the dashboard.

2. Enter a name your dashboard.

3. Drag **Total Average Power** from the **Demo Wind Farm Asset** to the dashboard to create a visualization.

4. Choose **Demo Turbine Asset 1** to show properties for that asset, and then drag **Wind Speed** to the dashboard to create a visualization for wind speed.
5. Add **Wind Speed** to the new wind speed visualization for each **Demo Turbine Asset 2**, **3**, and **4** (in that order).

Your **Wind Speed** visualization should look similar to the following screenshot.

6. Repeat steps 4 and 5 for the wind turbines’ **Torque (KiloNewton Meter)** properties to create a visualization for wind turbine torque.

7. Choose the visualization type icon for the **Torque (KiloNewton Meter)** visualization, and then choose the bar chart icon.
8. Repeat steps 4 and 5 for the wind turbines’ **Wind Direction** properties to create a visualization for wind direction.

9. Choose the visualization type icon for the **Wind Direction** visualization, and then choose the KPI chart icon (30%).
10. (Optional) Make other changes to each visualization's title, legend titles, type, size, and location as needed.

11. Choose **Save dashboard** in the upper right to save your dashboard.

   Your dashboard should look similar to the following screenshot.
12. (Optional) Create an additional dashboard for each wind turbine asset.

As a best practice, we recommend that you create a dashboard for each asset so that your project viewers can investigate any issues with each individual asset. You can only add up to 5 assets to each visualization, so you must create multiple dashboards for your hierarchical assets in many scenarios.

A dashboard for a demo wind turbine might look similar to the following screenshot.
13. (Optional) Change the timeline or select data points on a visualization to explore the data in your dashboard. For more information, see Viewing dashboards in the AWS IoT SiteWise Monitor Application Guide.

Exploring the portal

In this procedure, you can explore the portal as other SiteWise Monitor roles.

To explore the portal and finish the tutorial

1. (Optional) If you added other users to the project as owners or viewers, you can sign in to the portal as these users. This lets you explore the portal as a user with fewer permissions than a portal administrator.

   Important
   You're charged for each user that signs in to a portal. For more information, see AWS IoT SiteWise Pricing.

   To explore the portal as other users, do the following:

   a. Choose Log out in the bottom left of the portal to exit the web application.

   b. Choose Sign out in the upper right of the AWS SSO application portal to sign out of your AWS SSO user.
c. Sign in to the portal as the AWS SSO user that you assigned as a project owner or project viewer. For more information, see Signing in to a portal (p. 42).

2. You've completed the tutorial. When you finish exploring your demo wind farm in SiteWise Monitor, follow the next procedure to clean up your resources.

**Cleaning up resources after the tutorial**

After you complete the tutorial, you can clean up your resources. You aren't charged for SiteWise Monitor if users don't sign in to your portal, but you can delete your portal and AWS SSO users. Your demo wind farm assets are deleted at the end of the duration that you chose when you created the demo, or you can delete the demo manually. For more information, see Deleting the AWS IoT SiteWise demo (p. 12).

Use the following procedures to delete your portal and AWS SSO users.

**To delete a portal**

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Portals.
3. Choose your portal, **WindFarmPortal**, and then choose Delete.

   When you delete a portal or project, the assets associated to deleted projects aren't affected.

   ![Portals](image)

4. In the Delete portal dialog box, choose Remove administrators and users.

   ![Delete portal](image)

5. Enter **delete** to confirm deletion, and then choose Delete.
To delete AWS SSO users

1. Navigate to the AWS SSO console.
2. In the left navigation pane, choose Users.
3. Select the check box for each user to delete, and then choose Delete users.
4. In the Delete users dialog box, enter DELETE, and then choose Delete users.
You can store your data in Amazon DynamoDB to easily access historical asset data without needing to repeatedly query the AWS IoT SiteWise API, which returns paginated value histories. After you complete this tutorial, you can easily create custom software that consumes your asset data, such as a live map of wind speed and direction over an entire wind farm. If you're looking to monitor and visualize your data without implementing a custom software solution, see Monitoring data with AWS IoT SiteWise Monitor (p. 257).

In this tutorial, you build on the AWS IoT SiteWise demo that provides a sample set of data for a wind farm. You configure property value updates from the wind farm demo to send data, through AWS IoT Core rules, to a DynamoDB table that you create. When you enable property value updates, AWS IoT SiteWise sends your data to AWS IoT Core in MQTT messages. Then, you can define AWS IoT Core rules that perform actions, such as the DynamoDB action, depending on the contents of those messages. For more information, see Interacting with other AWS services (p. 291).

Topics
- Prerequisites (p. 56)
- Configuring AWS IoT SiteWise to publish property value updates (p. 56)
- Creating a rule in AWS IoT Core (p. 58)
- Creating a DynamoDB table (p. 60)
Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Setting up an AWS account (p. 9).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- A running AWS IoT SiteWise wind farm demo. When you set up the demo, it defines models and assets in AWS IoT SiteWise and streams data to them to represent a wind farm. For more information, see Using the AWS IoT SiteWise demo (p. 10).

Configuring AWS IoT SiteWise to publish property value updates

In this procedure, you enable property value notifications on your demo turbine assets’ Wind Speed properties. After you enable property value notifications, AWS IoT SiteWise publishes each value update in an MQTT message to AWS IoT Core.

To enable property value update notifications on asset properties

1. Sign in to the AWS IoT SiteWise console.
2. Review the AWS Regions (p. 9) where AWS IoT SiteWise is supported and switch AWS Regions, if necessary. Switch to a Region where you’re running the AWS IoT SiteWise demo.
3. In the left navigation pane, choose Assets.
4. Choose the arrow next to **Demo Wind Farm Asset** to expand the wind farm asset's hierarchy.

5. Choose a demo turbine and choose **Edit**.

6. Update the **Wind Speed** property's **Notification status** to **ENABLED**.

7. Choose **Save asset** at the bottom of the page.

8. Repeat steps 5 through 7 for each demo turbine asset.

9. Choose a demo turbine (for example, **Demo Turbine Asset 1**).

10. Choose **Measurements**.

11. Choose the copy icon next to the **Wind Speed** property to copy the notification topic to your clipboard. Save the notification topic to use later in this tutorial. You only need to record the notification topic from one turbine.

   The notification topic should look like the following example.

   ```
   $aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
   ```
Creating a rule in AWS IoT Core

In this procedure, you create a rule in AWS IoT Core that parses the property value notification messages and inserts data into a DynamoDB table. AWS IoT Core rules parse MQTT messages and perform actions based on the contents and topic of each message. You can create a rule with a DynamoDB action to insert data to a DynamoDB table that you create as part of this tutorial.

To create a rule with a DynamoDB action

1. Navigate to the AWS IoT console. If a Get started button appears, choose it.
2. In the left navigation pane, choose Act and then choose Rules.
3. If a You don't have any rules yet dialog box appears, choose Create a rule. Otherwise, choose Create.
4. Enter a name and description for the rule.
5. Find the notification topic that you saved earlier in this tutorial.

$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
Replace the asset ID (the ID after assets/) in the topic with a + to select the wind speed property for all demo wind turbine assets. The + topic filter accepts all nodes from a single level in a topic. Your topic should look the following example.

```
/aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
```

6. Enter the following rule query statement. Replace the topic in the FROM section with your notification topic.

```
SELECT
  payload.assetId AS asset,
  (SELECT VALUE (value.doubleValue) FROM payload.values) AS windspeed,
  timestamp() AS timestamp
FROM
  '$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
  properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE'
WHERE
  type = 'PropertyValueUpdate'
```

7. Under Set one or more actions, choose Add action.

8. On the Select an action page, choose Split message into multiple columns of a DynamoDB table (DynamoDBv2).

9. Choose Configure action at the bottom of the page.

10. On the Configure action page, choose Create a new resource.

    The DynamoDB console opens in a new tab. Keep the rule action tab open while you complete the following procedures.
Creating a DynamoDB table

In this procedure, you create a DynamoDB table to receive wind speed data from the rule action.

To create a DynamoDB table

1. In the DynamoDB console dashboard, choose Create table.
2. Enter a name for your table.

3. For Primary key, do the following:
   a. Enter timestamp as the partition key.
   b. Choose the Number type.
   c. Select the Add sort key check box.
   d. Enter asset as the sort key, and leave the default sort key type of String.
4. Choose Create.

   When the Table is being created notice disappears, your table is ready.
5. Return to the tab with the Configure action page. Keep the DynamoDB tab open while you complete the following procedures.
Configuring the DynamoDB rule action

In this procedure, you configure the DynamoDB rule action to insert data from property value updates to your new DynamoDB table.

To configure the DynamoDB rule action

1. On the **Configure action** page, refresh the **Table name** list, and choose your new DynamoDB table.

2. Choose **Create role** to create an IAM role that grants AWS IoT Core access to perform the rule action.

3. Enter a role name and choose **Create role**.

4. Choose **Add action**.

5. Choose **Create rule** at the bottom of the page to finish creating the rule.

Your demo asset data should start appearing in your DynamoDB table.

Exploring data in DynamoDB

In this procedure, you explore the demo assets' wind speed data in your new DynamoDB table.

To explore asset data in DynamoDB

1. Return to the tab with the DynamoDB table open.
2. In the table you created earlier, choose the **Items** tab to view the data in the table. Refresh the page if you don't see rows in the table. If rows don't appear after a few minutes, see **Troubleshooting a rule** (p. 65).

3. In a row in the table, choose the edit icon to expand the data.

4. Choose the arrow next to the **windspeed** structure to expand the list of wind speed data points. Each list reflects a batch of wind speed data points sent to AWS IoT SiteWise by the wind farm demo. You might want a different data format if you set up a rule action for your own use. For more information, see **Querying asset property notification messages** (p. 294).

Now that you completed the tutorial, you can disable or delete the rule and delete your DynamoDB table to avoid incurring additional charges. Follow the next procedure to clean up your resources.
Cleaning up resources after the tutorial

After you complete the tutorial, clean up your resources to avoid incurring additional charges. Your demo wind farm assets are deleted at the end of the duration that you chose when you created the demo, or you can delete the demo manually. For more information, see Deleting the AWS IoT SiteWise demo (p. 12).

Use the following procedures to disable property value update notifications (if you didn't delete the demo), disable or delete your AWS IoT rule, and delete your DynamoDB table.

To disable property value update notifications on asset properties

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Assets.
3. Choose the arrow next to Demo Wind Farm Asset to expand the wind farm asset's hierarchy.
4. Choose a demo turbine and choose Edit.
5. Update the Wind Speed property's Notification status to DISABLED.

6. Choose Save asset at the bottom of the page.
7. Repeat steps 4 through 6 for each demo turbine asset.

To disable or delete a rule in AWS IoT Core

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose Act and then choose Rules.
3. Choose the menu on your rule and choose Disable or Delete.

To delete a DynamoDB table

1. Navigate to the DynamoDB console.
2. In the left navigation pane, choose Tables.
3. Choose the table you created earlier, WindSpeedData.
4. Choose Delete table.
Troubleshooting a rule

Follow the steps in this procedure to troubleshoot your rule if the demo asset data isn't appearing in the DynamoDB table as expected. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client. You can also configure logging to CloudWatch Logs to troubleshoot. For more information, see Monitoring with CloudWatch Logs in the AWS IoT Developer Guide.

To add a republish error action to a rule

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose Act and then choose Rules.
3. Choose the rule that you created earlier.

5. In the **Delete table** dialog, choose **Delete**.
4. Under **Error action**, choose **Add action**.

5. Choose **Republish a message to an AWS IoT topic**.

6. Choose **Configure action** at the bottom of the page.

7. In **Topic**, enter **windspeed/error**. AWS IoT Core will republish error messages to this topic.
8. Choose **Select** to grant AWS IoT Core access to perform the error action using the role that you created earlier.

9. Choose **Select** next to your role.

10. Choose **Update Role** to add the additional permissions to the role.

11. Choose **Add action** to finish adding the error action.
12. Choose the back arrow in the upper left of the console to return to the AWS IoT Core console home.

After you set up the republish error action, you can view the error messages in the MQTT test client in AWS IoT Core.

In the following procedure, you subscribe to the error topic in the MQTT test client.

**To subscribe to the error action topic**

1. In the AWS IoT Core console’s left navigation page, choose **Test**.
2. In the **Subscription topic** field, enter `windspeed/error` and choose **Subscribe to topic**.

![MQTT client](image)

3. Watch for error messages to appear and explore the **failures** array in an error message to diagnose the following common issues:
   - Typos in the rule query statement
   - Insufficient role permissions

If errors don't appear, check that your rule is enabled and that you subscribed to the same topic that you configured in the republish error action. If errors still don't appear, check that your demo wind farm assets still exist and that you enabled notifications on the wind speed properties. If your demo assets expired and disappeared from AWS IoT SiteWise, you can create a new demo and update the rule query statement to reflect the updated asset model and property IDs.
Ingesting data to AWS IoT SiteWise

AWS IoT SiteWise consumes industrial data and matches data to assets that represent your industrial operations. You must create assets and asset models to receive data in AWS IoT SiteWise. You can configure your data sources before building assets, but AWS IoT SiteWise won't receive any data sent until you create assets and set asset property aliases to your data stream paths. For instructions to build your virtual industrial operation, see Modeling industrial assets (p. 139).

You can send industrial data to AWS IoT SiteWise using any of the following options:

- Use an AWS IoT SiteWise gateway (p. 79) to upload data from servers. The gateway serves as the intermediary between AWS IoT SiteWise and your data servers. AWS IoT SiteWise provides AWS IoT Greengrass connectors that you can deploy on any platform that can run AWS IoT Greengrass to set up a gateway. AWS IoT SiteWise supports linking with OPC-UA, Modbus TCP, and Ethernet/IP server protocols.
- Use AWS IoT Core rules (p. 69) to upload data from MQTT messages published by an AWS IoT thing or another AWS service.
- Use AWS IoT Events actions (p. 76) to upload data from AWS IoT Events when an event occurs.
- Use AWS IoT Greengrass stream manager (p. 76) to upload data from local data sources using an edge device.
- Use the AWS IoT SiteWise API (p. 77) to upload data from any other source.

Ingesting data using AWS IoT Core rules

You can send data to AWS IoT SiteWise from AWS IoT things and other AWS services by using rules in AWS IoT Core. Rules transform MQTT messages and perform actions to interact with AWS services. The AWS IoT SiteWise rule action forwards messages data to the BatchPutAssetPropertyValue operation from the AWS IoT SiteWise API. For more information, see Rules and AWS IoT SiteWise action in the AWS IoT Developer Guide.

You can follow a tutorial that walks through the steps required to set up a rule that ingests data from AWS IoT things through their device shadows. For more information, see Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 15).

You can also send data from AWS IoT SiteWise to other AWS services. For more information, see Interacting with other AWS services (p. 291).

Topics

- Granting AWS IoT the required access (p. 69)
- Configuring the AWS IoT SiteWise rule action (p. 70)
- Reducing costs with basic ingest (p. 75)
- Troubleshooting the AWS IoT SiteWise rule action (p. 76)

Granting AWS IoT the required access

You use IAM roles to control the AWS resources to which each rule has access. Before you create a rule, you must create an IAM role with a policy that allows access to the required AWS resource. AWS IoT assumes this role when running a rule.

If you create the rule action in the AWS IoT console, you can choose an root asset to create a role that has access to a selected asset hierarchy. For more information about how to manually define a role for a rule, see Granting AWS IoT the required access and Pass role permissions in the AWS IoT Developer Guide.
Configuring the AWS IoT SiteWise rule action

The AWS IoT SiteWise rule action sends data from the MQTT message that initiated the rule to asset properties in AWS IoT SiteWise. You can upload multiple data entries to different asset properties at the same time, so that you can send updates for all sensors of a device in one message. You can also upload multiple data points at once for each data entry.

**Note**
When you send data to AWS IoT SiteWise with the rule action, your data must meet all of the requirements of the `iotsitewise:BatchPutAssetPropertyValue` operation. For example, your data can't have a timestamp earlier than 7 days from current Unix epoch time. For more information, see Ingesting data with the AWS IoT SiteWise API.

For each data entry in the rule action, you identify an asset property and specify the timestamp, quality, and value of each data point for that asset property. The rule action expects strings for all parameters.

To identify an asset property in an entry, specify one of the following:

- The **Asset ID** (`assetId`) and **Property ID** (`propertyId`) of the asset property that you're sending data to. If you choose this option in the AWS IoT console, you can use a list to choose an asset model and property from AWS IoT SiteWise in the current AWS Region.
• The Property alias (propertyAlias), which is a data stream alias (for example, /company/windfarm/3/turbine/7/temperature). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 202).

For the timestamp in each entry, you can use the timestamp reported by your equipment or the timestamp provided by AWS IoT Core. The timestamp has two parameters:

• **Time in seconds** (timeInSeconds) – The Unix epoch time, in seconds, at which the sensor or equipment reported the data.

• **Offset in nanos** (offsetInNanos) – (Optional) The nanosecond offset from the time in seconds.

  **Important**
  If your timestamp is a string, has a decimal portion, or isn't in seconds, AWS IoT SiteWise rejects the request. You must convert the timestamp to seconds and nanosecond offset. Use features of the AWS IoT rules engine to convert the timestamp. For more information, see the following:

  • Getting timestamps for devices that don't report accurate time (p. 71)
  • Converting timestamps that are in string format (p. 71)

You can use substitution templates for several parameters in the action to perform calculations, invoke functions, and pull values from the message payload. For more information, see Substitution templates in the AWS IoT Developer Guide.

  **Note**
  Because an expression in a substitution template is evaluated separately from the SELECT statement, you can't use a substitution template to reference an alias created using an AS clause. You can reference only information present in the original payload, in addition to supported functions and operators.

**Topics**

• Getting timestamps for devices that don't report accurate time (p. 71)

• Converting timestamps that are in string format (p. 71)

• Converting nanosecond-precision timestamp strings (p. 72)

• Example rule configurations (p. 74)

**Getting timestamps for devices that don't report accurate time**

If your sensor or equipment doesn't report accurate time data, you can get the current Unix epoch time from the AWS IoT rules engine with `timestamp()`. This function outputs time in milliseconds, so you must convert the value to time in seconds and offset in nanoseconds. To do so, use the following conversions:

• For **Time in seconds** (timeInSeconds), use `floor(timestamp() / 1E3)` to convert the time from milliseconds to seconds.

• For **Offset in nanos** (offsetInNanos), use `((timestamp() % 1E3) * 1E6)` to calculate the nanosecond offset of the timestamp.

**Converting timestamps that are in string format**

If your sensor or equipment reports time data in string format (for example, `2020-03-03T14:57:14.699Z`), you can use `time_to_epoch(String, String)`. This function inputs the
timestamp and format pattern as parameters and outputs time in milliseconds. Then, you must convert the time to time in seconds and offset in nanoseconds. To do so, use the following conversions:

- For **Time in seconds** (timeInSeconds), use
  
  
  $\left\lfloor \frac{floor(time\_to\_epoch("2020-03-03T14:57:14.699Z", "yyyy-MM-dd'T'HH:mm:ss'Z'"))}{1E3}\right\rfloor$

  to convert the timestamp string to milliseconds, and then to seconds.

- For **Offset in nanos** (offsetInNanos), use
  
  $\left\{(time\_to\_epoch("2020-03-03T14:57:14.699Z", "yyyy-MM-dd'T'HH:mm:ss'Z'") \mod 1E3) \times 1E6\right\}$

  to calculate the nanosecond offset of the timestamp string.

**Note**

The `time_to_epoch` function supports up to millisecond-precision timestamp strings. To convert strings with microsecond or nanosecond precision, you can configure an AWS Lambda function that your rule calls to convert the timestamp into numerical values. For more information, see Converting nanosecond-precision timestamp strings (p. 72).

### Converting nanosecond-precision timestamp strings

If your device sends timestamp information in string format with nanosecond precision (for example, 2020-03-03T14:57:14.699728491Z), use the following procedure to configure your rule action. You can create an AWS Lambda function that converts the timestamp from a string into **Time in seconds** (timeInSeconds) and **Offset in nanos** (offsetInNanos). Then, you can use `aws_lambda(functionArn, inputJson)` in your rule action parameters to invoke that Lambda function and use the output in your rule.

**Note**

This section contains advanced instructions that assume that you’re familiar with how to create the following resources:

- Lambda functions. For more information, see Create a Lambda function with the console or Using Lambda with the AWS CLI in the AWS Lambda Developer Guide.
- AWS IoT rules with the AWS IoT SiteWise rule action. For more information, see Ingesting data using AWS IoT Core rules (p. 69).

To create an AWS IoT SiteWise rule action that parses timestamp strings

1. Create a Lambda function with the following properties:

   - **Function name** – Use a descriptive function name (for example, `ConvertNanosecondTimestampFromString`).
   - **Runtime** – Use a Python 3 runtime, such as Python 3.8 (`python3.8`).
   - **Permissions** – Create a role with basic Lambda permissions (`AWSLambdaBasicExecutionRole`).
   - **Layers** – Add the `AWSLambda-Python38-SciPy1x` layer for the Lambda function to use `numpy`.
   - **Function code** – Use the following function code, which consumes a string argument named `timestamp` and outputs `timeInSeconds` and `offsetInNanos` values for that timestamp.

```python
import json
import math
import numpy

# Converts a timestamp string into timeInSeconds and offsetInNanos in Unix epoch time.
# The input timestamp string can have up to nanosecond precision.
def lambda_handler(event, context):
    timestamp_str = event['timestamp']
```
# Parse the timestamp string as nanoseconds since Unix epoch.
nanoseconds = numpy.datetime64(timestamp_str, 'ns').item()
# Slice to avoid precision issues.
offset_in_nanos = int(str(nanoseconds)[-9:])
return {
    'timeInSeconds': time_in_seconds,
    'offsetInNanos': offset_in_nanos
}

This Lambda function inputs timestamp strings in ISO 8601 format using `datetime64` from NumPy.

**Note**
If your timestamp strings aren’t in ISO 8601 format, you can implement a solution with pandas that defines the timestamp format. For more information, see `pandas.to_datetime`.

2. When you configure the AWS IoT SiteWise action for your rule, use the following substitution templates for **Time in seconds** (`timeInSeconds`) and **Offset in nanos** (`offsetInNanos`). These substitution templates assume that your message payload contains the timestamp string in `timestamp`. The `aws_lambda` function consumes a JSON structure for its second parameter, so you can modify the below substitution templates if needed.

   • For **Time in seconds** (`timeInSeconds`), use the following substitution template.

   ```
   ${aws_lambda('arn:aws:lambda:region:account-id:function:ConvertNanosecondTimestampFromString', {'timestamp': timestamp}).timeInSeconds}
   ```

   • For **Offset in nanos** (`offsetInNanos`), use the following substitution template.

   ```
   ${aws_lambda('arn:aws:lambda:region:account-id:function:ConvertNanosecondTimestampFromString', {'timestamp': timestamp}).offsetInNanos}
   ```

For each parameter, replace **region** and **account-id** with your Region and AWS account ID. If you used a different name for your Lambda function, change that as well.

3. Grant AWS IoT permissions to invoke your function with the `lambda:InvokeFunction` permission. For more information, see `aws_lambda(FunctionArn, inputJson)`.

4. Test your rule (for example, use the AWS IoT MQTT test client) and verify that AWS IoT SiteWise receives the data that you send.

   If your rule doesn’t work as expected, see **Troubleshooting an AWS IoT SiteWise rule action** (p. 374).

**Note**
This solution invokes the Lambda function twice for each timestamp string. You can create another rule to reduce the number of Lambda function invocations if your rule handles multiple data points that have the same timestamp in each payload.

To do so, create a rule with a republish action that invokes the Lambda and publishes the original payload with the timestamp string converted to `timeInSeconds` and `offsetInNanos`. Then, create a rule with an AWS IoT SiteWise rule action to consume the converted payload. With this approach, you reduce the number of times that the rule invokes the Lambda but increase the number of AWS IoT rule actions run. Consider the pricing of each service if you apply this solution to your use case.
Example rule configurations

This section contains example rule configurations that you can use to create a rule with an AWS IoT SiteWise action.

Example Example rule action that uses property aliases as message topics

The following example creates a rule with an AWS IoT SiteWise action that uses the topic (through \( \text{topic()} \)) as the property alias to identify asset properties. You can use this example to define one rule for ingesting double-type data to all wind turbines in all wind farms. This example requires that you define property aliases on all turbine assets' properties. You would need to define a second, similar rule to ingest integer-type data.

```bash
aws iot create-topic-rule \
  --rule-name SiteWiseWindFarmRule \
  --topic-rule-payload file://sitewise-rule-payload.json
```

The example payload in `sitewise-rule-payload.json` contains the following content.

```json
{
  "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+' WHERE type = 'double'",
  "description": "Sends data to the wind turbine asset property with the same alias as the topic",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
    {
      "iotSiteWise": {
        "putAssetPropertyValueEntries": [
          {
            "propertyAlias": \${topic()}",
            "propertyValues": [
              {
                "timestamp": {
                  "timeInSeconds": \${timeInSeconds}
                },
                "value": {
                  "doubleValue": \${value}
                }
              }
            ]
          },
          "roleArn": "arn:aws:iam::account-id:role/MySiteWiseActionRole"
        }
      }
    }
  ]
}
```

With this rule action, you can send the following message to a wind turbine property alias (for example, /company/windfarm/3/turbine/7/temperature) as a topic to ingest data.

```json
{
  "type": "double",
  "value": "38.3",
  "timeInSeconds": "1581368533"
}
```
Example Example rule action that uses timestamp() to determine time

The following example creates a rule with an AWS IoT SiteWise action that identifies an asset property by IDs and uses `timestamp()` to determine the current time.

```bash
aws iot create-topic-rule \
  --rule-name SiteWiseAssetPropertyRule \n  --topic-rule-payload file://sitewise-rule-payload.json
```

The example payload in `sitewise-rule-payload.json` contains the following content.

```json
{
  "sql": "SELECT * FROM 'my/asset/property/topic'",
  "description": "Sends device data to an asset property",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
    {
      "iotSiteWise": {
        "putAssetPropertyValueEntries": [
          {
            "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "propertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
            "propertyValues": [
              {
                "timestamp": {
                  "timeInSeconds": "${floor(timestamp() / 1E3)}",
                  "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                },
                "value": {
                  "doubleValue": "${value}"
                }
              }
            ],
            "roleArn": "arn:aws:iam::<account-id>:role/MySiteWiseActionRole"
          }
        ]
      }
    }
  ]
}
```

With this rule action, you can send the following message to the `my/asset/property/topic` to ingest data.

```json
{
  "type": "double",
  "value": "38.3"
}
```

Reducing costs with basic ingest

AWS IoT Core provides a feature called Basic Ingest that you can use to send data through AWS IoT Core without incurring AWS IoT messaging costs. Basic Ingest optimizes data flow for high volume data ingestion workloads by removing the publish/subscribe message broker from the ingestion path. You can use Basic Ingest if you know which rules your messages should be routed to.

To use Basic Ingest, you send messages directly to a specific rule using a special topic, `$aws/rules/rule-name`. For example, to send a message to a rule named `SiteWiseWindFarmRule`, you send a message to the topic `$aws/rules/SiteWiseWindFarmRule`. 
If your rule action uses substitution templates that contain `topic(Decimal)`, you can pass the original topic at the end of the Basic Ingest special topic, such as `$aws/rules/rule-name/original-topic`. For example, to use Basic Ingest with the wind farm property alias example from the previous section, you can send messages to the following topic.

```plaintext
/aws/rules/SiteWiseWindFarmRule//company/windfarm/3/turbine/7/temperature
```

**Note**
The above example includes a second slash (`//`) because AWS IoT removes the Basic Ingest prefix (`$aws/rules/rule-name`) from the topic that's visible to the rule action. In this example, the rule receives the topic `/company/windfarm/3/turbine/7/temperature`.

For more information, see [Reducing messaging costs with basic ingest](https://docs.aws.amazon.com/iot-developer-guide/#messaging-basics) in the *AWS IoT Developer Guide*.

**Troubleshooting the AWS IoT SiteWise rule action**

To troubleshoot your AWS IoT SiteWise rule action in AWS IoT Core, you can configure CloudWatch Logs or you can configure a republish error action for your rule. For more information, see [Troubleshooting an AWS IoT SiteWise rule action](https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-rule-actions.html) (p. 374).

**Ingesting data from AWS IoT Events**

With AWS IoT Events, you can build complex event monitoring applications for your IoT fleet in the AWS Cloud. You can use the IoT SiteWise action in AWS IoT Events to send data to asset properties in AWS IoT SiteWise when an event occurs.

For more information, see the following topics in the *AWS IoT Events Developer Guide*:

- What is AWS IoT Events?
- AWS IoT Events actions
- IoT SiteWise action

**Ingesting data using AWS IoT Greengrass stream manager**

The AWS IoT Greengrass stream manager feature integrates with AWS IoT SiteWise to transfer data from local sources to the AWS Cloud. You can add a data destination by configuring a local source on the AWS IoT SiteWise console or you can use stream manager in your custom AWS IoT Greengrass solution to ingest data to AWS IoT SiteWise.

**Note**
To ingest data from OPC-UA, Modbus TCP, and Ethernet/IP sources, you can configure a gateway that runs on AWS IoT Greengrass. For more information, see [Ingesting data using a gateway](https://docs.aws.amazon.com/iot-greengrass/latest/developer-guide/gateway-data-ingestion.html) (p. 87).

For more information about how to configure a destination for local source data, see [Configuring data sources](https://docs.aws.amazon.com/iot-greengrass/latest/developer-guide/configuring-data-sources.html) (p. 102).

For more information about how to ingest data using stream manager in a custom AWS IoT Greengrass solution, see the following topics in the *AWS IoT Greengrass Version 1 Developer Guide*:

- What is AWS IoT Greengrass?
Ingesting data using the AWS IoT SiteWise API

You can use the AWS IoT SiteWise API to send timestamped industrial data to your assets’ attribute and measurement properties. The API accepts a payload that contains timestamp-quality-value (TQV) structures.

Use the BatchPutAssetPropertyValue operation to upload your data. With this operation, you can upload multiple data entries at a time, so that you can collect data from several devices and send it all in a single request.

**Important**
The BatchPutAssetPropertyValue operation is subject to quotas on the number of entries per request and the number of TQV data points per entry. AWS IoT SiteWise also rejects any data with a timestamp dated to more than 7 days in the past or more than 5 minutes in the future. For more information about these quotas, see BatchPutAssetPropertyValue in the AWS IoT SiteWise API Reference.

To identify an asset property, you can specify one of the following:

- The assetId and propertyId of the asset property that you are sending data to.
- The propertyAlias, which is a data stream alias (for example, /company/windfarm/3/turbine/7/temperature). To use this option, you must first set your asset property’s alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 202).

The following example demonstrates how to send a wind turbine’s temperature and rotations per minute (RPM) readings from a payload stored in a JSON file.

```bash
aws iotsitewise batch-put-asset-property-value --cli-input-json file://batch-put-payload.json
```

The example payload in batch-put-payload.json contains the following content.

```json
{
  "entries": [
  {
    "entryId": "unique entry ID",
    "propertyAlias": "/company/windfarm/3/turbine/7/temperature",
    "propertyValues": [
      {
        "value": {
          "integerValue": 38
        },
        "timestamp": {
          "timeInSeconds": 1575691200
        }
      }
    ]
  },
  {
    "entryId": "unique entry ID",
    "propertyAlias": "/company/windfarm/3/turbine/7/rpm",
    "propertyValues": [
      {
        "value": {
```
Each entry in the payload contains an entryId that you can define as any unique string. If any request entries fail, each error will contain the entryId of the corresponding request so that you know which requests to retry.

Each structure in the list of propertyValues is a timestamp-quality-value (TQV) structure that contains a value, a timestamp, and optionally a quality.

- **value** – A structure that contains one of the following fields, depending on the type of the property being set:
  - booleanValue
  - doubleValue
  - integerValue
  - stringValue
- **timestamp** – A structure that contains the current Unix epoch time in seconds, timeInSeconds. You can also set the offsetInNanos key in the timestamp structure if you have temporally precise data. AWS IoT SiteWise rejects any data points with timestamps older than 7 days in the past or newer than 5 minutes in the future.
- **quality** – (Optional) One of the following quality strings:
  - GOOD – (Default) The data isn’t affected by any issues.
  - BAD – The data is affected by an issue such as sensor failure.
  - UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.

For more information about how AWS IoT SiteWise handles data quality in computations, see Data quality in formula expressions (p. 197).
Using AWS IoT SiteWise gateways

A gateway serves as the intermediary between your servers and AWS IoT SiteWise. The gateway runs on AWS IoT Greengrass V2 that supports data collection and processing on premises. You can use AWS OpsHub to manage your gateways and monitor on-site operations.

You can also access SiteWise Monitor portals on your gateway devices. For more information, see Enabling your portal at the edge (p. 273).

**Note**
Gateways running on AWS IoT Greengrass V1 are available only if you started using this feature before July 29, 2021. Otherwise, you set up gateways running on AWS IoT Greengrass V2 (p. 79).

Setting up gateways (Greengrass V2)

AWS IoT SiteWise gateways run on AWS IoT Greengrass V2 as a Greengrass component that supports data collection and processing on premises. To configure a gateway that runs on AWS IoT Greengrass V2, you need to create a gateway in the AWS Cloud and run the gateway software to set up your gateway device.

**Topics**
- Requirements (p. 79)
- Create a gateway (p. 81)
- Installing the gateway software (p. 84)

**Requirements**

Gateway devices must meet the following requirements to install and run the gateway software.

- Supports AWS IoT Greengrass V2 Core software v2.3.0. For more information, see Requirements in the AWS IoT Greengrass Version 2 Developer Guide.
- One of the following supported platforms:
  - OS: Ubuntu 20.04 or 18.04
    - Architecture: x86_64 (AMD64)
  - OS: Red Hat Enterprise Linux (RHEL) 8
    - Architecture: x86_64 (AMD64)
  - OS: Amazon Linux 2
    - Architecture: x86_64 (AMD64)
- Minimum 1 GB RAM allocated to the gateway software.
- Minimum 10 GB disk space available for the gateway software.
- If you plan to process data at the edge with AWS IoT SiteWise, your gateway device must also meet the following requirements:
  - Has an x86 64 bit quad-core processor.
  - Has at least 16 GB of RAM.
  - Has at least 256 GB of free disk space.
  - Choose a gateway with sufficient disk, networking, and compute capacity for your workload.
The disk space required for caching data for intermittent internet connectivity depends on the following factors:

- Number of data streams uploaded
- Data points per data stream per second
- Size of each data point
- Communication speeds
- Expected network downtime

The compute capacity required to poll and upload data depends on the following factors:

- Number of data streams uploaded
- Data points per data stream per second

Configure your gateway device to make sure that the following ports are accessible:

- The gateway device must allow inbound traffic on port 443.
- The gateway device must allow outbound traffic on port 443 and 8883.
- The following ports are reserved for use by AWS IoT SiteWise: 80, 443, 3001, 8000, 8081, 8082, 8084, 8085, 8445, 8086, 9000, 9500, and 11080. Using a reserved port for traffic can result in a terminated connection.

You must have the following permissions to use AWS IoT SiteWise gateways:

- Uses the AWSServiceRoleForIoTSiteWise role that allows AWS IoT SiteWise to complete required actions on AWS IoT Greengrass’s and AWS IoT Core’s resources in your account. For more information, see Service-linked role permissions for AWS IoT SiteWise (p. 339).
- The IAM role for your gateway must allow you to use an AWS IoT SiteWise gateway on an AWS IoT Greengrass V2 device to process asset model data and asset data.

The role allows the following service to assume the role:

credentials.iot.amazonaws.com

Permissions details

The role must have the following permissions:

- ecr – Allows principals to download Sitewise Edge gateway docker containers from Amazon Elastic Container Registry (Amazon ECR).
- iotsitewise – Allows principals to retrieve asset model data and asset data at the edge.
- iot – Allows your AWS IoT Greengrass V2 devices to interact with AWS IoT.
- logs – Allows your AWS IoT Greengrass V2 devices to send logs to Amazon CloudWatch Logs.
- s3 – Allows your AWS IoT Greengrass V2 devices to download custom component artifacts from Amazon S3.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:GetAuthorizationToken"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
```
Create a gateway

You can use the AWS IoT SiteWise console to create a gateway. This topic contains the following steps:

Topics
- Step 1: Configure a gateway (p. 81)
- Step 2: Configure edge capabilities (p. 82)
- Step 3: Add data sources (p. 82)
- Step 4: Review and generate an installer (p. 84)

Step 1: Configure a gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Gateways.
3. Choose Create gateway.
4. Enter a name for your gateway or use the name generated by AWS IoT SiteWise.
5. For Greengrass core device, choose one of the following options:
   - **Default setup** - AWS automatically uses default settings to create a Greengrass core device in AWS IoT Greengrass V2.
     1. Enter a name for the Greengrass core device or use the name generated by AWS IoT SiteWise.
   - **Advanced setup** - Choose this option if you want to use an existing Greengrass core device or to create one manually.
     1. Choose a Greengrass core device or choose Create Greengrass core device to create one in the AWS IoT Greengrass V2 console. For more information, see Setting up AWS IoT Greengrass V2 core devices in the AWS IoT Greengrass Version 2 Developer Guide.
6. Choose Next.
Step 2: Configure edge capabilities

AWS IoT SiteWise provides the following packs that your gateway can use to determine how to collect and process your data. Select packs to enable edge capabilities for your gateway.

- **Data collection pack** enables your gateway to collect data from multiple OPC-UA servers, and then export the data from the edge to the AWS Cloud. By default, this pack is automatically enabled for your gateway. You can't disable this pack.

- **Data processing pack** enables your gateway to process your data at the edge. For example, you can use asset models to compute metrics and transforms. For more information about asset models and assets, see **Modeling industrial assets (p. 139)**.

  **Note**
  - AWS IoT SiteWise retains your edge data on your gateways up to 30 days. The retention of your data is dependent on your device and the available disk space.
  - 30 days after your gateway has been disconnected from the AWS Cloud, the data processing pack is automatically disabled.

1. (Optional) In the **Edge capabilities** pane, select **Data processing pack**.

2. (Optional) In the **Edge LDAP connection** pane, you can grant user groups in your corporate directory access to this gateway. The user groups can use the Lightweight Directory Access Protocol (LDAP) credentials to access the gateway. Then they can use the AWS OpsHub for AWS IoT SiteWise application, AWS IoT SiteWise APIs, or other tools to manage the gateway. For more information, see **Managing gateways (p. 134)**

  **Note**
  - You can also use the Linux credentials to access the gateway. For more information, see **Accessing your gateway using Linux credentials (p. 136)**.

  a. Select Enable.
  b. For **Provider name**, enter a name for your LDAP provider.
  c. For **Hostname or IP address**, enter the hostname or IP address of your gateway device.
  d. For **Port**, enter a port number.
  e. For **Base distinguished name (DN)**, enter a distinguished name (DN) for the base.

    The following attribute types are supported: commonName (CN), localityName (L), stateOrProvinceName (ST), organizationName (O), organizationalUnitName (OU), countryName (C), streetAddress (STREET), domainComponent (DC), and userid (UID).

  f. For **Admin group DN**, enter a DN.
  g. For **User group DN**, enter a DN.

3. Choose **Next**.

Step 3: Add data sources

Data sources are local servers or industrial equipment that are connected to gateways. You can add data sources so that your gateway can ingest data from the OPC-UA servers to AWS IoT SiteWise.

  **Note**
  - Gateways running on AWS IoT Greengrass V2 currently don't support Modbus TCP and Ethernet IP sources.

  **To add an OPC-UA source**

  1. Choose **Add data source**.
2. Enter a name for the source.

3. Enter the Local endpoint of the data source server. The endpoint can be the IP address or hostname. You may also add a port number to the local endpoint. For example, your local endpoint might look like opc.tcp://203.0.113.0:49320.

4. (Optional) For Node ID for selection, add node filters to limit which data streams are ingested to the AWS Cloud. By default, gateways use the root node of a server to ingest all data streams. To define node filters, you can use node IDs and the * and ** wildcard characters.

5. For Destinations, choose where the source data is sent.

   - AWS IoT SiteWise - Send data to AWS IoT SiteWise. Choose this option if you want to process data at the edge.
   - AWS IoT Greengrass stream manager - Use AWS IoT Greengrass stream manager to send data to the following AWS cloud destinations: channels in AWS IoT Analytics, streams in Amazon Kinesis Data Streams, asset properties in AWS IoT SiteWise, or objects in Amazon Simple Storage Service (Amazon S3). For more information, see Manage data streams on the AWS IoT Greengrass Core in AWS IoT Greengrass Version 2 Developer Guide.

   Enter a name for the AWS IoT Greengrass stream.

6. In the Advanced configuration pane, do the following:

   a. (Optional) Enter a Data stream prefix. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.

   b. Choose a Message security mode for connections and data in transit between your source server and your gateway. This field is the combination of the OPC-UA security policy and message security mode. You must choose the same security policy and message security mode that you specified for your OPC-UA server.

   c. If your source requires authentication, choose an AWS Secrets Manager secret from the Authentication configuration list. The gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your gateway's IoT SiteWise connector to use them for source authentication. For more information, see Configuring source authentication (p. 114).

      Tip
      Your data server might have an option named Allow anonymous login. If this option is Yes, then your source doesn't require authentication.

   d. For Property groups, choose Add new group.

   e. Enter a Name for the property group.

   f. For Properties:

      1. (Optional) For Node paths, add OPC-UA node filters to limit which OPC-UA paths are uploaded to AWS IoT SiteWise. You can use node filters to reduce your gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with /Server/. To define OPC-UA node filters, you can use node paths and the * and ** wildcard characters. For more information, see Using OPC-UA node filters (p. 120).

   g. For Group settings, do the following:

      1. For Scan mode, choose the mode that you want AWS IoT SiteWise to use to collect your data. For more information about scan mode, see the section called “Filtering data ingestion ranges with OPC-UA” (p. 119).

      2. For Scan rate, update the rate that want the gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your gateway.

      3. (Optional) Configure a Deadband setting for your source. This controls what data your source sends to your AWS IoT SiteWise, and what data it discards. For more information
Step 4: Review and generate an installer

In this step, you review the configuration of your gateway, and then do the following:

1. Choose one of the following operating system:
   - Amazon Linux
   - Red hat
   - Ubuntu
2. Choose Generate.
3. In the dialog box, choose Acknowledge.

AWS IoT SiteWise automatically generates an installer that you can use to configure your gateway device. Make sure that you save the installer file in a secure location. You will use the file later.

Installing the gateway software

This section shows you how to install the gateway software on your gateway device.

**Important**
Make sure that your gateway device connects to the Internet.

Topics
- Step 1: Copy the installer to your gateway device (p. 84)
- Step 2: Set up your environment (p. 85)
- Step 3: Install the gateway software (p. 86)

Step 1: Copy the installer to your gateway device

The following uses SSH to connect to your getway device. You can use a USB flash drive or other tools to transfer the installer file to your gateway device.

Connect to your gateway device using SSH

The following instructions explain how to connect to your gateway device using an SSH client.

Prerequisites

Before you connect to your device, complete the following prerequisites.

- Get the IP address and user name to connect to your device.
- Install an SSH client on your local computer as needed.

Your local computer might have an SSH client installed by default. You can verify this by typing `ssh` at the command line. If your computer doesn't recognize the command, you can install an SSH client.

- Recent versions of Windows Server 2019 and Windows 10 - OpenSSH is included as an installable component. For more information, see OpenSSH in Windows.
Installing the gateway software

- Earlier versions of Windows - Download and install OpenSSH. For more information, see Win32-OpenSSH.
- Linux and macOS X - Download and install OpenSSH. For more information, see https://www.openssh.com.

1. To connect to your device, run the following command in a terminal window on your computer.

   **Note**
   Replace *username* and *IP* with your user name and IP address.

   ```
   ssh username@IP
   ```

2. To transfer the installer file that AWS IoT SiteWise generated to your gateway device, run the following command.

   **Note**
   - Replace *path-to-saved-installer* with the path on your computer that you used to save the installer file and the name of the installer file.
   - Replace *IP-address* with the IP address of your gateway device.
   - Replace *directory-to-receive-installer* with the path on your gateway device that you use to receive the installer file.

   ```
   scp path-to-saved-installer.sh user-name@IP-address:directory-to-receive-installer
   ```

**Step 2: Set up your environment**

Follow the steps in this section to set up a Linux device to use as your AWS IoT SiteWise gateway device. These steps assume that you use a device with Ubuntu. If you use a different Linux distribution, consult the relevant documentation for your device.

**To set up a gateway device**

1. To set up a DNS compliant fully qualified hostname, run the following command in a terminal window on your gateway device. Replace *hostname* with a qualified hostname (for example, sitewise-gateway-rhel.amazon.com).

   ```
   sudo hostnamectl set-hostname hostname
   ```

   **Note**
   - If you use a registered domain, you can configure the IPv4 address for the hostname. In step 3, you must follow the second instruction.
   - If you use a registered domain, in step 3, you must follow the first option.

2. To verify the hostname, run the following command.

   ```
   hostname -f
   ```

   **Example output**

   ```
   sitewise-gateway-rhel-pdx.amazon.com
   ```
The output should be the hostname that you created in the previous step.

3. Do one of the following:

1. Add a DNS entry A record that resolves the IPv4 address for the hostname created in step 1. For more info about A record, see A record type in the Amazon Route 53 Developer Guide.
2. Set the `/etc/hosts` file on the gateway device and all user managed client machines.

Add the following message to the `/etc/hosts` file on the gateway device and all user-managed client machines.

**Note**
- Replace `IPv4-address` with your IPv4 address,
- Replace `simple-hostname` with the simple hostname.
- Replace `fully-qualified-hostname` with the hostname that you created in step 1.

```
IPv4-address  simple-hostname  fully-qualified-hostname
```

**Example**
```
127.0.0.0  sitewise-gateway  sitewise-gateway.amazonaws.com
```

**Note**
- You must use the configured hostname to ssh into the gateway device instead of the IPv4 address. Verify this before you run the installer.
- You can now use the configured hostname to sign in to the gateway on the AWS OpsHub for AWS IoT SiteWise application.

### Step 3: Install the gateway software

In the following procedures, run the commands in a terminal window on your gateway device.

1. Give the installer file the execute permission.

```
chmod +x path-to-installer.sh
```

2. Run the installer.

```
sudo ./path-to-installer.sh
```

### Setting up gateways (Greengrass V1)

**Note**
Gateways running on AWS IoT Greengrass V1 are available only if you started using this feature before July 29, 2021. Otherwise, you set up gateways running on AWS IoT Greengrass V2 (p. 79).

You can send industrial data to AWS IoT SiteWise using an AWS IoT SiteWise gateway to upload data from servers. The gateway serves as the intermediary between AWS IoT SiteWise and your data servers.
AWS IoT SiteWise provides AWS IoT Greengrass connectors that you can deploy on any platform that can run AWS IoT Greengrass to set up a gateway. AWS IoT SiteWise supports linking with OPC-UA, Modbus TCP, and Ethernet/IP server protocols.

Topics
- Ingesting data using a gateway (p. 87)
- Enabling edge data processing (p. 130)

Ingesting data using a gateway

An AWS IoT SiteWise gateway connects to data servers to deliver your industrial data streams to the AWS Cloud. AWS IoT SiteWise supports OPC-UA, Modbus TCP, and Ethernet/IP data server protocols. You can deploy the AWS IoT SiteWise provided AWS IoT Greengrass connectors on any device or platform that can run AWS IoT Greengrass.

Note
This documentation describes how to set up the latest version of the AWS IoT SiteWise connector, version 10. For more information and the connector version changelog, see the AWS IoT SiteWise connector in the AWS IoT Greengrass Version 1 Developer Guide.

Using packs

Gateways use different packs to determine how to collect and process your data.

Currently, the following packs are available:

- **Data collection pack** – Use this pack to collect your industrial data and route it to AWS Cloud destinations. By default, this pack is enabled automatically for your gateway.
- **Data processing pack** – Use this pack to enable gateway communication with edge-configured asset models and assets. You can use edge configuration to control what asset data to compute and process on-site. You can then send your data to AWS IoT SiteWise or other AWS services. For more information about the data processing pack, see the section called “Enabling edge data processing” (p. 130).
  
  Note
  You need v9 of the data collection pack to use the data processing pack on your gateway.

For more information about how to identify and resolve gateway issues, see Troubleshooting an AWS IoT SiteWise gateway (p. 372).

Topics
- Choosing a gateway platform (p. 87)
- Configuring a gateway (p. 88)
- Configuring data sources (p. 102)

Choosing a gateway platform

Choose an AWS IoT SiteWise gateway platform that best suits your industrial operation. You can configure a gateway on any platform that can run AWS IoT Greengrass. All gateway devices must meet the following requirements:

- Supports AWS IoT Greengrass Core software v1.10.2 or later. For more information, see Supported platforms and requirements in the AWS IoT Greengrass Version 1 Developer Guide.
- Has at least 1 GB of RAM.
Note
If you plan to process data at the edge with AWS IoT SiteWise, your device must meet more requirements. For more information, see the section called “Enabling edge data processing” (p. 130).

- Has at least 10 GB of free disk space.
- Supports a Java 8 virtual machine (JVM).

Choose a gateway with sufficient disk, networking, and compute capacity for your workload.

The disk space required for caching data for intermittent internet connectivity depends on the following factors:

- Number of data streams uploaded
- Data points per data stream per second
- Size of each data point
- Communication speeds
- Expected network downtime

The compute capacity required to poll and upload data depends on the following factors:

- Number of data streams uploaded
- Data points per data stream per second

Configuring a gateway

A gateway serves as the intermediary between your server and AWS IoT SiteWise. You can deploy the AWS IoT SiteWise gateway software on any platform that can run AWS IoT Greengrass. For more information, see Choosing a gateway platform (p. 87).

You can enable AWS IoT SiteWise to process data locally on your edge devices by using the data processing pack on your gateway. You do this when you add your gateway to AWS IoT SiteWise. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

To configure a gateway that runs on Amazon EC2, you can create the required dependencies from an AWS CloudFormation template. For more information, see Configuring gateway dependencies on Amazon Elastic Compute Cloud (p. 101).

Note
We recommend that you complete the following steps with someone who has IT administrative access to your local and corporate networks. These steps might require someone with knowledge of your server and the authority to configure firewall settings.

Topics
- Setting up the gateway environment (p. 89)
- Creating an IAM policy and role (p. 91)
- Configuring an AWS IoT Greengrass group (p. 96)
- Configuring the AWS IoT SiteWise connector (p. 98)
- Adding the gateway to AWS IoT SiteWise (p. 100)
- Configuring gateway dependencies on Amazon Elastic Compute Cloud (p. 101)
Setting up the gateway environment

In this procedure, you install AWS IoT Greengrass and configure your gateway to use with AWS IoT SiteWise.

**Note**

This section includes instructions to install packages using the `apt` command. This is applicable to systems running Ubuntu or similar. If you aren't using a similar system, consult the documentation for your distribution and use the recommended package installer.

**To set up the gateway**

1. As appropriate, modify the BIOS settings of the gateway as follows.
   a. Ensure that the gateway automatically restarts after a potential power failure, if applicable.
   b. Ensure that the gateway won't hibernate or sleep, if applicable.
2. Ensure that the gateway connects to the internet.
3. (Optional) To use the gateway without the mouse, keyboard, and monitor, do the following steps to set up `ssh` on the gateway:
   a. If you haven't already installed the SSH package, run the following command.
      
      ```
      sudo apt install ssh
      ```
   b. Run the following command.
      
      ```
      service ssh status
      ```
   c. Search for `Active: active (running)` in the output to confirm that the SSH server is running,
   d. Press Q to exit.

Run the following command to use SSH to connect to the gateway from another computer. Replace `username` with the user login and `IP` with the IP address of the gateway.

```
ssh username@IP
```

You can use the `-p port-number` argument to connect to a port other than the default port 22.

4. Download and install AWS IoT Greengrass Core software v1.10.2 or later, and create an AWS IoT Greengrass group for your gateway. To do so, follow the instructions in Getting started with AWS IoT Greengrass in the AWS IoT Greengrass Developer Guide.

We recommend that you run the AWS IoT Greengrass device setup script to quickly get started. If you want to review AWS IoT Greengrass requirements and processes more closely, you can walk through the steps in Module 1 and Module 2 to set up AWS IoT Greengrass.

**Important**

Review the AWS Regions (p. 9) where AWS IoT SiteWise is supported. When you choose a Region for AWS IoT Greengrass, make sure that the Region also supports AWS IoT SiteWise. Otherwise, you can't connect your gateway to AWS IoT SiteWise.

Before you continue to the next step, you should have AWS IoT Greengrass Core software installed on your gateway.

5. Run the following commands to install Java 8.

```
sudo apt update
```
sudo apt install openjdk-8-jre

The AWS IoT SiteWise gateway software that you install later in this guide uses a Java 8 runtime.

6. Run the following command to verify that Java installed successfully.

```bash
java -version
```

7. The AWS IoT Greengrass Core software assumes a `java8` directory. Run the following command to link your Java installation to that `java8` directory.

```bash
sudo ln -s /usr/bin/java /usr/bin/java8
```

8. Run the following command to create a `/var/sitewise` data directory and give the `ggc_user` permissions for that directory. AWS IoT SiteWise stores data in this directory. You created the `ggc_user` when you set up AWS IoT Greengrass earlier in this procedure.

```bash
sudo mkdir /var/sitewise
sudo chown ggc_user /var/sitewise
sudo chmod 700 /var/sitewise
```

The `/var/sitewise` is the default directory that AWS IoT SiteWise uses. You can customize the directory path (for example, replace `/var/sitewise` with `/var/custom/path/`), but doing so requires extra steps after the AWS IoT SiteWise gateway is created. For more information, see step 6 in Configuring the AWS IoT SiteWise connector (p. 98).

9. If needed, ask your IT administrator to add the following endpoints and ports to your local network allow list:

- Ports: 443, 8443, and 8883

  **Important**
  You can configure AWS IoT Greengrass Core to use only port 443 for all network communications. For more information, see Connect on port 443 or through a network proxy in the AWS IoT Greengrass Developer Guide.

- The IP address of your gateway (port 443). To obtain the IP address, run the `ip address` or `ifconfig` command and note the `inet` value (for example, 203.0.113.0).

- The AWS IoT SiteWise data endpoint: `data.iotsitewise.region.amazonaws.com` (port 443).

- The following AWS endpoints that the gateway uses. You can find these in the `/greengrass-root/config/config.json` file. Replace `greengrass-root` with the root of your AWS IoT Greengrass installation.
  - `ggHost`: `greengrass-ats.iot.region.amazonaws.com` (ports 443, 8443, and 8883).
  - `iotHost`: `prefix-ats.iot.region.amazonaws.com` (ports 443, 8443, and 8883).

  For more information, see AWS IoT Greengrass endpoints and quotas.

10. If the AWS IoT Greengrass Core software isn't already running, run the following command to start the AWS IoT Greengrass Core software. Replace `greengrass-root` with the root of your AWS IoT Greengrass installation. The default `greengrass-root` is `/greengrass`.

```bash
cd /greengrass-root/ggc/core
sudo ./greengrassd start
```

You should see this message: Greengrass successfully started with PID: `some-PID-number`

11. Configure the AWS IoT Greengrass Core software to automatically start when your gateway turns on. Consult the documentation for your gateway's operating system.
Creating an IAM policy and role

You must create an AWS Identity and Access Management (IAM) policy and role to allow the gateway to access AWS IoT SiteWise on your behalf.

To create an IAM policy and role

1. Navigate to the IAM console.
2. In the navigation pane, choose Policies, and then choose Create policy.

3. On the JSON tab, delete the current contents of the policy field, and paste the following policy into the field.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "iotsitewise:BatchPutAssetPropertyValue",
         "Resource": "*"
      }
   ]
}
```

Note

To improve security, you can specify an AWS IoT SiteWise asset hierarchy path in the Condition property. The following example is a trust policy that specifies an asset hierarchy path.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "iotsitewise:BatchPutAssetPropertyValue",
         "Resource": "*",
         "Condition": {
            "StringLike": {
               "iotsitewise:assetHierarchyPath": [
```
5. Enter a name and description for the policy, and then choose Create policy.
6. In the navigation pane, choose Roles, and then choose Create role.

7. Under Select type of trusted entity, choose AWS service. Under Choose the service that will use the role, choose Greengrass as the service that will use the role, and then choose Next: Permissions.
8. Search for the policy that you created, select the check box, and then choose Next: Tags.
9. (Optional) Add tags to your role, and then choose **Next: Review**.
10. Enter a name and description for the role, and then choose **Create role**.
11. In the green banner, choose the link to your new role. You can also use the search field to find the role.

![The role SiteWiseDemo has been created.]

12. Choose the Trust relationships tab, and then choose Edit trust relationship.
13. Replace the current contents of the policy field with the following, and then choose **Update Trust Policy**.

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

### Configuring an AWS IoT Greengrass group

**To attach an IAM role to a group and enable stream manager**

1. Navigate to the **AWS IoT Greengrass console**.
2. In the left navigation pane, under **Greengrass**, choose **Groups**, and then choose the group that you created in **Setting up the gateway environment (p. 89)**.
3. In the left navigation pane, choose **Settings**. In the **Group Role** section, choose **Add Role**.

4. Choose the role that you created in **Creating an IAM policy and role (p. 91)**, and then choose **Save**.

5. On the **Settings** page, in the **Stream manager** section, choose **Edit**.

Stream manager is a feature of AWS IoT Greengrass that enables your AWS IoT Greengrass Core to stream data to the AWS Cloud. AWS IoT SiteWise gateways require that stream manager is enabled. For more information, see Manage data streams on the AWS IoT Greengrass Core in the AWS IoT Greengrass Version 1 Developer Guide.
6. Choose **Enable**, and then choose **Save**.
7. In the upper-left corner, choose **Services** to prepare for the next procedure.

### Configuring the AWS IoT SiteWise connector

In this procedure, you configure the AWS IoT SiteWise connector on your Greengrass group. Connectors are prebuilt modules that accelerate the development lifecycle for common edge scenarios. For more information, see [AWS IoT Greengrass connectors](https://docs.aws.amazon.com/iot/greengrass/latest/developerguide/iot-greengrass-connector.html) in the [AWS IoT Greengrass Version 1 Developer Guide](https://docs.aws.amazon.com/iot/greengrass/latest/developerguide/).

**To configure the AWS IoT SiteWise connector**

1. Navigate to the **AWS IoT Greengrass** console.
2. In the left navigation pane, under **Greengrass**, choose **Groups**, and then choose the group that you created in [Setting up the gateway environment](p. 89).
3. In the left navigation page, choose **Connectors**. On the **Connectors** page, choose **Add a connector**.
4. Choose **IoT SiteWise** from the list and choose **Next**.
5. If your server requires authentication, you can create AWS Secrets Manager secrets with the server’s user name and password. Then, you can attach each secret to your Greengrass group and choose them under List of ARNs for username/password secrets. For more information about how to create and configure secrets, see Configuring source authentication (p. 114). You can also add secrets to your connector later.

6. If you set up your gateway with a different path than /var/sitewise, enter that path for Local storage path.

7. (Optional) Enter a maximum disk buffer size for the connector. If the AWS IoT Greengrass core loses connection to the AWS Cloud, the connector caches data until it can successfully connect. If the
cache size exceeds the maximum disk buffer size, the connector discards the oldest data from the queue.

8. Choose Add.

9. In the upper-right corner, in the Actions menu, choose Deploy.

10. Choose Automatic detection to start the deployment.

   If the deployment fails, choose Deploy again. If the deployment continues to fail, see AWS IoT Greengrass deployment troubleshooting.

Adding the gateway to AWS IoT SiteWise

In this procedure, you add your gateway's Greengrass group to AWS IoT SiteWise. After you register your gateway with AWS IoT SiteWise, the service can deploy your data source configurations to your gateway.

To add the gateway to AWS IoT SiteWise

1. Navigate to the AWS IoT SiteWise console.
2. Choose Add gateway.
3. On the Add SiteWise gateway page, do the following:
   a. Enter a Name for the gateway. Consider including the location of the gateway in the name so that you can easily identify it.
   b. For Greengrass group ID, choose the Greengrass group that you created earlier.

Example

![Add SiteWise gateway](image)

c. (Optional) For Edge capabilities, choose Data processing pack. This enables communication between your gateway and any asset models and assets configured for the edge. For more information, see the section called “Enabling edge data processing” (p. 130).
If you add the data processing pack to your gateway, you must configure and deploy the SiteWise Edge connector on your AWS IoT Greengrass group. Follow the next steps:

d. Choose **Add gateway**.

4. If you add the data processing pack to your gateway, configure and deploy the AWS IoT SiteWise Data Processor connector on your AWS IoT Greengrass group. Follow the steps in the section called “Configuring the AWS IoT SiteWise connector” (p. 98) to configure the AWS IoT SiteWise Data Processor connector:

   a. For **Select a connector** in the AWS IoT Greengrass console, choose **AWS IoT SiteWise Data Processor**.
   
   b. For **Local storage path**, enter the path to your gateway.
   
   c. Choose **Add**.
   
   d. In the upper-right corner, in the **Actions** menu, choose **Deploy**, and then choose **Automatic detection** to start the deployment.

After your gateway deploys, you can add a source for each server from which you want your gateway to ingest data. For more information, see **Configuring data sources** (p. 102).

You can view Amazon CloudWatch metrics to verify that your gateway connects to AWS IoT SiteWise. For more information, see **Gateway metrics** (p. 365).

### Configuring gateway dependencies on Amazon Elastic Compute Cloud

You can configure a gateway that runs on Amazon EC2. The gateway runs in the AWS Cloud and ingests data from your industrial data sources to AWS IoT SiteWise. For more information, see **What is Amazon EC2?** in the **Amazon EC2 User Guide for Linux Instances**.

AWS IoT SiteWise provides an AWS CloudFormation template that you can use to easily create gateway dependencies on an Amazon EC2 instance. When you create a stack from the template, AWS CloudFormation creates the required AWS resources for you to run a gateway. Then, you can create a gateway that uses the AWS IoT Greengrass Core running on the Amazon EC2 instance.

The AWS CloudFormation template creates the following resources:

- An Amazon EC2 instance with AWS IoT Greengrass Core software
- An AWS IoT Greengrass group with info-level logging for CloudWatch Logs and the local file system
- An Amazon Virtual Private Cloud
- AWS Identity and Access Management (IAM) roles

**Important**

You will be charged for the resources that this AWS CloudFormation template creates and uses. These charges include an AWS IoT Greengrass device and compute capacity and data transfer for Amazon EC2.

### Prerequisites

To create gateway dependencies on Amazon EC2 from this stack template, you need the following:

- The AWS IoT SiteWise service-linked role in your AWS account. This role creates automatically when you use the AWS IoT SiteWise console. For more information, see **Using service-linked roles for AWS IoT SiteWise** (p. 339).
- An AWS IoT Greengrass service role attached to your AWS account in the Region where you create this stack. For more information, see **AWS IoT Greengrass service role** in the **AWS IoT Greengrass Version 1 Developer Guide**.
Creating the AWS CloudFormation stack

You can create a stack in AWS CloudFormation to create an Amazon EC2 instance with AWS IoT Greengrass gateway dependencies.

To create gateway dependencies on Amazon EC2

1. Open the AWS CloudFormation template and sign in to the AWS Management Console.
2. On the Create stack page, choose Next at the bottom of the page.
3. On the Specify stack details page, enter a GroupName for the AWS IoT Greengrass group that this template creates for the gateway.
4. (Optional) Change any of the template's other parameters:
   - **InstanceType** – The Amazon EC2 instance type. For more information, see Instance types in the Amazon EC2 User Guide for Linux Instances.
   - **SecurityAccessCIDR** – The CIDR block for the virtual private cloud (VPC). For more information, see VPCs and subnets in the Amazon VPC User Guide.
5. Choose Next.
6. On the Configure stack options page, choose Next.
7. At the bottom of the page, choose the check boxes that acknowledge that AWS CloudFormation requires access capabilities.
8. Choose Create stack.

The stack takes around 5 minutes to create. If the stack fails to create, your account might have insufficient permissions, or you might not have the prerequisite IAM roles. Follow these steps to delete the stack and try again:

a. Choose Delete in the upper-right corner.
   The stack takes a few minutes to delete.

b. If the stack fails to delete, choose Delete again.

c. If the stack fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.

9. After the stack creates successfully, you can create a gateway with the AWS IoT Greengrass group that deploys to the Amazon EC2 instance. For more information, see Adding the gateway to AWS IoT SiteWise (p. 100).

Important
After you create the stack, you can see the new resources in your AWS account. Your gateway might stop working correctly if you delete or modify these resources. We recommend that you don't modify these resources unless you want to change settings on your gateway's AWS IoT Greengrass group.

Configuring data sources

After you set up a gateway, you can configure data sources so that your gateway can ingest data from local servers to AWS IoT SiteWise. Each source represents a local server, such as an OPC-UA server, that your gateway connects and retrieves industrial data streams. For more information about setting up a gateway, see Configuring a gateway (p. 88).

Note
AWS IoT SiteWise restarts your gateway each time you add or edit a source. Your gateway won't ingest data while it's restarting. The time to restart your gateway depends on the number of tags on your gateway's sources. Restart time can range from a few seconds (for a gateway with few tags) to several minutes (for a gateway with many tags).
After you create sources, you can associate your data streams with asset properties. For more information about how to create and use assets, see Modeling industrial assets (p. 139) and Mapping industrial data streams to asset properties (p. 202).

You can view CloudWatch metrics to verify that a data source is connected to AWS IoT SiteWise. For more information, see Gateway metrics (p. 365).

Currently, AWS IoT SiteWise supports the following data source protocols:

- **OPC-UA** – A machine-to-machine (M2M) communication protocol for industrial automation.
- **Modbus TCP** – A data communications protocol used to interface with programmable logic controllers (PLCs).
- **Ethernet/IP (EIP)** – An industrial network protocol that adapts the Common Industrial Protocol (CIP) to standard Ethernet.

**Note**
Gateways running on AWS IoT Greengrass V2 currently don’t support Modbus TCP and Ethernet IP sources.

**Topics**

- Configure an OPC-UA source (p. 103)
- Configure a Modbus TCP source (p. 121)
- Configure an Ethernet/IP (EIP) source (p. 124)
- Choosing a destination for your source server data (p. 128)
- Upgrading a connector (p. 129)

**Configure an OPC-UA source**

You can use the AWS IoT SiteWise console or a gateway capability to define and add an OPC-UA source to your gateway to represent a local OPC-UA server.

**Topics**

- Configure an OPC-UA source (console) (p. 103)
- Configure an OPC-UA source (CLI) (p. 105)
- Enabling your source servers to trust the gateway (p. 112)
- Configuring source authentication (p. 114)
- Filter data ingestion ranges with OPC-UA (p. 119)
- Using OPC-UA node filters (p. 120)

**Configure an OPC-UA source (console)**

**To configure an OPC-UA source using the AWS IoT SiteWise console**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Gateways**.
3. On the gateway that you want to create a source for, choose **Manage**, and then choose **View details**.
4. Choose **New source** in the upper-right corner.
5. For **Protocol options**, choose **OPC-UA**.
6. For **OPC-UA source configuration**, enter a **Name** for the source.
7. For **IP address or hostname**, enter the local endpoint of the data source server. For example, your local endpoint might look like `opc.tcp://203.0.113.0:49320`.
8. (Optional) Enter a **Data stream prefix**. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.
9. Choose a **Message security mode** for connections and data in transit between your source server and your gateway. This field is the combination of the OPC-UA security policy and message security mode. You must choose the same security policy and message security mode that you specified for your OPC-UA server.

Choose the security policy from the following options:

- **None** – The gateway doesn’t secure connections to the OPC-UA source. We recommend that you choose a different security policy.
- **Basic256Sha256** – The **Basic256Sha256** security policy.
- **Aes128_Sha256_RsaOaep** – The **Aes128_Sha256_RsaOaep** security policy.
- **Aes256_Sha256_RsaPss** – The **Aes256_Sha256_RsaPss** security policy.
- **Basic128Rsa15** – (Deprecated) The **Basic128Rsa15** security policy is deprecated in the OPC-UA specification because it’s no longer considered secure. We recommend that you choose a different security policy. For more information, see **Basic128Rsa15**.
- **Basic256** – (Deprecated) The **Basic256** security policy is deprecated in the OPC-UA specification because it’s no longer considered secure. We recommend that you choose a different security policy. For more information, see **Basic256**.

Except for the **None** option, each security policy has two options for message security mode:

- **Sign** – The data in transit between the gateway and the source is signed but not encrypted.
- **Sign and encrypt** – The data in transit between the gateway and the source is signed and encrypted.

**Important**

If you choose a message security mode other than **None**, you must enable your source server to trust the gateway. For more information, see **Enabling your source servers to trust the gateway (p. 112)**.

10. If your source requires authentication, choose an AWS Secrets Manager secret from the **Authentication configuration list**. The gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your gateway’s IoT SiteWise
connector to use them for source authentication. For more information, see Configuring source authentication (p. 114).

**Tip**
Your data server might have an option named **Allow anonymous login**. If this option is **Yes**, then your source doesn't require authentication.

11. For **Property groups**, enter a **Name**.

12. For **Properties**:
   a. (Optional) For **Node paths**, add OPC-UA node filters to limit which OPC-UA paths are uploaded to AWS IoT SiteWise. You can use node filters to reduce your gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with `/Server/`. To define OPC-UA node filters, you can use node paths and the * and ** wildcard characters. For more information, see Using OPC-UA node filters (p. 120).
   b. For **Scan mode**, choose the mode that you want AWS IoT SiteWise to use to collect your data. For more information about scan mode, see the section called "Filtering data ingestion ranges with OPC-UA" (p. 119).
   c. For **Scan rate**, update the rate that want the gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your gateway.
   d. (Optional) Configure a **Deadband setting** for your source. This controls what data your source sends to your AWS IoT SiteWise, and what data it discards. For more information about the deadband setting, see the section called "Filtering data ingestion ranges with OPC-UA" (p. 119).

13. (Optional) For **Destination**, choose where the source data is sent. By default, your source sends data to AWS IoT SiteWise. You can use a AWS IoT Greengrass stream to export your data to a local destination or to the AWS Cloud instead.

   **Note**
   You must choose AWS IoT SiteWise as your source destination if you want to process data from this source at the edge with AWS IoT SiteWise. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

   To send your data to another destination:
   a. For **Destination options**, choose **Other destinations**.
   b. For **Greengrass stream name**, enter the exact name of your AWS IoT Greengrass stream.

   **Note**
   You can use a stream that you've already created, or you can create a new AWS IoT Greengrass stream to export your data. If you want to use an existing stream, you must enter the exact name of the stream or a new stream will be created.
   For more information about working with AWS IoT Greengrass streams, see Manage data streams in the AWS IoT Greengrass Version 1 Developer Guide.

14. Choose **Add source**.

   AWS IoT SiteWise deploys the gateway configuration to your AWS IoT Greengrass core. You don't need to manually trigger a deployment.

**Configure an OPC-UA source (CLI)**

You can define OPC-UA data sources in a gateway capability. You must define all of your OPC-UA sources in a single capability configuration.

For more information about defining sources with the AWS Command Line Interface, see the section called "Configuring data sources (AWS CLI)” (p. 127).
This capability has the following versions.

<table>
<thead>
<tr>
<th>Version</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iotsitewise:opcuacollector:1</td>
</tr>
</tbody>
</table>

**OPC-UA capability configuration parameters**

When you define OPC-UA sources in a capability configuration, you must specify the following information in the `capabilityConfiguration` JSON document:

**sources**

A list of OPC-UA source definition structures that each contain the following information:

- **name**
  A unique, friendly name for the source.

- **endpoint**
  An endpoint structure that contains the following information:

  - **certificateTrust**
    A certificate trust policy structure that contains the following information:

    - **type**
      The certificate trust mode for the source. Choose one of the following:
      - **TrustAny** – The gateway trusts any certificate when it connects to the OPC-UA source.
      - **X509** – The gateway trusts an X.509 certificate when it connects to the OPC-UA source. If you choose this option, you must define `certificateBody` in `certificateTrust`. You can also define `certificateChain` in `certificateTrust`.

    - **certificateBody**
      (Optional) The body of an X.509 certificate.
      This field is required if you choose X509 for `type` in `certificateTrust`.

    - **certificateChain**
      (Optional) The chain of trust for an X.509 certificate.
      This field is used only if you choose X509 for `type` in `certificateTrust`.

- **endpointUri**
  The local endpoint of the OPC-UA source. For example, your local endpoint might look like `opc.tcp://203.0.113.0:49320`.

- **securityPolicy**
  The security policy to use so that you can secure messages that are read from the OPC-UA source. Choose one of the following:
  - **NONE** – The gateway doesn’t secure messages from the OPC-UA source. We recommend that you choose a different security policy. If you choose this option, you must also choose NONE for `messageSecurityMode`.
  - **BASIC256_SHA256** – The Basic256Sha256 security policy.
  - **AES128_SHA256_RSAOAEP** – The Aes128_Sha256_RsaOaep security policy.
Using a gateway

- **AES256_SHA256_RSAPSS** – The Aes256_Sha256_RsaPss security policy.
- **BASIC128_RSA15** – (Deprecated) The Basic128Rsa15 security policy is deprecated in the OPC-UA specification because it's no longer considered secure. We recommend that you choose a different security policy. For more information, see Basic128Rsa15.
- **BASIC256** – (Deprecated) The Basic256 security policy is deprecated in the OPC-UA specification because it's no longer considered secure. We recommend that you choose a different security policy. For more information, see Basic256.

**Important**
If you choose a security policy other than **NONE** you must choose **SIGN** or **SIGN_AND_ENCRYPT** for messageSecurityMode. You must also configure your source server to trust the gateway. For more information, see Enabling your source servers to trust the gateway (p. 112).

**messageSecurityMode**
The message security mode to use to secure connections to the OPC-UA source. Choose one of the following:

- **NONE** – The gateway doesn't secure connections to the OPC-UA source. We recommend that you choose a different message security mode. If you choose this option, you must also choose **NONE** for securityPolicy.
- **SIGN** – Data in transit between the gateway and the OPC-UA source is signed but not encrypted.
- **SIGN_AND_ENCRYPT** – Data in transit between the gateway and the OPC-UA source is signed and encrypted.

**Important**
If you choose a message security mode other than **NONE** you must choose a securityPolicy other than **NONE**. You must also configure your source server to trust the gateway. For more information, see Enabling your source servers to trust the gateway (p. 112).

**identityProvider**
An identity provider structure that contains the following information:

- **type**
The type of authentication credentials required by the source. Choose one of the following:
  - **Anonymous** – The source doesn't require authentication to connect.
  - **Username** – The source requires a user name and password to connect. If you choose this option, you must define **usernameSecretArn** in identityProvider.

- **usernameSecretArn**
(Optional) The ARN of an AWS Secrets Manager secret. The gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your gateway's IoT SiteWise connector to use them for source authentication. For more information, see Configuring source authentication (p. 114).

This field is required if you choose Username for type in identityProvider.

- **nodeFilterRules**
A list of node filter rule structures that define the OPC-UA data stream paths to send to the AWS Cloud. You can use node filters to reduce your gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with /Server/. To define OPC-UA node filters, you can use node paths and the * and ** wildcard characters. For more information, see Using OPC-UA node filters (p. 120).
Each structure in the list must contain the following information:

**action**

The action for this node filter rule. You can choose the following option:
- **INCLUDE** – The gateway includes only data streams that match this rule.

**definition**

A node filter rule structure that contains the following information:

**type**

The type of node filter path for this rule. You can choose the following option:
- **OpcUaRootPath** – The gateway evaluates this node filter path against the root of the OPC-UA path hierarchy.

**rootPath**

The node filter path to evaluate against the root of the OPC-UA path hierarchy. This path must start with `/`.

**measurementDataStreamPrefix**

A string to prepend to all data streams from the source. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.

**propertyGroups**

(Optional) The list of property groups that define the deadband and scanMode requested by the protocol.

**name**

The name of the property group. This should be a unique identifier.

**deadband**

The deadband structure that contains the following information:

**type**

The supported types of deadband. Accepted values are **ABSOLUTE** and **PERCENT**.

**value**

The value of the deadband. When **type** is **ABSOLUTE**, this value is a unitless double. When **type** is **PERCENT**, this value is a double between 1 and 100.

**eguMin**

(Optional) The engineering unit minimum when using a **PERCENT** deadband. You set this if the OPC-UA server doesn't have engineering units configured.

**eguMax**

(Optional) The engineering unit maximum when using a **PERCENT** deadband. You set this if the OPC-UA server doesn't have engineering units configured.

**timeoutMilliseconds**

The duration in milliseconds before timeout. The minimum is 100.

**scanMode**

The scanMode structure that contains the following information:

**type**

The supported types of **scanMode**. Accepted values are **POLL** and **EXCEPTION**.
rate

The sampling interval for the scan mode.

Capability configuration examples

The following example defines an OPC-UA gateway capability configuration from a payload stored in a JSON file.

```bash
aws iotsitewise update-gateway-capability-configuration \
    --capability-namespace "iotsitewise:opcuacollector:1" \
    --capability-configuration file://opc-ua-configuration.json
```

Example : OPC-UA source configuration

The following `opc-ua-configuration.json` file defines a basic, insecure OPC-UA source configuration.

```json
{
    "sources": [
        {
            "name": "Wind Farm #1",
            "endpoint": {
                "certificateTrust": {
                    "type": "TrustAny"
                },
                "endpointUri": "opc.tcp://203.0.113.0:49320",
                "securityPolicy": "NONE",
                "messageSecurityMode": "NONE",
                "identityProvider": {
                    "type": "Anonymous"
                },
                "nodeFilterRules": []
            },
            "measurementDataStreamPrefix": ""
        }
    ]
}
```

Example : OPC-UA source configuration with defined property groups

The following `opc-ua-configuration.json` file defines a basic, insecure OPC-UA source configuration with defined property groups.

```json
{
    "sources": [
        {
            "name": "source1",
            "endpoint": {
                "certificateTrust": {
                    "type": "TrustAny"
                },
                "endpointUri": "opc.tcp://10.0.0.9:49320",
                "securityPolicy": "NONE",
                "messageSecurityMode": "NONE",
                "identityProvider": {
                    "type": "Anonymous"
                },
                "nodeFilterRules": [
                    {
                        "action": "INCLUDE",
                        "definition": {
```
Example: OPC-UA source configuration with properties

The following JSON example for opc-ua-configuration.json defines an OPC-UA source configuration with the following properties:
• Trusts any certificate.
• Uses the BASIC256 security policy to secure messages.
• Uses the SIGN_AND_ENCRYPT mode to secure connections.
• Uses authentication credentials stored in a Secrets Manager secret.
• Filters out data streams except those whose path starts with /WindFarm/2/WindTurbine/.
• Adds /Washington to the start of every data stream path to distinguish between this “Wind Farm #2” and a “Wind Farm #2” in another area.

```json
{
    "sources": [
    {
        "name": "Wind Farm #2",
        "endpoint": {
            "certificateTrust": {
                "type": "TrustAny"
            },
            "endpointUri": "opc.tcp://203.0.113.1:49320",
            "securityPolicy": "BASIC256",
            "messageSecurityMode": "SIGN_AND_ENCRYPT",
            "identityProvider": {
                "type": "Username",
            },
            "nodeFilterRules": [
                {
                    "action": "INCLUDE",
                    "definition": {
                        "type": "OpcUaRootPath",
                        "rootPath": "/WindFarm/2/WindTurbine/
                    }
                }
            ],
            "measurementDataStreamPrefix": "/Washington"
        }
    }
}
```

**Example**

The following JSON example for opc-ua-configuration.json defines an OPC-UA source configuration with the following properties:

• Trusts a given X.509 certificate.
• Uses the BASIC256 security policy to secure messages.
• Uses the SIGN_AND_ENCRYPT mode to secure connections.

```json
{
    "sources": [
    {
        "name": "Wind Farm #3",
        "endpoint": {
            "certificateTrust": {
                "type": "X509",
                "certificateBody": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNV
BAgTAldBMRAwDgYDVQQHEwdTZXJ0b1dQcGxheXBlZXMwIGh0bWwgaXMgdG8gaWJsb2d5IHRoZ
A==
-----END CERTIFICATE-----
"
            },
            "endpointUri": "opc.tcp://203.0.113.1:49320",
            "securityPolicy": "BASIC256",
            "messageSecurityMode": "SIGN_AND_ENCRYPT",
            "identityProvider": {
                "type": "Username",
            },
            "nodeFilterRules": [
                {
                    "action": "INCLUDE",
                    "definition": {
                        "type": "OpcUaRootPath",
                        "rootPath": "/WindFarm/2/WindTurbine/
                    }
                }
            ],
            "measurementDataStreamPrefix": "/Washington"
        }
    }
}
```
Enabling your source servers to trust the gateway

If you choose a message security mode other than None, you must enable your source servers to trust the gateway. The gateway generates a certificate that you must accept on your source server. Steps can vary depending on the source servers that you use. Consult the documentation for each server.

The procedure might be similar to the following steps.

To enable an OPC-UA server to trust the gateway

1. Open the interface for configuring your OPC-UA server (for example, right-click the OPC-UA icon in the system tray).
2. Enter the user name and password for the OPC-UA server administrator.
3. Locate Trusted Clients in the interface, and then choose AWS IoT SiteWise Gateway Client.
4. Choose Trust.
Exporting the OPC-UA client certificate

Some OPC-UA servers require access to the OPC-UA client certificate file to trust the gateway. If this applies to your OPC-UA servers, you can use the following procedure to export the OPC-UA client certificate from the gateway. Then, you can import the certificate on your OPC-UA server.

To export the OPC-UA client certificate file for a source

1. The gateway stores an OPC-UA client certificate for each source. Each source is identified by a unique ID. The gateway stores source IDs in a configuration file located at /sitewise-root/config/sitewise-COLLECTOR-config.json. You can't use the AWS IoT SiteWise API to return the source IDs, so you must find them in this configuration file.

On the gateway, run one of the following commands to print the output of the collector configuration file. Replace sitewise-root with the local storage path for your AWS IoT SiteWise configuration. The default sitewise-root is /var/sitewise.

- If you have jq installed, run the following command to pretty-print the configuration file with syntax highlighting.
  
  ```bash
  cat /sitewise-root/config/sitewise-COLLECTOR-config.json | jq .
  ```

- If you have Python installed, run the following command to pretty-print the configuration file.
  
  ```bash
  cat /sitewise-root/config/sitewise-COLLECTOR-config.json | python -m json.tool
  ```

- If you don't have a JSON printing tool, run the following command to print the configuration file.
  
  ```bash
  cat /sitewise-root/config/sitewise-COLLECTOR-config.json
  ```

Example: Configuration file for a gateway

The following JSON example demonstrates a configuration file for a gateway with one basic OPC-UA source.

```json
{
   "creationDate": 1588369971457,
   "dataVersion": null,
   "gatewayConfiguration": {
      "schemaVersion": "DefaultSchemaVersion",
      "sources": [
         {
            "endpoint": {
               "certificateTrust": {
                  "type": "TrustAny"
               },
               "endpointUri": "opc.tcp://203.0.113.0:49320",
               "identityProvider": {
                  "type": "Anonymous"
               },
               "messageSecurityMode": "NONE",
               "nodeFilterRules": [],
               "securityPolicy": "NONE"
            },
            "id": "a1b2c3d4-5678-90ab-cdef-1c1c1EXAMPLE",
            "measurementDataStreamPrefix": "",
            "name": "Wind Farm #1",
            "type": "OpcUaSource"
         }
      ]
   }
}
```
Find the source that corresponds to the OPC-UA server. The ID of the source is in the `id` field. In the above example, `a1b2c3d4-5678-90ab-cdef-1c1c1EXAMPLE` is the source ID for the OPC-UA source named Wind Farm #1.

2. Run the following command to change to the directory that contains the certificate file. Replace `sitewise-root` with the local storage path for your AWS IoT SiteWise configuration and replace `source-id` with the source ID that you found in the previous step.

   ```bash
cd /sitewise-root/pusher/source-id/opcua-certificate-store
   ```

3. The gateway's OPC-UA client certificate for this source is in the `aws-iot-opcua-client.pfx` file.

   Run the following command to export the certificate to a `.pem` file called `aws-iot-opcua-client-certificate.pem`.

   ```bash
   ```

4. Transfer the certificate file, `aws-iot-opcua-client-certificate.pem`, from the gateway to the OPC-UA server.

   To do so, you can use common software such as the `scp` program to transfer the file using the SSH protocol. For more information, see Secure copy on Wikipedia.

   **Note**
   If your gateway is running on Amazon Elastic Compute Cloud (Amazon EC2) and you're connecting to it for the first time, you must configure prerequisites to connect. For more information, see Connect to your Linux instance in the Amazon EC2 User Guide for Linux Instances.

5. Import the certificate file, `aws-iot-opcua-client-certificate.pem`, on the OPC-UA server to trust the gateway. Steps can vary depending on the source server that you use. Consult the documentation for the server.

**Configuring source authentication**

If your OPC-UA servers require authentication credentials to connect, you can define a user name and password in a secret for each source in AWS Secrets Manager. Then, you add the secret to your Greengrass group and IoT SiteWise connector to make the secret available to your gateway. For more information, see Deploy secrets to the AWS IoT Greengrass core in the AWS IoT Greengrass Version 1 Developer Guide.
After a secret is available to your gateway, you can choose it when you configure a source. Then, the gateway uses the authentication credentials from the secret when it connects to the source. For more information, see Configuring data sources (p. 102).

Topics
- Creating source authentication secrets (p. 115)
- Adding secrets to a Greengrass group (p. 116)
- Adding secrets to an IoT SiteWise connector (p. 117)

Creating source authentication secrets

In this procedure, you create an authentication secret for your source in Secrets Manager. In the secret, define `username` and `password` key-value pairs that contain authentication details for your source.

To create a source authentication secret

1. Navigate to the Secrets Manager console.
2. Choose Store a new secret.
3. Under Select secret type, choose Other type of secrets.
4. Enter `username` and `password` key-value pairs for your OPC-UA server’s authentication values, and then choose Next.

5. Enter a Secret name that begins with `greengrass-`, such as `greengrass-factory1-auth`.

   **Important**
   You must use the `greengrass-` prefix for the default AWS IoT Greengrass service role to access your secrets. If you want to name your secrets without this prefix, you must grant AWS IoT Greengrass custom permissions to access your secrets. For more information, see Allow AWS IoT Greengrass to get secret values in the AWS IoT Greengrass Version 1 Developer Guide.
6. Enter a **Description** and choose **Next**.

7. (Optional) On the **Configure automatic rotation** page, configure automatic rotation for your secrets. If you configure automatic rotation, you must redeploy your Greengrass group each time a secret rotates.

8. On the **Configure automatic rotation** page, choose **Next**.

9. Review your new secret and choose **Store**.

**Adding secrets to a Greengrass group**

In this procedure, you add your source authentication secrets to your AWS IoT Greengrass group to make them available to your IoT SiteWise connector.

**To add a secret to your Greengrass group**

1. Navigate to the AWS IoT Greengrass console.
2. In the navigation pane, under **Greengrass**, choose **Groups**, and then choose your group.
3. In the navigation page, choose **Resources**.
4. On the **Resources** page, choose the **Secret** tab, and then choose **Add a secret resource**.
5. Choose Select and choose your secret from the list.
6. Choose Next.
7. In Secret resource name, enter a name for your secret resource and choose Save.

Adding secrets to an IoT SiteWise connector

In this procedure, you add your source authentication secrets to your IoT SiteWise connector to make them available to AWS IoT SiteWise and your gateway.

To add a secret to your IoT SiteWise connector
1. Navigate to the AWS IoT Greengrass console.
2. In the navigation pane, under Greengrass, choose Groups, and then choose your group.
3. In the navigation page, choose Connectors.
4. Choose the ellipsis icon for the IoT SiteWise connector to open the options menu, and then choose Edit.

5. Under List of ARNs for OPC-UA username/password secrets, choose Select, and then select each secret to add to this gateway. If you need to create secrets, see Creating source authentication secrets (p. 115).
If your secret doesn't appear, choose Refresh. If your secret still doesn't appear, check that you added the secret to your Greengrass group (p. 116).

6. Choose Save.
7. In the upper-right corner, in the Actions menu, choose Deploy.
8. Choose Automatic detection to start the deployment.

If the deployment fails, choose Deploy again. If the deployment continues to fail, see AWS IoT Greengrass deployment troubleshooting.

After your group deploys, you can configure a source that uses the new secret. For more information, see Configuring data sources (p. 102).

Filter data ingestion ranges with OPC-UA

You can control the way you ingest data with an OPC-UA source by using scan mode and deadband ranges. These features let you control what kind of data to ingest, and how and when your server and gateway exchange this information.

Control data collection frequency with Scan mode

You can configure your OPC-UA scan mode to control the way you collect data from your OPC-UA source. You can choose subscription or polling mode.

- Subscription mode – The OPC-UA source collects data to send to your gateway at the frequency defined by your scan rate. The server only sends data when the value has changed, so this is the maximum frequency your gateway receives data.
- Polling mode – Your gateway polls the OPC-UA source at a set frequency defined by your scan rate. The server sends data regardless of whether the value has changed, so your gateway always receives data at this interval.

Note
The polling mode option overrides your deadband settings for this source.

Filter OPC-UA data ingestion with deadband ranges

You can apply a deadband to your OPC-UA source property groups to filter out and discard certain data instead of sending it to the AWS Cloud. A deadband specifies a window of expected fluctuations in the incoming data values from your OPC-UA source. If the values fall within this window, your OPC-UA server won't send it to the AWS Cloud. You can use deadband filtering to reduce the amount of data you're processing and sending to the AWS Cloud. To learn how to set up OPC-UA sources for your gateway, see the section called “Configuring data sources” (p. 102).

Note
Your server deletes all data that falls inside the window specified by your deadband. You can't recover this discarded data.

Types of deadbands

You can specify two types of deadbands for your OPC-UA server property group. These let you choose how much data is sent to the AWS Cloud, and how much is discarded.

- Percentage – You specify a window using a percentage of expected fluctuation in the measurement value. The server calculates the exact window from this percentage, and sends data to the AWS Cloud that exceeds falls outside the window. For example, specifying a 2% deadband value on a sensor with a range from -100 degrees Fahrenheit to +100 degrees Fahrenheit tells the server to send data to the AWS Cloud when the value changes by 4 degrees Fahrenheit or more.
Note
You can optionally specify a minimum and maximum deadband value for this window if your source server doesn't define engineering units. If an engineering unit range is not provided, the OPC-UA server defaults to the full range of the measurement data type.

- Absolute – You specify a window using exact units. For example, specifying a deadband value of 2 on a sensor tells the server to send data to the AWS Cloud when its value changes by at least 2 units. You can use absolute deadbanding for dynamic environments where fluctuations are regularly expected during normal operations.

Deadband timeouts
You can optionally configure a deadband timeout setting. After this timeout, the OPC-UA server sends the current measurement value even if it is within the expected deadband fluctuation. You can use the timeout setting to ensure that AWS IoT SiteWise is ingesting a steady stream of data at all times, even when values do not exceed the defined deadband window.

Using OPC-UA node filters
When you define OPC-UA data sources for an AWS IoT SiteWise gateway, you can define node filters. Node filters let you limit which data stream paths the gateway sends to the cloud. You can use node filters to reduce your gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with /Server/. You can use the * and ** wildcard characters in your node filters to include multiple data stream paths with one filter. To learn how to set up OPC-UA sources for your gateway, see Configuring data sources (p. 102).

Note
AWS IoT SiteWise restarts your gateway each time you add or edit a source. Your gateway won't ingest data while it's restarting. The time to restart your gateway depends on the number of tags on your gateway's sources. Restart time can range from a few seconds (for a gateway with few tags) to several minutes (for a gateway with many tags).

The following table lists the wildcards that you can use to filter OPC-UA data sources.

### OPC-UA node filter wildcards

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches a single level in a data stream path.</td>
</tr>
<tr>
<td>**</td>
<td>Matches multiple levels in a data stream path.</td>
</tr>
</tbody>
</table>

Note
If you configure a source with a broad filter and then later change the source to use a more restrictive filter, AWS IoT SiteWise stops storing data that doesn't match the new filter.

Example Example scenario using node filters
Consider the following hypothetical data streams:

- /WA/Factory 1/Line 1/PLC1
- /WA/Factory 1/Line 1/PLC2
- /WA/Factory 1/Line 2/Counter1
- /WA/Factory 1/Line 2/PLC1
- /OR/Factory 1/Line 1/PLC1
- /OR/Factory 1/Line 2/Counter2
Using the previous data streams, you can define node filters to limit what data to include from your OPC-UA source.

1. To select all nodes in the state of Washington, use `/WA/`.
2. To select all `PLC` data streams, use `/*/*/PLC*` or `/**/PLC*`. You can include multiple directories or folders with the `*` wildcard characters.
3. To select all counters in the state of Washington, use `/WA/**/Counter*`.
4. To select all counters from `Line 2`, use `/**/Line 2/Counter*`.

**Configure a Modbus TCP source**

You can use the AWS IoT SiteWise console or a gateway capability to define and add a Modbus TCP source to your gateway. This source represents a local Modbus TCP server.

**Note**

- Gateways running on AWS IoT Greengrass V2 currently don't support Modbus TCP sources.
- You must install the AWS IoT SiteWise connector to use a Modbus TCP source.

You can use the Modbus TCP source to convert the data type from your source into a different data type when it's received on your gateway. The source data type determines the data types that you can choose for your destination data. You can also choose to swap bytes using the Modbus TCP source. The following table provides more information on the source data types, destination data types, and swap modes that are compatible.

For more information about swap modes, see the [How Real (Floating Point) and 32-bit Data is Encoded in Modbus RTU Messages](https://docs.aws.amazon.com/IoT-SiteWise/latest/userguide/how-real-data-and-32bit-data-are-encoded-in-modbus-rtu-messages.html) article on Modbus message encoding.

<table>
<thead>
<tr>
<th>Source data type</th>
<th>Compatible destination data types</th>
<th>Compatible swap modes</th>
<th>Compatible connector versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>String</td>
<td>noSwap</td>
<td>2</td>
</tr>
<tr>
<td>UTF8</td>
<td>String</td>
<td>noSwap</td>
<td>2</td>
</tr>
<tr>
<td>ISO8859</td>
<td>String</td>
<td>noSwap</td>
<td>2</td>
</tr>
<tr>
<td>Int16</td>
<td>Integer, Double, String</td>
<td>noSwap</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Int32</td>
<td>Integer, Double, String</td>
<td>noSwap, byteWordSwap, byteSwap, wordSwap</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Float</td>
<td>Double, String</td>
<td>noSwap, byteWordSwap, byteSwap, wordSwap</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>noSwap</td>
<td>1 and 2</td>
</tr>
<tr>
<td>Hex-dump</td>
<td>String</td>
<td>noSwap</td>
<td>1 and 2</td>
</tr>
</tbody>
</table>

**Topics**

Configure a Modbus TCP source (console)

To configure a Modbus TCP source

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Gateways.
3. On the gateway you want to create a source for, choose Manage, and then choose View details.

4. Choose New source in the upper-right corner.
5. For Protocol options, choose Modbus TCP.
6. For Modbus TCP source configuration, enter a Name for the source.
7. For IP address, enter the IP address for the data source server.
8. (Optional) Enter the Port and Unit ID for the source server.
9. (Optional) For Minimum inter-request duration, enter the time interval between subsequent requests sent to your server. Your gateway automatically calculates the minimum allowable interval based on your device and the number of registers you have.
10. For Property groups, enter a Name.
11. For Properties:
   a. For Tag, enter a property alias for your register set. For example, TT-001.
   b. For Register address, enter the register address that starts the register set.
   c. For Source data type, choose the Modbus TCP data type you want to convert data from. This defaults to Hex dump.

   **Note**
   The source data type you choose determines the data size, destination data type, and swap mode you can choose. For more information, see the section called “Configure a Modbus TCP source” (p. 121).

d. For Data size, enter the number of registers to read when starting from the Register address. This is determined by the source data type you choose for this source.

e. For Destination data type, choose the AWS IoT SiteWise data type that you want your data to be converted to. The default is String. The destination type must be compatible with the source data type you choose for this source. For more information, see the section called “Configure a Modbus TCP source” (p. 121).

f. For Swap mode, choose the data swap mode you want to use to read data from your register set. The swap mode must be compatible with the source data type you choose for this source. For more information, see the section called "Configure a Modbus TCP source" (p. 121).

12. For Scan rate, update the rate at which you want the gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your gateway.
13. (Optional) For Destination, choose where the source data is sent. By default, your source sends data to AWS IoT SiteWise. You can use a AWS IoT Greengrass stream to export your data to a local destination or to the AWS Cloud instead.
Note
You must choose AWS IoT SiteWise as your source destination if you want to process data from this source at the edge with AWS IoT SiteWise. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

To send your data to another destination:

a. For **Destination options**, choose **Other destinations**.

b. For **Greengrass stream name**, enter the exact name of your AWS IoT Greengrass stream.

Note
You can use a stream that you've already created, or you can create a new AWS IoT Greengrass stream to export your data. If you want to use an existing stream, you must enter the exact name of the stream or a new stream will be created.

For more information about working with AWS IoT Greengrass streams, see Manage data streams in the AWS IoT Greengrass developer guide.

14. Choose **Add source**.

AWS IoT SiteWise deploys the gateway configuration to your AWS IoT Greengrass core. You don't need to manually trigger a deployment.

**Configure a Modbus TCP source (CLI)**

You can define Modbus TCP data sources in a gateway capability. You must define all of your Modbus TCP sources in a single capability configuration.

For more information about defining sources with the AWS CLI, see the section called “Configuring data sources (AWS CLI)” (p. 127).

Note
You must install the AWS IoT SiteWise connector to use a Modbus TCP source.

This capability has the following versions.

<table>
<thead>
<tr>
<th>Version</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iotsitewise:modbuscollector:1</td>
</tr>
</tbody>
</table>

**Modbus TCP capability configuration parameters**

When you define Modbus TCP sources in a capability configuration, you must specify the following information in the `capabilityConfiguration` JSON document:

`sources`

A list of Modbus-TCP source definition structures that each contain the following information:

`name`

A unique, friendly name for the source.

`measurementDataStreamPrefix`

(Optional) A string to prepend to all data streams from the source. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.
destination

A destination structure that contains the following information:

type

The type of the destination.

streamName

The name of the AWS IoT Greengrass stream.

streamBufferSize

The size of the stream buffer.

endpoint

An endpoint structure that contains the following information:

ipAddress

The IP address of the Modbus TCP source.

port

(Optional) The port of the Modbus TCP source.

unitId

(Optional) The unitId. This defaults to a value of 1.

minimumInterRequestDuration

The minimum duration between each request in milliseconds.

propertyGroups

The list of property groups that define the tag definition requested by the protocol.

ame

The name of the property group. This should be a unique identifier.

tagPathDefinitions

The location of the measurement within the source. For example, the byte and word order, address, and transformation type. The structure of each MeasurementPathDefinition is defined by the connector.

scanMode

Defines the scan mode behavior and configurable parameters for the source.

Configure an Ethernet/IP (EIP) source

You can use the AWS IoT SiteWise console or a gateway capability to define and add an Ethernet IP source to your gateway. This source represents a local Ethernet IP server.

Note

- Gateways running on AWS IoT Greengrass V2 currently don't support Ethernet IP sources.
- You must install the AWS IoT SiteWise connector to use an Ethernet IP source.

Topics

- Configure an Ethernet/IP source (console) (p. 125)
- Configure an Ethernet/IP source (CLI) (p. 126)
- Configuring data sources (AWS CLI) (p. 127)
Configure an Ethernet/IP source (console)

To configure an Ethernet/IP source

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Gateways.
3. On the gateway you want to create a source for, choose Manage, and then choose View details.
4. Choose New source in the upper-right corner.
5. For Protocol options, choose Ethernet/IP (EIP).
6. For EtherNet/IP source configuration, enter a Name for the source.
7. For IP address, enter the IP address for the data source server.
8. (Optional) Enter the Port for the source server.
9. For Minimum inter-request duration, enter the time interval between subsequent requests sent to your server. Your gateway automatically calculates the minimum allowable interval based on your device and the number of registers you have.
10. For Property groups, enter a Name.
11. For Properties:
   a. For Tag, enter the property alias for your register set. For example, boiler.inlet.temperature.value.
   b. For Destination data type, choose the AWS IoT SiteWise data type that you want your data to be converted to. The default is String.
12. For Scan rate, update the rate at which you want the gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your gateway.
13. (Optional) For Destination, choose where the source data is sent. By default, your source sends data to AWS IoT SiteWise. You can use a AWS IoT Greengrass stream to export your data to a local destination or to the AWS Cloud instead.
   
   **Note**
   You must choose AWS IoT SiteWise as your source destination if you want to process data from this source at the edge with AWS IoT SiteWise. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

To send your data to another destination:

a. For Destination options, choose Other destinations.
b. For Greengrass stream name, enter the exact name of your AWS IoT Greengrass stream.

**Note**
You can use a stream that you've already created, or you can create a new AWS IoT Greengrass stream to export your data. If you want to use an existing stream, you must enter the exact name of the stream or a new stream will be created.
For more information about working with AWS IoT Greengrass streams, see Manage data streams in the AWS IoT Greengrass developer guide.
14. Choose **Add source**.

AWS IoT SiteWise deploys the gateway configuration to your AWS IoT Greengrass core. You don't need to manually trigger a deployment.

**Configure an Ethernet/IP source (CLI)**

You can define EIP data sources in a gateway capability. You must define all of your EIP sources in a single capability configuration.

For more information about defining sources with the AWS CLI, see the section called "Configuring data sources (AWS CLI)" (p. 127).

**Note**

You must install the AWS IoT SiteWise connector to use an Ethernet IP source.

This capability has the following versions.

<table>
<thead>
<tr>
<th>Version</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iotsitewise:eipcollector:1</td>
</tr>
</tbody>
</table>

**EIP capability configuration parameters**

When you define EIP sources in a capability configuration, you must specify the following information in the `capabilityConfiguration` JSON document:

```
sources
```

A list of EIP source definition structures that each contain the following information:

```
name
```

A unique, friendly name for the source. This can be up to 256 characters.

```
destinationPathPrefix
```

(Optional) A string to prepend to all data streams from the source. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.

```
destination
```

A destination structure that contains the following information:

```
type
```

The type of the destination.

```
streamName
```

The name of the AWS IoT Greengrass stream.

```
streamBufferSize
```

The size of the stream buffer.

```
endpoint
```

An endpoint structure that contains the following information:
ipAddress

The IP address of the EIP source.

port

(Optional) The port of the EIP source. Accepted values are numbers between 1 and 65535.

minimumInterRequestDuration

(Optional) The minimum duration between each request in milliseconds.

propertyGroups

The list of property groups that define the tag definition requested by the protocol. Each source can have one property group.

name

The name of the property group. This should be a unique identifier with a maximum length of 256 characters.

tagPathDefinitions

The list of structures specifying the data to collect from the Ethernet/IP device and how to transform it for output.

type

The type of the tagPathDefinition. For example, EIPTagPath.

path

The path of the tagPathDefinition. Each tag in a path can be a maximum length of 40 characters and can start with a letter or an underscore. Tags can't contain consecutive or trailing underscores. The path is prefixed with any value of destinationPathPrefix.

dstDataType

The data type to output the tag data. Accepted values are integer, double, string, and boolean.

scanMode

Defines the scan mode behavior and configurable parameters for the source.

type

The type of the scan mode behavior. Accepted values are POLL.

rate

The rate in milliseconds that the connector should read tags from the Ethernet/IP source.

**Configuring data sources (AWS CLI)**

You can use the AWS IoT SiteWise API and AWS Command Line Interface to add sources to your gateway. You define sources in gateway capabilities. A gateway capability represents a software feature that runs on the gateway, such as the capability to collect industrial data from OPC-UA sources.

Gateway capabilities have the following components:

- A configuration – A JSON document that defines all of the data sources for a capability.
A namespace – A unique string that identifies the type and version of a capability. For example, the OPC-UA source capability namespace is `iotsitewise:opcuacollector:version`, where `version` is the version of the OPC-UA capability. All OPC-UA sources are defined in one capability with this namespace.

A synchronization status – A status that indicates if a capability is synchronized between the AWS Cloud and the gateway. The sync status can be one of the following:
- **IN_SYNC** – The gateway is running the capability configuration.
- **OUT_OF_SYNC** – The gateway hasn't received the capability configuration.
- **SYNC_FAILED** – The gateway rejected the capability configuration.

After you update a capability configuration, its sync status is **OUT_OF_SYNC** until the gateway receives and applies or rejects the updated configuration.

Use the following operations to query and update your gateway sources and capability configurations:

- **DescribeGateway** – Retrieves information about a specific gateway. The response includes a list of capability summaries, including capability namespaces.
- **DescribeGatewayCapabilityConfiguration** – Retrieves the configuration of a specific capability. Use this operation to retrieve a capability configuration to update.
- **ListGateways** – Lists information about all gateways. The response includes a list of capability summaries for each gateway, including capability namespaces.
- **UpdateGatewayCapabilityConfiguration** – Updates a gateway capability configuration or defines a new capability configuration. This operation identifies capabilities by a capability namespace. If you provide a namespace that already exists, this operation updates the capability for that namespace. Otherwise, this operation creates a new capability.

**Warning**
The **UpdateGatewayCapabilityConfiguration** operation overwrites the existing capability configuration with the configuration that you provide in the payload. To avoid deleting your capability's configuration, you must add to the existing configuration when you update the capability.

**Gateway capabilities**

- ??? (p. 105)
- ??? (p. 123)
- ??? (p. 126)

**Choosing a destination for your source server data**

You can use destinations to control where to send your source's incoming data. You can either send your data to AWS IoT SiteWise, or you can use a AWS IoT Greengrass stream to send your data to a different location. You configure a different source destination for each source server in your gateway.

- **AWS IoT SiteWise** – Send your source data to AWS IoT SiteWise for storage and processing. This is the default option.

  **Note**
  You must choose AWS IoT SiteWise as your source destination if you want to process data from this source at the edge with AWS IoT SiteWise. For more information about processing data at the edge, see the section called “Enabling edge data processing” (p. 130).

- **AWS IoT Greengrass stream** – You choose a custom AWS IoT Greengrass stream to receive your source data. You can use the AWS IoT Greengrass stream to forward received data to an on-premises application, or to another AWS IoT service in the AWS Cloud.
You can choose an existing AWS IoT Greengrass stream for your source destination, or you can create a new one. For more information about how to choose a custom AWS IoT Greengrass stream as your destination, see Configuring data sources (p. 102).

The following example shows the required data stream message structure. All fields are required.

```json
{
   "alias" : "string",
   "messages" : [
      {
         "name": "string",
         "value": boolean|double|integer|string,
         "timestamp": number,
         "quality": "string"
      }
   ]
}
```

**alias**

The alias of the data stream. See the following example.

```
/company/windfarm/3/turbine/7/temperature
```

**messages**

The list of tuples in this batch. Each tuple contains a timestamp, value, and quality.

**name**

The alias of the data stream. This must match the alias.

**value**

The type of value contained in this message. Valid values are boolean, double, integer, or string.

**timestamp**

The timestamp of the tag data. This is formatted as the number of milliseconds since the Unix epoch.

**quality**

The quality of the tag data. Valid values are GOOD, BAD, or UNCERTAIN.

### Upgrading a connector

**Important**

Version 6 of the IoT SiteWise connector introduces new requirements: AWS IoT Greengrass Core software v1.10.0 and stream manager. Before you upgrade your connector, check that your gateway meets these requirements, or you won't be able to deploy your gateway.

You can easily upgrade your gateway's connector after a new IoT SiteWise connector version is released.

**Note**

In this procedure, you redeploy your Greengrass group and restart your gateway. Your gateway won't ingest data while it's restarting. The time to restart your gateway depends on the number of tags on your gateway's sources. Restart time can range from a few seconds (for a gateway with few tags) to several minutes (for a gateway with many tags).
To upgrade an IoT SiteWise connector

1. Navigate to the AWS IoT Greengrass console.
2. In the navigation pane, under Greengrass, choose Groups, and then choose the group that you created when you set up your gateway.
3. In the navigation pane, choose Connectors.
4. On the Connectors page, choose Available next to the IoT SiteWise connector.

If you don't see the Available element, your connector is already the latest version.

5. On the Upgrade connector page, enter your connector's parameters and then choose Upgrade.
6. In the upper-right corner, in the Actions menu, choose Deploy.
7. Choose Automatic detection to start the deployment.

If the deployment fails, choose Deploy again. If the deployment continues to fail, see AWS IoT Greengrass deployment troubleshooting.

Enabling edge data processing

You can use AWS IoT SiteWise to collect, organize, process, and monitor equipment data locally. You can use AWS IoT SiteWise so that you can use asset models and SiteWise Monitor on your local data. You
can process your data locally and send it to AWS IoT SiteWise, or load it into on-premise applications by using AWS IoT SiteWise API operations.

Because you can use AWS IoT SiteWise to process and route your data locally, you can choose to send only aggregated data to the AWS Cloud. Use this feature to optimize your bandwidth usage and cloud storage costs.

**Note**

- AWS IoT SiteWise retains your edge data on your gateways up to 30 days. The retention of your data is dependent on your device and the available disk space.
- 30 days after your gateway has been disconnected from the AWS Cloud, the data processing pack is automatically disabled.

**Topics**

- [Edge processing prerequisites](#)
- [Setting up edge capability](#)

## Edge processing prerequisites

Before you can use edge processing with AWS IoT SiteWise, you need to verify that your AWS account and devices are configured correctly.

Ensure that you have met the following prerequisites:

- An edge device. This device must meet the minimum configuration requirements of an x86 64 bit quad-core processor, 16 GB of RAM, and 256 GB in disk space. This device should be running Linux. In addition, the device must meet the following requirements:
  
  - Enabled inbound MQTT communication – Consult your device documentation for more information. For more information about enabling MQTT communication on your device, see the [MQTT documentation](#).
  
  - Java 8 – Run the following example command to verify your Java version.

        java8 --version

  
  - Python 3.7 – Run the following example command to verify your Python version

        python3.7 --version

  
  - Docker and docker-compose – Run the following example commands to verify your Docker and Docker-compose versions.

        docker --version
docker-compose --version

  
  - A Docker user – To create a home directory and a user for the deployment of the AWS IoT SiteWise Data Processor connector, do the following:

    1. Run the following command to create the home directory.

        sudo mkdir /sitewise_edge

    2. Run the following command to add the ggc_user user.

        sudo useradd -u ggc_user-id ggc_user
Enabling edge data processing

```bash
sudo usermod -aG docker ggc_user
sudo chown ggc_user-id /sitewise_edge
sudo chmod 700 /sitewise_edge
```

**Note**
- Replace `ggc_user-id` with the ID of the `ggc_user` user.
- Run the following command to find the ID of the `ggc_user` user.

```
id -u ggc_user
```

- Choose one of the following options:
  - If you use AWS IoT SiteWise Data Processor connector version 1, 2, 3, or 4, ports 443, 8443, and 8883 must be externally accessible on your device.
  - If you use AWS IoT SiteWise Data Processor connector version 5, ports 443 and 8883 must be externally accessible on your device.
  - An AWS IoT Greengrass service role attached to your AWS account and the AWS IoT Core console. This role must have permissions that allow your AWS IoT Greengrass device to perform AWS IoT SiteWise operations and access Amazon Elastic Container Registry (Amazon ECR).

**Note**
You need to re-deploy your AWS IoT SiteWise connectors after updating your service role policy. For more information, see *Introducing AWS IoT SiteWise* in the AWS official blog.

To grant permissions to the role:

1. Navigate to the IAM console and search for the role that is associated with your gateway Greengrass group.
2. Attach an inline policy to this role.

The following example service role policy grants access to AWS IoT SiteWise, Amazon ECR, and the AWS IoT SiteWise Data Processor connector.

```json
{
    "Version": "2012-10-17",
    "Statement": [
    
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage"
            ],
            "Resource": "*"
        },

        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        },

        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotsitewise:*",
            "Resource": "*"
        }

    ]
}
```
• A configured and deployed AWS IoT SiteWise gateway with the data processing pack enabled. For more information about setting up your gateway, see the section called “Configuring a gateway” (p. 88).

For more information about the prerequisites you need to use edge processing, see Introducing AWS IoT SiteWise in the AWS official blog.

**Setting up edge capability**

To use edge processing, you must configure your AWS IoT SiteWise gateway and asset model for the edge. Your gateway ingests data from your source server, and sends that data to the destination of your choice. Your asset model controls specifies where your assets are stored and computed.

**Note**
Before you begin, make sure that you meet the Edge processing prerequisites (p. 131).

You must complete the following steps to use edge processing. You don't need to complete these steps in order, because AWS IoT SiteWise automatically syncs your the AWS Cloud with your gateway for you every 10 minutes.

For more information about getting started with edge processing, see Introducing AWS IoT SiteWise in the AWS official blog.

• **Add the data processing pack to your gateway** – Add the data processing pack so that your gateway can communicate with all asset models that you configured for the edge. You add this pack when you add your gateway to AWS IoT SiteWise, or when you edit an existing gateway. For more information about your AWS IoT SiteWise gateway, see the section called “Using a gateway” (p. 87).

After you add the data processing pack to your gateway, you must configure and add the AWS IoT SiteWise Data Processor connector to your AWS IoT Greengrass group. For more information about adding the AWS IoT SiteWise Data Processor connector to your AWS IoT Greengrass group, see the configuration step (p. 101) in the section called “Adding the gateway to AWS IoT SiteWise” (p. 100).

**Note**
You need v9 of the data collection pack to use the data processing pack on your gateway.

• **Configure your source destination to AWS IoT SiteWise** – This specifies where your gateway source sends your data. You configure this when you add a source to your gateway, or when you edit an existing source. To process data at the edge, you must choose AWS IoT SiteWise as your source destination. For more information about source destinations, see the section called "Choosing a destination for your source server data" (p. 128).

• **Configure your asset model for the edge** – Your asset model edge configuration specifies where your assets properties are computed. You can compute all properties at the edge, or you can configure your asset model properties separately.

Asset model properties include metrics, transforms, and measurements:

• Metrics are the asset's aggregated data over a specified period of time. You can compute new metrics by using existing metric data. AWS IoT SiteWise always sends your metrics to the AWS Cloud for long-term storage. AWS IoT SiteWise computes metrics on the AWS Cloud by default. You can configure your asset model to compute your metrics at the edge. AWS IoT SiteWise sends processed results to the AWS Cloud.

• Transforms are mathematical expressions that map an asset property's data points from one form to another. Transforms can use metrics as input data and must be computed and stored at the same location as their inputs. If you configure a metric input to compute at the edge, AWS IoT SiteWise also computes its associated transform at the edge.

• Measurements are formatted as raw data that your device collects and sends to the AWS Cloud by default. You can configure your asset model to store this data on your local device.
For more information about asset properties, see the section called “Defining data properties” (p. 150).

After you create your asset model, you can then configure it for the edge. For more information about configuring your asset model for the edge, see the section called “Creating an asset model (console)” (p. 143).

**Note**

Asset models and dashboards are automatically synced between the AWS Cloud and your AWS IoT SiteWise gateway every 10 minutes. You can also sync manually from the local gateway application.

---

**Upgrading gateways from Greengrass V1 to Greengrass V2**

If you have gateways that run on AWS IoT Greengrass V1, you can upgrade your gateways to AWS IoT Greengrass V2. For more information, see Instructions for upgrading AWS IoT SiteWise gateways from AWS IoT Greengrass V1 to AWS IoT Greengrass V2.

**Managing gateways**

You can use the AWS IoT SiteWise console, APIs, or the AWS OpsHub for AWS IoT SiteWise application to manage gateways.

We highly recommend that you use the AWS OpsHub for AWS IoT SiteWise application to monitor the disk usage on your gateway device. Make sure that your device has enough space for upcoming data. When you're about to run out of space on your gateway device, the service automatically deletes a small amount of data with the oldest timestamps to make room for upcoming data.

To check if the service deleted your data, do the following:

1. Sign in to the AWS OpsHub for AWS IoT SiteWise application.
2. Choose **Settings**
3. For **Logs**, specify a time range, and then choose **Download**.
4. Unzip the log file.
5. If the log file contains the following message, the service deleted your data.

   ```
   number bytes of data have been deleted to prevent gateway storage from running out of space.
   ```

**Managing gateways using AWS OpsHub for AWS IoT SiteWise**

You use the AWS OpsHub for AWS IoT SiteWise application to manage and monitor your gateways. This application provides the following monitoring and management options:

- Under **Overview**, you can do the following:
  - View gateway details that help you get insights into your gateway device data, identify issues, and improve the gateway's performance.
Managing your gateway with the AWS IoT SiteWise console

You can use the AWS IoT SiteWise console to configure, update, and monitor all gateways in your AWS account.

You can view your AWS IoT SiteWise gateways by navigating to the Gateways page in the AWS IoT SiteWise console. The AWS IoT SiteWise console provides the following monitoring and management options:

- Update data source configuration and configure additional data sources

Important
Your gateway device and the AWS OpsHub application must be connected to the same network.

To manage gateways using AWS OpsHub

1. Download and install the AWS OpsHub for AWS IoT SiteWise for Windows application.
2. Open the application.
3. You can sign in to your gateway with your Linux or Lightweight Directory Access Protocol (LDAP) credentials. To sign in to your gateway, do one of the following:
   
   Linux
   1. For Hostname or IP address, enter the hostname or IP address of your gateway device.
   2. For Authentication, choose Linux.
   3. For User name, enter the user name of your Linux operating system.
   4. For Password, enter the password of your Linux operating system.
   5. Choose Sign in.

   LDAP
   1. For Hostname or IP address, enter the hostname or IP address of your gateway device.
   2. For Authentication, choose LDAP.
   3. For User name, enter your LDAP's user name.
   4. For Password, enter your LDAP's password.
   5. Choose Sign in.
• View the number of data points ingested per data source
• Add data packs to your gateway
• View the connectivity status of your gateways
• View the gateway sync status of resources and configuration changes

Accessing your gateway using Linux credentials

Besides Lightweight Directory Access Protocol (LDAP), you can use the Linux credentials to access your gateway. The following steps assume that you use a device with Ubuntu. If you use a different Linux distribution, consult the relevant documentation for your device.

Important
To access your gateway with Linux credentials, you must enable the data processing pack for your gateway.

To create a Linux user pool

1. To create an admin group, run the following command.

   ```bash
   sudo groupadd --system SWE_ADMIN_GROUP
   ```

   Users in the SWE_ADMIN_GROUP group can allow admin access for the gateway.

2. To create a user group, run the following command.

   ```bash
   sudo groupadd --system SWE_USER_GROUP
   ```

   Users in the SWE_USER_GROUP group can allow ready-only access for the gateway.

3. To add a user to the admin group, run the following command. Replace `user-name` with the user name that you want to add.

   ```bash
   sudo adduser --system user-name && sudo adduser user-name SWE_ADMIN_GROUP
   ```

4. To create a password, run the following command. Replace `user-name` and `password` with the user name that you added in the previous step and the password that you want to create.

   ```bash
   sudo passwd user-name password
   ```

You can now use the user name and password to sign in to the gateway on the AWS OpsHub for AWS IoT SiteWise application.

Managing the gateway certificate

You can use SiteWise Monitor and third-party applications, such as Grafana on your gateway devices. These applications require a TLS connection to the service. Gateways currently use a service-signed certificate. If you use a browser to open the applications, such as a SiteWise Monitor portal, you might receive a warning for untrusted certificate.

The following shows how to download the trusted certificate from the AWS OpsHub for AWS IoT SiteWise application.

1. Sign in to the application.
2. Choose Settings.
3. For **Authentication**, choose **Download certificate**.

The following assumes that you use Google Chrome or FireFox. If you use a different browser, consult the relevant documentation for your browser. To add the certificate that you downloaded in the previous step to a browser, do one of the following:

- If you use Google Chrome, follow the Set up certificates in the **Google Chrome Enterprise Help documentation**.
- If you use Firefox, follow the To Load the Certificate into the Mozilla or Firefox Browser in the **Oracle documentation**.

**Processing data locally**

You must configure your asset model for the edge before you can process your gateway data at the edge. Your asset model edge configuration specifies where your assets properties are computed. You can compute all properties at the edge, or you can configure your asset model properties separately.

Asset model properties include metrics, transforms, and measurements:

- **Metrics** are the asset's aggregated data over a specified period of time. You can compute new metrics by using existing metric data. AWS IoT SiteWise always sends your metrics to the AWS Cloud for long-term storage. AWS IoT SiteWise computes metrics on the AWS Cloud by default. You can configure your asset model to compute your metrics at the edge. AWS IoT SiteWise sends processed results to the AWS Cloud.

- **Transforms** are mathematical expressions that map an asset property's data points from one form to another. Transforms can use metrics as input data and must be computed and stored at the same location as their inputs. If you configure a metric input to compute at the edge, AWS IoT SiteWise also computes its associated transform at the edge.

- **Measurements** are formatted as raw data that your device collects and sends to the AWS Cloud by default. You can configure your asset model to store this data on your local device.

For more information about asset properties, see the section called “Defining data properties” (p. 150).

After you create your asset model, you can then configure it for the edge. For more information about configuring your asset model for the edge, see the section called “Creating an asset model (console)” (p. 143).

**Note**

Asset models and dashboards are automatically synced between the AWS Cloud and your AWS IoT SiteWise gateway every 10 minutes. You can also sync manually from the local gateway application (p. 134).

You can use the AWS IoT SiteWise REST APIs and the AWS Command Line Interface (AWS CLI) to query your gateway for data at the edge. Before you query your gateway for data at the edge, you must meet the following prerequisites:

- Your credentials must be set for the REST APIs. For more information about setting credentials, see the section called “Managing gateways” (p. 134).
- The SDK endpoint must point to the IP address of your gateway. You can find more information in the documentation for your SDK. For example, see Specifying Custom Endpoints in the **AWS SDK for Java 2.x Developer Guide**.
- Your gateway certificate must be registered. You can find more information about registering your gateway certificate in the documentation for your SDK. For example, see the **Registering Certificate Bundles in Node.js** in the **AWS SDK for Java 2.x Developer Guide**.
For more information about querying data with AWS IoT SiteWise, see *Querying asset property data (p. 285).*
Modeling industrial assets

You can create virtual representations of your industrial operation with AWS IoT SiteWise assets. An asset represents a device, a piece of equipment, or a process that uploads one or more data streams to the AWS Cloud. For example, an asset device can be a wind turbine that sends air temperature, propeller rotation speed, and power output time-series measurements to asset properties in AWS IoT SiteWise. Each data stream corresponds to unique property alias. For example, the alias /company/windfarm/3/turbine/7/temperature uniquely identifies the temperature data stream coming from turbine #7 in wind farm #3. You can configure AWS IoT SiteWise assets to transform incoming measurement data using mathematical expressions, such as to convert temperature data from Celsius to Fahrenheit.

An asset can also represent a logical grouping of devices, such as an entire wind farm. You can associate assets to other assets to create asset hierarchies that represent complex industrial operations. Assets can access the data within their associated child assets so that you can use AWS IoT SiteWise expressions to calculate aggregate metrics, such as the net power output of a wind farm.

You must create every asset from an asset model. Asset models are declarative structures that standardize the format of your assets. Asset models enforce consistent information across multiple assets of the same type, so that you can process data in assets that represent groups of devices. In the preceding diagram, you use the same asset model for all three turbines because all turbines share a common set of properties.

After you define your asset models, you can create your industrial assets. To create an asset, select an ACTIVE asset model to create an asset from that model. Then, you can populate asset-specific information such as data stream aliases and attributes. In the preceding diagram, you create three turbine assets from one asset model and then associate data stream aliases like /company/windfarm/3/turbine/7/temperature for each turbine.

You can also update and delete existing assets and asset models. When you update an asset model, every asset based on that asset model reflects any changes that you make to the underlying model.

Topics

- Asset and model states (p. 140)
- Creating asset models (p. 143)
- Creating assets (p. 199)
- Mapping industrial data streams to asset properties (p. 202)
- Updating attribute values (p. 204)
- Associating and disassociating assets (p. 207)
Asset and model states

When you create, update, or delete an asset or an asset model, the changes take time to propagate. AWS IoT SiteWise resolves these operations asynchronously and updates the status of each resource. Each asset and asset model has a status field that contains the state of the resource and any error message, if applicable. The state can be one of the following values:

- **ACTIVE** – The asset or asset model is active. This is the only state in which you can query and interact with assets and asset models.
- **CREATING** – The asset or asset model is being created.
- **UPDATING** – The asset or asset model is being updated.
- **DELETING** – The asset or asset model is being deleted.
- **PROPAGATING** – (Asset models only) The asset model’s changes are propagating to all of its assets.
- **FAILED** – The asset or asset model failed to validate during a create or update operation, possibly due to a circular reference in an expression. You can delete assets and asset models that are in the FAILED state.

Some of the create, update, and delete operations in AWS IoT SiteWise place an asset or asset model in a state other than ACTIVE while the operation resolves. If you need to query or interact with an asset or asset model after you perform one of these operations, you must wait until the state changes to ACTIVE. Otherwise, your requests fail.

**Topics**
- Checking the status of an asset (p. 140)
- Checking the status of an asset model (p. 142)

Checking the status of an asset

You can use the AWS IoT SiteWise console or API to check the status of an asset.

**Topics**
- Checking the status of an asset (console) (p. 140)
- Checking the status of an asset (CLI) (p. 141)

Checking the status of an asset (console)

Use the following procedure to check the status of an asset in the AWS IoT SiteWise console.

**To check the status of an asset (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to check.

**Tip**
You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Find **Status** in the **Asset details** panel.

Checking the status of an asset (CLI)

You can use the AWS Command Line Interface (AWS CLI) to check the status of an asset.

To check the status of an asset, use the `DescribeAsset` operation with the `assetId` parameter.

To check the status of an asset (CLI)

- Run the following command to describe the asset. Replace `asset-id` with the asset's ID.

  ```bash
  aws iotsitewise describe-asset --asset-id asset-id
  ```

  The operation returns a response that contains the asset's details. The response contains an `assetStatus` object that has the following structure.

  ```json
  {
  ...
  "assetStatus": {
    "state": "String",
    "error": {
      "code": "String",
      "message": "String"
    }
  }
  }
  ```
The asset's state is in `assetStatus.state` in the JSON object.

Checking the status of an asset model

You can use the AWS IoT SiteWise console or API to check the status of an asset model.

Topics
- Checking the status of an asset model (console) (p. 142)
- Checking the status of an asset model (CLI) (p. 142)

Checking the status of an asset model (console)

Use the following procedure to check the status of an asset model in the AWS IoT SiteWise console.

To check the status of an asset model (console)
1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Models.
3. Choose the asset model to check.
4. Find Status in the Details panel.

Checking the status of an asset model (CLI)

You can use the AWS CLI to check the status of an asset model.

To check the status of an asset model, use the DescribeAssetModel operation with the `assetModelId` parameter.

To check the status of an asset model (CLI)
- Run the following command to describe the asset model. Replace `asset-model-id` with the asset model's ID.

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id
```

The operation returns a response that contains the asset model's details. The response contains an `assetModelStatus` object that has the following structure.
Creating asset models

AWS IoT SiteWise asset models drive standardization of your industrial data. An asset model contains a name, description, asset properties, and asset hierarchy definitions. For example, you can define a wind turbine model with temperature, rotations per minute (RPM), and power properties. Then, you can define a wind farm model with a net power output property and a wind turbine hierarchy definition.

**Note**
We recommend that you model your operation starting with the lowest-level nodes. For example, create your wind turbine model before you create your wind farm model. Asset hierarchy definitions contain references to existing asset models. With this approach, you can define asset hierarchies as you create your models.

The following sections describe how to use the AWS IoT SiteWise console or API to create asset models. The following sections also describe the different types of asset properties and asset hierarchies that you can use to create models.

**Topics**
- Creating an asset model (console) (p. 143)
- Creating an asset model (CLI) (p. 144)
- Example asset models (p. 145)
- Defining data properties (p. 150)
- Defining relationships between assets (hierarchies) (p. 198)

Creating an asset model (console)

You can use the AWS IoT SiteWise console to create an asset model. The AWS IoT SiteWise console provides various features, such as formula autocompletion, that can help you define valid asset models.

**To create an asset model (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose **Create model**.
4. On the **Create model** page, do the following:
   a. Enter a **Name** for the asset model, such as **Wind Turbine** or **Wind Turbine Model**. This name must be unique across all models in your account in this Region.
   b. (Optional) Add **Attribute definitions** for the model. Attributes represent information that rarely changes. For more information, see **Defining static data (attributes)** (p. 151).
c. (Optional) Add **Measurement definitions** for the model. Measurements represent data streams from your equipment. For more information, see Defining data streams from equipment (measurements) (p. 152).

d. (Optional) Add **Transform definitions** for the model. Transforms are formulas that map data from one form to another. For more information, see Transforming data (transforms) (p. 154).

e. (Optional) Add **Metric definitions** for the model. Metrics are formulas that aggregate data over time intervals. Metrics can input data from associated assets, so that you can calculate values that represent your operation or a subset of your operation. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).

f. (Optional) Add **Hierarchy definitions** for the model. Hierarchies are relationships between assets. For more information, see Defining relationships between assets (hierarchies) (p. 198).

g. (Optional) Add tags for the asset model. For more information, see Tagging your AWS IoT SiteWise resources (p. 369).

h. Choose **Create model**.

When you create an asset model, the AWS IoT SiteWise console navigates to the new model's page. On this page, you can see the model's **Status**, which is initially **CREATING**. This page automatically updates, so you can wait for the model's status to update.

**Note**
The asset model creation process can take up to a few minutes for complex models. After the asset model status is **ACTIVE**, you can use the asset model to create assets. For more information, see Asset and model states (p. 140).

5. (Optional) After you create your asset model, you can configure your asset model for the edge. For more information about Sitewise Edge, see the section called “Enabling edge data processing” (p. 130).

   a. On the model page, choose **Configure for Edge**.

   b. On the model configuration page, choose the edge configuration for your model. This controls where AWS IoT SiteWise can compute and store properties associated with this asset model. For more information about configuring your model for the edge, see the section called “Setting up edge capability” (p. 133).

   c. For **Custom edge configuration**, choose the location that you want AWS IoT SiteWise to compute and store each of your asset model properties.

      **Note**
      Transforms and metrics that are associated must be configured for the same location. For more information about configuring your model for the edge, see the section called “Setting up edge capability” (p. 133).

   d. Choose **Save**. On the model page, your **Edge configuration** should now be **Configured**.

### Creating an asset model (CLI)

You can use the AWS Command Line Interface (AWS CLI) to create an asset model.

Use the **CreateAssetModel** operation to create an asset model with properties and hierarchies. This operation expects a payload with the following structure.

```json
{
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": Array of AssetModelProperty,
  "assetModelHierarchies": Array of AssetModelHierarchyDefinition
}
```
To create an asset model (CLI)

1. Create a file called `asset-model-payload.json` and then copy the following JSON object into the file.

   ```json
   {
   "assetModelName": "",
   "assetModelDescription": "",
   "assetModelProperties": [],
   "assetModelHierarchies": [],
   "assetModelCompositeModels": []
   }
   ```

2. Use your preferred JSON text editor to edit the `asset-model-payload.json` file for the following:

   a. Enter a name (`assetModelName`) for the asset model, such as Wind Turbine or Wind Turbine Model. This name must be unique across all models in your account in this Region.
   
   b. (Optional) Enter a description (`assetModelDescription`) for the asset model, or remove the `assetModelDescription` key-value pair.
   
   c. (Optional) Define asset properties (`assetModelProperties`) for the model. For more information, see Defining data properties (p. 150).
   
   d. (Optional) Define asset hierarchies (`assetModelHierarchies`) for the model. For more information, see Defining relationships between assets (hierarchies) (p. 198).
   
   e. (Optional) Define alarms for the model. Alarms monitor other properties so that you can identify when equipment or processes require attention. Each alarm definition is a composite model (`assetModelCompositeModels`) that standardizes the set of properties that the alarm uses. For more information, see Monitoring data with alarms (p. 220) and Defining alarms on asset models (p. 223).
   
   f. (Optional) Add tags (`tags`) for the asset model. For more information, see Tagging your AWS IoT SiteWise resources (p. 369).

3. Run the following command to create an asset model from the definition in the JSON file.

   ```
   aws iotsitewise create-asset-model --cli-input-json file://asset-model-payload.json
   ```

   The operation returns a response that contains the `assetModelId` that you refer to when creating an asset. The response also contains the state of the model (`assetModelStatus.state`), which is initially CREATING. The asset model's status is CREATING until the changes propagate.

   **Note**
   
   The asset model creation process can take up to a few minutes for complex models. To check the current status of your asset model, use the DescribeAssetModel operation by specifying the `assetModelId`. After the asset model status is ACTIVE, you can use the asset model to create assets. For more information, see Asset and model states (p. 140).

**Example asset models**

This section contains example asset models definitions that you can use to create asset models with the AWS CLI and AWS IoT SiteWise SDKs. These asset models represent a wind turbine and a wind farm.
Wind turbine assets ingest raw sensor data and calculate values such as power and average wind speed. Wind farm assets calculate values such as total power for all wind turbines in the wind farm.

Topics

- Wind turbine asset model (p. 146)
- Wind farm asset model (p. 149)

Wind turbine asset model

The following asset model represents a turbine in a wind farm. The wind turbine ingests sensor data to calculate values such as power and average wind speed.

**Note**

This example model resembles the wind turbine model from the AWS IoT SiteWise demo. For more information, see Using the AWS IoT SiteWise demo (p. 10).

```json
{
  "assetModelName": "Wind Turbine Asset Model",
  "assetModelDescription": "Represents a turbine in a wind farm.",
  "assetModelProperties": [
    {
      "name": "Location",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "Renton"
        }
      }
    },
    {
      "name": "Make",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "Amazon"
        }
      }
    },
    {
      "name": "Model",
      "dataType": "INTEGER",
      "type": {
        "attribute": {
          "defaultValue": "500"
        }
      }
    },
    {
      "name": "Torque (KiloNewton Meter)",
      "dataType": "DOUBLE",
      "unit": "kNm",
      "type": {
        "measurement": {}
      }
    },
    {
      "name": "Wind Direction",
      "dataType": "DOUBLE",
      "unit": "Degrees",
      "type": {
        "measurement": {}
      }
    }
  ]
}
```
{  
  "name": "RotationsPerMinute",
  "dataType": "DOUBLE",
  "unit": "RPM",
  "type": {
    "measurement": {}
  }
},
{  
  "name": "Wind Speed",
  "dataType": "DOUBLE",
  "unit": "m/s",
  "type": {
    "measurement": {}
  }
},
{  
  "name": "RotationsPerSecond",
  "dataType": "DOUBLE",
  "unit": "RPS",
  "type": {
    "transform": {
      "expression": "rpm / 60",
      "variables": [
        {
          "name": "rpm",
          "value": {
            "propertyId": "RotationsPerMinute"
          }
        }
      ]
    }
  }
},
{  
  "name": "Overdrive State",
  "dataType": "DOUBLE",
  "type": {
    "transform": {
      "expression": "gte(torque, 3)",
      "variables": [
        {
          "name": "torque",
          "value": {
            "propertyId": "Torque (KiloNewton Meter)"
          }
        }
      ]
    }
  }
},
{  
  "name": "Average Power",
  "dataType": "DOUBLE",
  "unit": "Watts",
  "type": {
    "metric": {
      "expression": "avg(torque) * avg(rps) * 2 * 3.14",
      "variables": [
        {
          "name": "torque",
          "value": {
            "propertyId": "Torque (Newton Meter)"
          }
        }]
    }
  }
}
{  
  "name": "rps",
  "value": {
    "propertyId": "RotationsPerSecond"
  }
},
"window": {
  "tumbling": {
    "interval": "5m"
  }
}
],
"window": {
  "tumbling": {
    "interval": "5m"
  }
}
},
"name": "Average Wind Speed",
"dataType": "DOUBLE",
"unit": "m/s",
"type": {
  "metric": {
    "expression": "avg(windspeed)",
    "variables": [
      {
        "name": "windspeed",
        "value": {
          "propertyId": "Wind Speed"
        }
      }
    ],
    "window": {
      "tumbling": {
        "interval": "5m"
      }
    }
  }
}
},
"name": "Torque (Newton Meter)",
"dataType": "DOUBLE",
"unit": "Nm",
"type": {
  "transform": {
    "expression": "knm * 1000",
    "variables": [
      {
        "name": "knm",
        "value": {
          "propertyId": "Torque (KiloNewton Meter)"
        }
      }
    ]
  }
}
},
"name": "Overdrive State Time",
"dataType": "DOUBLE",
"unit": "Seconds",
"type": {
  "metric": {
    "expression": "statetime(overdrive_state)",
    "variables": [
      {
        "name": "overdrive_state",
        "value": {
          "propertyId": "overdrive_state"
        }
      }
    ]
  }
}
Wind farm asset model

The following asset model represents a wind farm that comprises multiple wind turbines. This asset model defines a hierarchy (p. 198) to the wind turbine model. This enables the wind farm to calculate values (such as average power) from data for all wind turbines in the wind farm.

**Note**
This example model resembles the wind farm model from the AWS IoT SiteWise demo. For more information, see Using the AWS IoT SiteWise demo (p. 10).

This asset model depends on the Wind turbine asset model (p. 146). Replace the `propertyId` and `childAssetModelId` values with those from an existing wind turbine asset model.
Defining data properties

Asset properties are the structures within each asset that contain asset data. Asset properties can be any of the following types:

- **Attributes** – An asset's generally static properties, such as device manufacturer or geographic region. For more information, see Defining static data (attributes) (p. 151).
• **Measurements** – An asset's raw device's sensor data streams, such as timestamped rotation speed values or timestamped temperature values in Celsius. A measurement is defined by a data stream alias. For more information, see Defining data streams from equipment (measurements) (p. 152).

• **Transforms** – An asset's transformed time-series values, such as timestamped temperature values in Fahrenheit. A transform is defined by an expression and the variables to consume with that expression. For more information, see Transforming data (transforms) (p. 154).

• **Metrics** – An asset's data aggregated over a specified time interval, such as the hourly average temperature. A metric is defined by a time interval, an expression, and the variables to consume with that expression. Metric expressions can input associated assets' metric properties, so that you can calculate metrics that represent your operation or a subset of your operation. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).

For an example of how to use measurements, transforms, and metrics to calculate Overall Equipment Effectiveness (OEE), see Calculating OEE in AWS IoT SiteWise (p. 13).

**Topics**

- Defining static data (attributes) (p. 151)
- Defining data streams from equipment (measurements) (p. 152)
- Transforming data (transforms) (p. 154)
- Aggregating data from properties and other assets (metrics) (p. 157)
- Using formula expressions (p. 163)

**Defining static data (attributes)**

Asset attributes represent information that is generally static, such as device manufacturer or geographic location. Each asset that you create from an asset model contains the attributes of that model.

**Topics**

- Defining attributes (console) (p. 151)
- Defining attributes (CLI) (p. 152)

**Defining attributes (console)**

When you define an attribute for an asset model in the AWS IoT SiteWise console, you specify the following parameters:

- **Name** – The property's name.
- **Default value** – (Optional) The default value for this attribute. Assets created from the model have this value for the attribute. For more information about how to override the default value in an asset created from a model, see Updating attribute values (p. 204).
- **Data type** – The property's data type, which is one of the following:
  - **String** – A string with up to 1024 bytes.
  - **Integer** – A signed 32-bit integer with range \([-2,147,483,648, 2,147,483,647]\).
  - **Double** – A floating point number with range \([-10^{100}, 10^{100}]\) and IEEE 754 double precision.
  - **Boolean** – true or false.

For more information, see Creating an asset model (console) (p. 143).
Example Example attribute definition

The following example demonstrates an attribute that represents an asset's model number with a default value.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model number</td>
<td>BLT123</td>
<td>STRING</td>
</tr>
</tbody>
</table>

Defining attributes (CLI)

When you define an attribute for an asset model with the AWS IoT SiteWise API, you specify the following parameters:

- **name** – The property's name.
- **defaultValue** – (Optional) The default value for this attribute. Assets created from the model have this value for the attribute. For more information about how to override the default value in an asset created from a model, see Updating attribute values (p. 204).
- **dataType** – The property's data type, which is one of the following:
  - **STRING** – A string with up to 1024 bytes.
  - **INTEGER** – A floating point number with range \([-10^{100}, 10^{100}]\) and IEEE 754 double precision.
  - **DOUBLE** – A signed 32-bit integer with range \([-2,147,483,648, 2,147,483,647]\).
  - **BOOLEAN** – true or false.

Example Example attribute definition

The following example demonstrates an attribute that represents an asset's model number with a default value. This object is an example of an AssetModelProperty that contains an Attribute. You can specify this object as a part of the CreateAssetModel request payload to create an attribute property. For more information, see Creating an asset model (CLI) (p. 144).

```json
{
  ...
  "assetModelPropertyProperties": [
    {
      "name": "Model number",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "BLT123"
        }
      }
    },
    ...
  ]
}
```

Defining data streams from equipment (measurements)

A measurement represents a device's raw sensor data stream, such as timestamped temperature values or timestamped rotations per minute (RPM) values.
Defining data properties

Topics
- Defining measurements (console) (p. 153)
- Defining measurements (CLI) (p. 153)

Defining measurements (console)

When you define a measurement for an asset model in the AWS IoT SiteWise console, you specify the following parameters:

- **Name** – The property's name.
- **Unit** – (Optional) The scientific unit for the property, such as mm or Celsius.
- **Data type** – The property's data type, which is one of the following:
  - **String** – A string with up to 1024 bytes.
  - **Integer** – A signed 32-bit integer with range \([-2,147,483,648, 2,147,483,647]\).
  - **Double** – A floating point number with range \([-10^{100}, 10^{100}]\) and IEEE 754 double precision.
  - **Boolean** – `true` or `false`.

For more information, see Creating an asset model (console) (p. 143).

Example Example measurement definition

The following example demonstrates a measurement that represents an asset's temperature sensor readings.

![Measurement definitions](image)

Defining measurements (CLI)

When you define a measurement for an asset model with the AWS IoT SiteWise API, you specify the following parameters:

- **name** – The property's name.
- **dataType** – The property's data type, which is one of the following:
  - **STRING** – A string with up to 1024 bytes.
  - **INTEGER** – A floating point number with range \([-10^{100}, 10^{100}]\) and IEEE 754 double precision.
  - **DOUBLE** – A signed 32-bit integer with range \([-2,147,483,648, 2,147,483,647]\).
  - **BOOLEAN** – `true` or `false`.
- **unit** – (Optional) The scientific unit for the property, such as mm or Celsius.

Example Example measurement definition

The following example demonstrates a measurement that represents an asset's temperature sensor readings. This object is an example of an `AssetModelProperty` that contains a `Measurement`. You can
specify this object as a part of the CreateAssetModel request payload to create a measurement property. For more information, see Creating an asset model (CLI) (p. 144).

The Measurement structure is an empty structure when you define an asset model because you later configure each asset to use unique device data streams. For more information about how to connect an asset's measurement property to a device's sensor data stream, see the Mapping industrial data streams to asset properties (p. 202).

```json
{
  ...
  "assetModelProperties": [ 
    {
      "name": "Temperature C",
      "dataType": "DOUBLE",
      "type": {
        "measurement": {},
        "unit": "Celsius"
      }
    },
    ...
  }
}
```

**Transforming data (transforms)**

Transforms are mathematical expressions that map asset properties' data points from one form to another. A transform expression consists of asset property variables, literals, operators, and functions. The transformed data points hold a one-to-one relationship with the input data points. AWS IoT SiteWise calculates a new transformed data point each time any of the input properties receives a new data point.

For example, if your asset has a temperature measurement stream named Temperature_C with units in Celsius, you can convert each data point to Fahrenheit with the formula Temperature_F = 9/5 * Temperature_C + 32. Each time AWS IoT SiteWise receives a data point in the Temperature_C measurement stream, the corresponding Temperature_F value is calculated within a few seconds and available as the Temperature_F property.

If your transform contains more than one variable, the data point that arrives earlier initiates the computation immediately. Consider an example where a parts manufacturer uses a transform to monitor product quality. Using a different standard based on the part type, the manufacturer uses the following measurements to represent the process:

- Part_Number - A string that identifies the part type.
- Good_Count - An integer that increases by one if the part meets the standard.
- Bad_Count - An integer that increases by one if the part doesn't meet the standard.

The manufacturer also creates a transform, Quality_Monitor, that equals if(eq(Part_Number, "BLT123") and (Bad_Count / (Good_Count + Bad_Count) > 0.1), "Caution", "Normal").

This transform monitors the percentage of bad parts produced for a specific part type. If the part number is BLT123 and the percentage of bad parts exceeds 10 percent (0.1), the transform returns "Caution". Otherwise, the transform returns "Normal".

**Note**

- If Part_Number receives a new data point before other measurements, the Quality_Monitor transform uses the new Part_Number value and the latest Good_Count
Defining data properties

and Bad_Count values. To avoid errors, reset Good_Count and Bad_Count before the next manufacturing run.

- Use metrics (p. 157) if you want to evaluate expressions only after all variables receive new data points.

Topics

- Defining transforms (console) (p. 155)
- Defining transforms (CLI) (p. 156)

Defining transforms (console)

When you define a transform for an asset model in the AWS IoT SiteWise console, you specify following parameters:

- **Name** – The property's name.
- **Formula** – The transform expression. Transform expressions can't use aggregation functions or temporal functions. Start typing or press the down arrow key to open the autocomplete feature. For more information, see Using formula expressions (p. 163).

  **Important**
  
  Transforms can input properties that are integer, double, Boolean, or string type. Booleans convert to 0 (false) and 1 (true).
  
  Transforms must input one or more properties that aren't attributes and any number of attribute properties. AWS IoT SiteWise calculates a new transformed data point each time the input property that isn't an attribute receives a new data point. New attribute values don't trigger transform updates.
  
  Formula expressions can only output double or string values. Nested expressions can output other data types, such as strings, but the formula as a whole must evaluate to a number or string. You can use the `jp` function (p. 171) to convert a string to a number. The Boolean value must be 1 (true) or 0 (false). For more information, see Undefined, infinite, and overflow values (p. 197).

  - **Unit** – (Optional) The scientific unit for the property, such as mm or Celsius.
  - **Data type** – The data type of the transform, which can be **Double** or **String**.

  For more information, see Creating an asset model (console) (p. 143).

Example transform definition

The following example demonstrates a transform property that converts an asset's temperature measurement data from Celsius to Fahrenheit.

Example transform definition that contains three variables

The following example demonstrates a transform property that returns a warning message ("Caution") if more than 10 percent of the BLT123 parts don't meet the standard. Otherwise, it returns an information message ("Normal").
Defining transforms (CLI)

When you define a transform for an asset model with the AWS IoT SiteWise API, you specify the following parameters:

- **name** – The property's name.
- **dataType** – The data type of the transform, which must be **DOUBLE** or **STRING**.
- **expression** – The transform expression. Transform expressions can't use aggregation functions or temporal functions. For more information, see Using formula expressions (p. 163).
- **variables** – The list of variables that defines the other properties of your asset to use in the expression. Each variable structure contains a simple name to use in the expression and a value structure that identifies which property to link to that variable. The value structure contains the following information:
  - **propertyId** – The ID of the property from which to input values. You can use the property's name instead of its ID.

  **Important**
  
  Transforms can input properties that are integer, double, Boolean, or string type. Booleans convert to 0 (false) and 1 (true).
  
  Transforms must input one or more properties that aren't attributes and any number of attribute properties. AWS IoT SiteWise calculates a new transformed data point each time the input property that isn't an attribute receives a new data point. New attribute values don’t trigger transform updates.
  
  Formula expressions can only output double or string values. Nested expressions can output other data types, such as strings, but the formula as a whole must evaluate to a number or string. You can use the **jp function** (p. 171) to convert a string to a number. The Boolean value must be 1 (true) or 0 (false). For more information, see Undefined, infinite, and overflow values (p. 197).

- **unit** – (Optional) The scientific unit for the property, such as mm or Celsius.

Example transform definition

The following example demonstrates a transform property that converts an asset's temperature measurement data from Celsius to Fahrenheit. This object is an example of an **AssetModelProperty** that contains a **Transform**. You can specify this object as a part of the **CreateAssetModel** request payload to create a transform property. For more information, see Creating an asset model (CLI) (p. 144).

```json
{
  ...
  "assetModelProperties": [
    {
      "name": "Temperature F",
      "dataType": "DOUBLE",
      "type": {
        "transform": {
          "expression": "9/5 * temp_c + 32",
          "variables": [
            {
```
"name": "temp_c",
"value": {
"propertyId": "Temperature C"
}
],
},
"unit": "Fahrenheit"
],
...}

Example transform definition that contains three variables

The following example demonstrates a transform property that returns a warning message ("Caution") if more than 10 percent of the BLT123 parts don't meet the standard. Otherwise, it returns an information message ("Normal").

{
...
"assetModelProperties": [
...
{
"name": "Quality_Monitor",
"dataType": "STRING",
"type": {
"transform": {
"expression": "if(eq(Part_Number,"BLT123") and (Bad_Count / (Good_Count + Bad_Count) > 0.1), "Caution", "Normal")",
"variables": [
{
"name": "Part_Number",
"value": {
"propertyId": "Part Number"
}
},
{
"name": "Good_Count",
"value": {
"propertyId": "Good Count"
}
},
{
"name": "Bad_Count",
"value": {
"propertyId": "Bad Count"
}
]}
}]
...}

Aggregating data from properties and other assets (metrics)

Metrics are mathematical expressions that use aggregation functions to process all input data points and output a single data point per specified time interval. For example, a metric can calculate the average hourly temperature from a temperature data stream.
Metrics can input data from associated assets' metrics, so you can calculate statistics that provide
insight to your operation or a subset of your operation. For example, a metric can calculate the average
hourly temperature across all wind turbines in a wind farm. For more information about how to define
associations between assets, see Defining relationships between assets (hierarchies) (p. 198).

Metrics can also input data from other properties without aggregating data over each time interval. If
you specify an attribute (p. 151) in a formula, AWS IoT SiteWise uses the latest (p. 175) value for that
attribute when it computes the formula. If you specify a metric in a formula, AWS IoT SiteWise uses the
last (p. 175) value for the time interval over which it computes the formula. This means you can define
metrics like $OEE = Availability \times Quality \times Performance$, where Availability, Quality,
and Performance are all other metrics on the same asset model.

AWS IoT SiteWise also automatically computes a set of basic aggregation metrics for all asset properties.
To reduce computation costs, you can use these aggregates instead of defining custom metrics for basic
computations. For more information, see Querying asset property aggregates (p. 288).

Topics
- Defining metrics (console) (p. 158)
- Defining metrics (CLI) (p. 160)

Defining metrics (console)

When you define a metric for an asset model in the AWS IoT SiteWise console, you specify the following
parameters:

- **Name** – The property's name.
- **Formula** – The metric expression. Metric expressions can use aggregation functions (p. 173) to input
data from a property for all associated assets in a hierarchy. Start typing or press the down arrow key
to open the autocomplete feature. For more information, see Using formula expressions (p. 163).

  **Important**
  Metrics can only be properties that are integer, double, Boolean, or string type. Booleans
  convert to 0 (false) and 1 (true).
  If you define any metric input variables in a metric's expression, those inputs must have the
  same time interval as the output metric.
  Formula expressions can only output double or string values. Nested expressions can output
  other data types, such as strings, but the formula as a whole must evaluate to a number or
  string. You can use the `jp` function (p. 171) to convert a string to a number. The Boolean
  value must be 1 (true) or 0 (false). For more information, see Undefined, infinite, and overflow
  values (p. 197).
- **Data type** – The data type of the transform, which can be **Double** or **String**.
- **Time interval** – The metric time interval. AWS IoT SiteWise supports the following tumbling window
time intervals, where each interval starts when the previous one ends:
  - **1 minute** – 1 minute, computed at the end of each minute (12:00:00 AM, 12:01:00 AM, 12:02:00 AM,
    and so on).
  - **5 minutes** – 5 minutes, computed at the end of every five minutes starting on the hour (12:00:00
    AM, 12:05:00 AM, 12:10:00 AM, and so on).
  - **15 minutes** – 15 minutes, computed at the end of every fifteen minutes starting on the hour
    (12:00:00 AM, 12:15:00 AM, 12:30:00 AM, and so on).
  - **1 hour** – 1 hour (60 minutes), computed at the end of every hour in UTC (12:00:00 AM, 01:00:00 AM,
    02:00:00 AM, and so on).
  - **1 day** – 1 day (24 hours), computed at the end of every day in UTC (12:00:00 AM Monday, 12:00:00
    AM Tuesday, and so on).
  - **1 week** – 1 week (7 days), computed at the end of every Sunday in UTC (every 12:00:00 AM Monday).
• **Custom interval** – You can enter any time interval between a minute and a week.

• **Offset date** – (Optional) The reference date from which to aggregate data.

• **Offset time** – (Optional) The reference time from which to aggregate data. The offset time must be between 00:00:00 and 23:59:59.

• **Offset time zone** – (Optional) The time zone for the offset. If it isn't specified, the default offset time zone is the Universal Coordinated Time (UTC).

**Supported time zones**
- (UTC+00:00) Universal Coordinated Time
- (UTC+01:00) European Central Time
- (UTC+02:00) Eastern European
- (UTC+03+:00) Eastern African Time
- (UTC+04:00) Near East Time
- (UTC+05:00) Pakistan Lahore Time
- (UTC+05:30) India Standard Time
- (UTC+06:00) Bangladesh Standard Time
- (UTC+07:00) Vietnam Standard Time
- (UTC+08:00) China Taiwan Time
- (UTC+09:00) Japan Standard Time
- (UTC+09:30) Australia Central Time
- (UTC+10:00) Australia Eastern Time
- (UTC+11:00) Solomon Standard Time
- (UTC+12:00) New Zealand Standard Time
- (UTC-11:00) Midway Islands Time
- (UTC-10:00) Hawaii Standard Time
- (UTC-09:00) Alaska Standard Time
- (UTC-08:00) Pacific Standard Time
- (UTC-07:00) Phoenix Standard Time
- (UTC-06:00) Central Standard Time
- (UTC-05:00) Eastern Standard Time
- (UTC-04:00) Puerto Rico and US Virgin Islands Time
- (UTC-03:00) Argentina Standard Time
- (UTC-02:00) South Georgia Time
- (UTC-01:00) Central African Time

**Example custom time interval with an offset (console)**

The following example shows you how to define a 12-hour time interval with an offset on February 20, 2021, at 6:30:30 PM (PST).

**To define a custom interval with an offset**

1. For **Time interval**, choose **Custom interval**.
2. For **Time interval**, do one of the following:
   - Enter **12**, and then choose **hours**.
   - Enter **720**, and then choose **minutes**.
   - Enter **43200**, and then choose **seconds**.
Defining data properties

Important
The **Time interval** must be an integer regardless of the unit.

3. For **Offset date**, choose **2021/02/20**.
4. For **Offset time**, enter **18:30:30**.
5. For **Offset timezone**, choose **(UTC-08:00) Pacific Standard Time**.

If you create the metric on July 1, 2021, before or at 06:30:30 PM (PST), you get the first aggregation result on July 1, 2021, at 06:30:30 PM (PST), the second aggregation result on July 2, 2021, at 06:30:30 AM (PST), and so on.

Example Example metric definition

The following example demonstrates a metric property that aggregates an asset's temperature data to calculate maximum hourly temperature.

Example Example metric definition that inputs data from associated assets

The following example demonstrates a metric property that aggregates multiple wind turbines' average power data to calculate total average power for a wind farm.

Defining metrics (CLI)

When you define a metric for an asset model with the AWS IoT SiteWise API, you specify the following parameters:

- **name** – The property's name.
- **dataType** – The data type of the metric, which can be **DOUBLE** or **STRING**.
- **expression** – The metric expression. Metric expressions can use aggregation functions (p. 173) to input data from a property for all associated assets in a hierarchy. For more information, see Using formula expressions (p. 163).
- **window** – The time interval and offset for the metric's tumbling window, where each interval starts when the previous one ends:
  - **interval** – The time interval for the tumbling window. The time interval must be between a minute and a week.
• offsets – The offset for the tumbling window.

For more information, see TumblingWindow in the AWS IoT SiteWise API Reference.

Example custom time interval with an offset (AWS CLI)

The following example shows you how to define a 12-hour time interval with an offset on February 20, 2021, at 06:30:30 PM (PST).

```json
{
    "window": {
        "tumbling": {
            "interval": "12h",
            "offset": "2021-07-23T18:30:30-08"
        }
    }
}
```

If you create the metric on July 1, 2021, before or at 06:30:30 PM (PST), you get the first aggregation result on July 1, 2021, at 06:30:30 PM (PST), the second aggregation result on July 2, 2021, at 06:30:30 AM (PST), and so on.

• variables – The list of variables that defines the other properties of your asset or child assets to use in the expression. Each variable structure contains a simple name for use in the expression and a value structure that identifies which property to link to that variable. The value structure contains the following information:
  • propertyId – The ID of the property from which to pull values. You can use the property's name instead of its ID if the property is defined in the current model (rather than defined in a model from a hierarchy).
  • hierarchyId – (Optional) The ID of the hierarchy from which to query child assets for the property. You can use the hierarchy definition’s name instead of its ID. If you omit this value, AWS IoT SiteWise finds the property in the current model.

  Important
  Metrics can only be properties that are integer, double, Boolean, or string type. Booleans convert to 0 (false) and 1 (true).
  If you define any metric input variables in a metric’s expression, those inputs must have the same time interval as the output metric.
  Formula expressions can only output double or string values. Nested expressions can output other data types, such as strings, but the formula as a whole must evaluate to a number or string. You can use the `jp function (p. 171)` to convert a string to a number. The Boolean value must be 1 (true) or 0 (false). For more information, see Undefined, infinite, and overflow values (p. 197).

• unit – (Optional) The scientific unit for the property, such as mm or Celsius.

Example Example metric definition

The following example demonstrates a metric property that aggregates an asset's temperature measurement data to calculate maximum hourly temperature in Fahrenheit. This object is an example of an AssetModelProperty that contains a Metric. You can specify this object as a part of the CreateAssetModel request payload to create a metric property. For more information, see Creating an asset model (CLI) (p. 144).

```json
{
    ...
    "assetModelProperty": [
        ...
        {
```
"name": "Max temperature",
"dataType": "DOUBLE",
"type": {
  "metric": {
    "expression": "max(temp_f)",
    "variables": [
      {
        "name": "temp_f",
        "value": {
          "propertyId": "Temperature F"
        }
      }
    ],
    "window": {
      "tumbling": {
        "interval": "1h"
      }
    },
    "unit": "Fahrenheit"
  }
},
...

### Example Example metric definition that inputs data from associated assets

The following example demonstrates a metric property that aggregates multiple wind turbines' average power data to calculate total average power for a wind farm. This object is an example of an AssetModelProperty that contains a Metric. You can specify this object as a part of the CreateAssetModel request payload to create a metric property. For more information, see Creating an asset model (CLI) (p. 144).

```json
...
  "assetModelProperties": [
    {
      "name": "Total Average Power",
      "dataType": "DOUBLE",
      "type": {
        "metric": {
          "expression": "avg(power)",
          "variables": [
            {
              "name": "power",
              "value": {
                "propertyId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
                "hierarchyId": "Turbine Asset Model"
              }
            }
          ],
          "window": {
            "tumbling": {
              "interval": "5m"
            }
          },
          "unit": "kWh"
        }
      }
    }
  ],
...
```
Using formula expressions

With formula expressions, you can define the mathematical functions to transform and aggregate your raw industrial data to gain insights about your operation. For more information about how to define asset properties that use formula expressions, see Transforming data (transforms) (p. 154) and Aggregating data from properties and other assets (metrics) (p. 157). Transforms and metrics are formula properties.

Topics
- Variables (p. 163)
- Literals (p. 163)
- Operators (p. 164)
- Constants (p. 166)
- Functions (p. 167)
- Formula expression tutorials (p. 195)

Variables

Variables represent AWS IoT SiteWise asset properties in formula expressions. Use variables to input values from other asset properties in your expressions, so that you can process data from constant properties (attributes (p. 151)), raw data streams (measurements (p. 152)), and other formula properties.

Variables can represent asset properties from the same asset model or from associated child asset models. Only metric formulas can input variables from child asset models.

You identify variables by different names in the console and the API.

- AWS IoT SiteWise console – Use asset property names as variables in your expressions.
- AWS IoT SiteWise API (AWS CLI, AWS SDKs) – Define variables with the ExpressionVariable structure, which requires a variable name and a reference to an asset property. The variable name can contain lowercase letters, numbers, and underscores. Then, use variable names to reference asset properties in your expressions.

Variable names are case sensitive.

For more information, see Defining transforms (p. 154) and Defining metrics (p. 157).

Literals

You can define number and string literals in formula expressions.

- Numbers
  
  Use numbers and scientific notation to define integers and doubles. You can use E notation to express numbers with scientific notation.

  Examples: 1, 2.0, -9, -23.1, 7.89e3, 3.4E-5

- Strings
Use the ' (quote) and " (double quote) characters to define strings. The quote type for the start and end must match. To escape a quote that matches the one that you use to declare a string, include that quote character twice. This is the only escape character in AWS IoT SiteWise strings.

Examples: 'active', "inactive", '{"temp": 52}', '{"temp": "high"}"

Operators

You can use the following common operators in formula expressions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>If both operands are numbers, this operator adds the left and right operands. If either operand is a string, this operator concatenates the left and right operands as strings. For example, the expression 1 + 2 + &quot; is three&quot; evaluates to &quot;3 is three&quot;. The concatenated string can have up to 1024 characters. If the string exceeds 1024 characters, then AWS IoT SiteWise doesn't output a data point for that computation.</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts the right operand from the left operand. You can only use this operator with numeric operands.</td>
</tr>
<tr>
<td>/</td>
<td>Divides the left operand by the right operand. You can only use this operator with numeric operands.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies the left and right operands. You can only use this operator with numeric operands.</td>
</tr>
<tr>
<td>^</td>
<td>Raises the left operand to the power of the right operand (exponentiation). You can only use this operator with numeric operands.</td>
</tr>
<tr>
<td>%</td>
<td>Returns the remainder from dividing the left operand by the right operand. The result has the same sign as the left operand. This behavior differs from the modulo operation. You can only use this operator with numeric operands.</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>Returns 1 if x is less than y, otherwise 0.</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>Returns 1 if x is greater than y, otherwise 0.</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>x &lt;= y</code></td>
<td>Returns 1 if <code>x</code> is less than or equal to <code>y</code>, otherwise 0.</td>
</tr>
<tr>
<td><code>x &gt;= y</code></td>
<td>Returns 1 if <code>x</code> is greater than or equal to <code>y</code>, otherwise 0.</td>
</tr>
<tr>
<td><code>x == y</code></td>
<td>Returns 1 if <code>x</code> is equal to <code>y</code>, otherwise 0.</td>
</tr>
<tr>
<td><code>x != y</code></td>
<td>Returns 1 if <code>x</code> is not equal to <code>y</code>, otherwise 0.</td>
</tr>
</tbody>
</table>
| `!x`     | Returns 1 if `x` is evaluated to 0 (false), otherwise 0.  
          | `x` is evaluated to false if:  
          | • `x` is a numeric operand and it's evaluated to 0.  
          | • `x` is evaluated to an empty string.  
          | • `x` is evaluated to an empty array.  
          | • `x` is evaluated to `None`. |
| `x and y`| Returns 0 if `x` is evaluated to 0 (false). Otherwise, returns the evaluated result of `y`.  
          | `x` or `y` is evaluated to false if:  
          | • `x` or `y` is a numeric operand and it's evaluated to 0.  
          | • `x` or `y` is evaluated to an empty string.  
          | • `x` or `y` is evaluated to an empty array.  
          | • `x` or `y` is evaluated to `None`. |
| `x or y` | Returns 1 if `x` is evaluated to 1 (true). Otherwise, returns the evaluated result of `y`.  
          | `x` or `y` is evaluated to false if:  
          | • `x` or `y` is a numeric operand and it's evaluated to 0.  
          | • `x` or `y` is evaluated to an empty string.  
          | • `x` or `y` is evaluated to an empty array.  
          | • `x` or `y` is evaluated to `None`. |
| `not x`  | Returns 1 if `x` is evaluated to 0 (false), otherwise 0.  
          | `x` is evaluated to false if:  
          | • `x` is a numeric operand and it's evaluated to 0.  
          | • `x` is evaluated to an empty string.  
          | • `x` is evaluated to an empty array.  
<pre><code>      | • `x` is evaluated to `None`. |
</code></pre>
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Returns the character at an index \texttt{index} of the string \texttt{s}. This is equivalent to the index syntax in Python.</td>
</tr>
<tr>
<td>\texttt{s[index]}</td>
<td></td>
</tr>
<tr>
<td>[]</td>
<td>Returns a slice of the string \texttt{s}. This is equivalent to the slice syntax in Python. This operator has the following arguments:</td>
</tr>
<tr>
<td>\texttt{s[start:end:step]}</td>
<td></td>
</tr>
</tbody>
</table>

\textbf{Example Examples}

- "Hello!"[1] returns e.
- "Hello!"[-2] returns o.

\textbf{Example Examples}

- "Hello!"[1:4] returns "ell".
- "Hello!"[:2] returns "He".
- "Hello!"[3:] returns "lo!".
- "Hello!"[:4] returns "He".
- "Hello!"[::2] returns "Hlo".
- "Hello!"[::-1] returns "!olleH".

\section*{Constants}

You can use the following common mathematical constants in your expressions. All constants are case insensitive.

\textbf{Note}

If you define a variable with the same name as a constant, the variable overrides the constant.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{pi}</td>
<td>The number pi (#): 3.141592653589793</td>
</tr>
<tr>
<td>Constant</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>e</td>
<td>The number e: 2.718281828459045</td>
</tr>
<tr>
<td>true</td>
<td>Equivalent to the number 1. In AWS IoT SiteWise, Booleans convert to their number equivalents.</td>
</tr>
<tr>
<td>false</td>
<td>Equivalent to the number 0. In AWS IoT SiteWise, Booleans convert to their number equivalents.</td>
</tr>
<tr>
<td>none</td>
<td>Equivalent to no value. You can use this constant to output nothing as the result of a conditional expression (p. 170).</td>
</tr>
</tbody>
</table>

**Functions**

You can use the following functions to operate on data in your formula expressions.

Transforms and metrics support different functions. The following table indicates which types of functions are compatible with each type of formula property.

<table>
<thead>
<tr>
<th>Function type</th>
<th>Transforms</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common functions (p. 168)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>Comparison functions (p. 169)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>Conditional functions (p. 170)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>String functions (p. 171)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>Aggregation functions (p. 173)</td>
<td>❌ No</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>Temporal functions (p. 174)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
<tr>
<td>Date and time functions (p. 177)</td>
<td>✔ Yes</td>
<td>✔ Yes</td>
</tr>
</tbody>
</table>

**Function syntax**

You can use the following syntax to create functions:

**Regular syntax**

With the regular syntax, the function name is followed by parentheses with zero or more arguments.

`function_name(argument1, argument2, argument3, ...).` For example, functions with the regular syntax might look like `log(x)` and `contains(s, substring).`

**Uniform function call syntax (UFCS)**

UFCS enables you to call functions using the syntax for method calls in object-oriented programming. With UFCS, the first argument is followed by dot (.), then the function name and the remaining arguments (if any) inside parentheses.

`argument1.function_name(argument2, argument3, ...).` For example, functions with UFCS might look like `x.log()` and `s.contains(substring).`
You can also use UFCS to chain subsequent functions. AWS IoT SiteWise uses the evaluation result of the current function as the first argument for the next function.

For example, you can use `message.jp('$.status').lower().contains('fail')` instead of `contains(lower(jp(message, '$.status')),'fail')`.

For more information, visit the D Programming Language website.

**Note**
You can use UFCS for all AWS IoT SiteWise functions. AWS IoT SiteWise functions are not case sensitive. For example, you can use `lower(s)` and `Lower(s)` interchangeably.

### Common functions

In transforms (p. 154) and metrics (p. 157), you can use the following functions to calculate common mathematical functions in transforms and metrics.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(x)</code></td>
<td>Returns the absolute value of x.</td>
</tr>
<tr>
<td><code>acos(x)</code></td>
<td>Returns the arccosine of x.</td>
</tr>
<tr>
<td><code>asin(x)</code></td>
<td>Returns the arcsine of x.</td>
</tr>
<tr>
<td><code>atan(x)</code></td>
<td>Returns the arctangent of x.</td>
</tr>
<tr>
<td><code>cbrt(x)</code></td>
<td>Returns the cubic root of x.</td>
</tr>
<tr>
<td><code>ceil(x)</code></td>
<td>Returns the nearest integer greater than x.</td>
</tr>
<tr>
<td><code>cos(x)</code></td>
<td>Returns the cosine of x.</td>
</tr>
<tr>
<td><code>cosh(x)</code></td>
<td>Returns the hyperbolic cosine of x.</td>
</tr>
<tr>
<td><code>cot(x)</code></td>
<td>Returns the cotangent of x.</td>
</tr>
<tr>
<td><code>exp(x)</code></td>
<td>Returns e to the power of x.</td>
</tr>
<tr>
<td><code>expm1(x)</code></td>
<td>Returns <code>exp(x) - 1</code>. Use this function to more accurately calculate <code>exp(x) - 1</code> for small values of x.</td>
</tr>
<tr>
<td><code>floor(x)</code></td>
<td>Returns the nearest integer less than x.</td>
</tr>
<tr>
<td><code>log(x)</code></td>
<td>Returns the ( \log_e ) (base e) of x.</td>
</tr>
<tr>
<td><code>log10(x)</code></td>
<td>Returns the ( \log_{10} ) (base 10) of x.</td>
</tr>
<tr>
<td><code>log1p(x)</code></td>
<td>Returns ( \log(1 + x) ). Use this function to more accurately calculate ( \log(1 + x) ) for small values of x.</td>
</tr>
<tr>
<td><code>log2(x)</code></td>
<td>Returns the ( \log_2 ) (base 2) of x.</td>
</tr>
<tr>
<td><code>pow(x, y)</code></td>
<td>Returns x to the power of y. This is equivalent to ( x^y ).</td>
</tr>
<tr>
<td><code>signum(x)</code></td>
<td>Returns the sign of x (-1 for negative inputs, 0 for zero inputs, +1 for positive inputs).</td>
</tr>
</tbody>
</table>
### Function Definitions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin(x)</td>
<td>Returns the sine of x.</td>
</tr>
<tr>
<td>sinh(x)</td>
<td>Returns the hyperbolic sine of x.</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Returns the square root of x.</td>
</tr>
<tr>
<td>tan(x)</td>
<td>Returns the tangent of x.</td>
</tr>
<tr>
<td>tanh(x)</td>
<td>Returns the hyperbolic tangent of x.</td>
</tr>
</tbody>
</table>

### Comparison Functions

In transforms (p. 154) and metrics (p. 157), you can use the following comparison functions to compare two values and output 1 (true) or 0 (false). AWS IoT SiteWise compares strings by lexicographic order.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gt(x, y)</td>
<td>Returns 1 if x is greater than y, otherwise 0 (x &gt; y).</td>
</tr>
<tr>
<td></td>
<td>This function doesn't return a value if x and y are incompatible types, such as a number and a string.</td>
</tr>
<tr>
<td>gte(x, y)</td>
<td>Returns 1 if x is greater than or equal to y, otherwise 0 (x ≥ y).</td>
</tr>
<tr>
<td></td>
<td>AWS IoT SiteWise considers the arguments equal if they are within a relative tolerance of 1E-9. This behaves similar to the isclose function in Python.</td>
</tr>
<tr>
<td></td>
<td>This function doesn't return a value if x and y are incompatible types, such as a number and a string.</td>
</tr>
<tr>
<td>eq(x, y)</td>
<td>Returns 1 if x is equal to y, otherwise 0 (x == y).</td>
</tr>
<tr>
<td></td>
<td>AWS IoT SiteWise considers the arguments equal if they are within a relative tolerance of 1E-9. This behaves similar to the isclose function in Python.</td>
</tr>
<tr>
<td></td>
<td>This function doesn't return a value if x and y are incompatible types, such as a number and a string.</td>
</tr>
<tr>
<td>neq(x, y)</td>
<td>Returns 1 if x is not equal to y, otherwise 0 (x != y).</td>
</tr>
<tr>
<td></td>
<td>This function doesn't consider any tolerances.</td>
</tr>
<tr>
<td></td>
<td>This function doesn't return a value if x and y are incompatible types, such as a number and a string.</td>
</tr>
<tr>
<td>lt(x, y)</td>
<td>Returns 1 if x is less than y, otherwise 0 (x &lt; y).</td>
</tr>
</tbody>
</table>
### Defining data properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>This function doesn’t return a value if x and y are incompatible types, such as a number and a string.</strong></td>
<td></td>
</tr>
<tr>
<td><code>lte(x, y)</code></td>
<td>Returns 1 if x is less than or equal to y, otherwise 0 ((x \leq y)). AWS IoT SiteWise considers the arguments equal if they are within a relative tolerance of 1E-9. This behaves similar to the <code>isclose</code> function in Python.</td>
</tr>
<tr>
<td><strong>This function doesn’t return a value if x and y are incompatible types, such as a number and a string.</strong></td>
<td></td>
</tr>
<tr>
<td><code>isnan(x)</code></td>
<td>Returns 1 if x is equal to NaN, otherwise 0.</td>
</tr>
<tr>
<td><strong>This function doesn’t return a value if x is a string.</strong></td>
<td></td>
</tr>
</tbody>
</table>

#### Conditional functions

In transforms (p. 154) and metrics (p. 157), you can use the following function to check a condition and return different results whether the condition evaluates to true or false.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `if(condition, result_if_true, result_if_false)` | Evaluates the condition and returns `result_if_true` if the condition evaluates to true or `result_if_false` if the condition evaluates to false.  
  
  *condition* must be a number. This function considers 0 and an empty string as false and everything else (including NaN) as true. Booleans convert to 0 (false) and 1 (true).  
  
  You can return the `none constant (p. 167)` from this function to discard the output for a particular condition. This means you can filter out data points that don’t meet a condition. For more information, see Filtering data points (p. 196).  
  
  **Example Examples**  
  
  - `if(0, x, y)` returns the variable y.  
  - `if(5, x, y)` returns the variable x.  
  - `if(gt(temp, 300), x, y)` returns the variable x if the variable temp is greater than 300.  
  - `if(gt(temp, 300), temp, none)` returns the variable temp if it’s greater than or equal to 300, or `none` (no value) if temp is less than 300. |
### Defining data properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(s)</td>
<td>Returns the length of the string s.</td>
</tr>
<tr>
<td>find(s, substring)</td>
<td>Returns the index of the string substring in the string s.</td>
</tr>
<tr>
<td>contains(s, substring)</td>
<td>Returns 1 if the string s contains the string substring, otherwise 0.</td>
</tr>
<tr>
<td>upper(s)</td>
<td>Returns the string s in uppercase form.</td>
</tr>
<tr>
<td>lower(s)</td>
<td>Returns the string s in lowercase form.</td>
</tr>
<tr>
<td>jp(s, json_path)</td>
<td>Evaluates the string s with the JsonPath expression json_path and returns the result.</td>
</tr>
</tbody>
</table>

We recommend that you use UFCS for nested conditional functions where one or more arguments are conditional functions. You can use `if(condition, result_if_true)` to evaluate a condition and `elif(condition, result_if_true, result_if_false)` to evaluate additional conditions.

For example, you can use `if(condition1, result1_if_true).elif(condition2, result2_if_true, result2_if_false)` instead of `if(condition1, result1_if_true, if(condition2, result2_if_true, result2_if_false))`.

**Important**  
You must use `elif(condition, result_if_true, result_if_false)` with UFCS.

### String functions

In [transforms](p. 154) and [metrics](p. 157), you can use the following functions to operate on strings. For more information, see [Using strings in formulas](p. 196).

**Important**  
Formula expressions can only output double or string values. Nested expressions can output other data types, such as strings, but the formula as a whole must evaluate to a number or string. You can use the **jp function** (p. 171) to convert a string to a number. The Boolean value must be 1 (true) or 0 (false). For more information, see [Undefined, infinite, and overflow values](p. 197).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(s)</td>
<td>Returns the length of the string s.</td>
</tr>
<tr>
<td>find(s, substring)</td>
<td>Returns the index of the string substring in the string s.</td>
</tr>
<tr>
<td>contains(s, substring)</td>
<td>Returns 1 if the string s contains the string substring, otherwise 0.</td>
</tr>
<tr>
<td>upper(s)</td>
<td>Returns the string s in uppercase form.</td>
</tr>
<tr>
<td>lower(s)</td>
<td>Returns the string s in lowercase form.</td>
</tr>
<tr>
<td>jp(s, json_path)</td>
<td>Evaluates the string s with the JsonPath expression json_path and returns the result.</td>
</tr>
</tbody>
</table>

Use this function to do the following:

- Extract a value, array, or object from a serialized JSON structure.
- Convert a string to a number. For example, the formula `jp('111', '#')` returns 111 as a number.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `jp`     | To extract a string value from a JSON structure and return it as a number, you must use multiple nested `jp` functions. The outer `jp` function extracts the string from the JSON structure, and the inner `jp` function converts the string to a number. The string `json_path` must contain a string literal. This means that `json_path` can't be an expression that evaluates to a string. Example Examples  
  - `jp('{"status":"active","value":15}'  
    
    - `$json_path` returns 15.  
    - `jp({"measurement":  
        "reading":25,"confidence":0.95}'  
      
    - `$json_path` returns 25.  
    - `jp('[2,8,23]', '[$[2]'  
      
    - `json_path` returns 23.  
    - `jp({"values": [3,6,7]'  
      
    - `$json_path` returns 6.  
    - `jp('111', '[$'  
      
    - `json_path` returns 111.  
    - `jp(jp('{"measurement":  
        "reading":25,"confidence":0.95}'  
      
    - `$json_path` returns 0.95.  |
| `join`   | Returns a concatenated string with a delimiter. This function uses the first input string as a delimiter and joins the remaining input strings together. This behaves similar to the `join(CharSequence delimiter, CharSequence... elements)` function in Java. Example Examples  
  - `join("-", "aa", "bb", "cc") returns aa-bb-cc` |
| `format` | Returns a string in the specified format. This function evaluates expression to a value, and then returns the value in the specified format. This behaves similar to the `format(String format, Object... args)` function in Java. For more information about supported formats, see Conversions under Class Formatter in the Java Platform, Standard Edition 7 API Specification. Example Examples  
  - `format(100+1: "d") returns an integer, 101.  
  - `format("The result is %d", 100+1)` returns a string, The result is 101. |
### Defining data properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f'expression'</code></td>
<td>Returns a concatenated string. With this formatted function, you can use a simple expression to concatenate and format strings. These functions may contain nested expressions. You can use <code>{}</code> (curly braces) to interpolate expressions. This behaves similar to the formatted string literals in Python.</td>
</tr>
</tbody>
</table>

**Example Examples**

- `f'abc{1+2: "f"}d'` returns `abc3.000000d`.

  To evaluate this example expression, do the following:
  1. `format(1+2: "f")` returns a floating point number, `3.000000`.
  2. `join('', "abc", 1+2, 'd')` returns a string, `abc3.000000d`.

  You can also write the expression in the following way: `join('', "abc", format(1+2: "f"), 'd')`.

### Aggregation functions

In **metrics** (p. 157) only, you can use the following functions that aggregate input values over each time interval and calculate a single output value. Aggregation functions can aggregate data from associated assets.

Aggregation function arguments can be **variables** (p. 163), **number literals** (p. 163), **temporal functions** (p. 174), or aggregation functions. This means that you can't provide nested expressions as arguments to aggregation functions. For example, the formula `avg(x + 1)` isn't valid. By contrast, the formula `max(latest(x), latest(y), latest(z))` is valid and returns the largest current value of the `x`, `y`, and `z` properties.

You can use nested expressions in aggregation functions. When you use nested expressions, the following rules apply:

- Each argument can have only one variable.

  **Example**

  For example, `avg(x*(x-1))` and `sum(x/2 )/avg(y^2 )` are supported.

  For example, `min(x/y)` isn't supported.

- Each argument can have multilevel nested expressions.

  **Example**

  For example, `sum(avg(x^2 )/2)` is supported.

- Different arguments can have different variables.

  **Example**

  For example, `sum(x/2, y*2)` is supported.
AWS IoT SiteWise User Guide
Defining data properties

**Note**

- If your expressions contain measurements, AWS IoT SiteWise uses the last values over the current time interval for the measurements to compute aggregates.
- If your expressions contain attributes, AWS IoT SiteWise uses the latest values for the attributes to compute aggregates.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avg(x₀, ..., xₙ)</code></td>
<td>Returns the mean of the given variables' values over the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function outputs a data point only if the given variables have at least one data point over the current time interval.</td>
</tr>
<tr>
<td><code>sum(x₀, ..., xₙ)</code></td>
<td>Returns the sum of the given variables' values over the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function outputs a data point only if the given variables have at least one data point over the current time interval.</td>
</tr>
<tr>
<td><code>min(x₀, ..., xₙ)</code></td>
<td>Returns the minimum of the given variables' values over the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function outputs a data point only if the given variables have at least one data point over the current time interval.</td>
</tr>
<tr>
<td><code>max(x₀, ..., xₙ)</code></td>
<td>Returns the maximum of the given variables' values over the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function outputs a data point only if the given variables have at least one data point over the current time interval.</td>
</tr>
<tr>
<td><code>count(x₀, ..., xₙ)</code></td>
<td>Returns the total number of data points for the given variables over the current time interval. For more information about how to count the number of data points that meet a condition, see Counting data points that match a condition (p. 196).</td>
</tr>
<tr>
<td></td>
<td>This function computes a data point for every time interval.</td>
</tr>
</tbody>
</table>

**Temporal functions**

You can use temporal functions to return values based on timestamps of data points.

**Using temporal functions in metrics**

In metrics (p. 157) only, you can use the following functions that return values based on timestamps of data points.
Temporal function arguments must be properties from the local asset model. This means that you can't use properties from child asset models in temporal functions. You also can't use expressions as arguments to temporal functions.

You can use nested expressions in temporal functions. When you use nested expressions, the following rules apply:

- Each argument can have only one variable.

  For example, `latest( t*9/5 + 32 )` is supported.

- Arguments can't be aggregation functions.

  For example, `first( sum(x) )` isn't supported.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>first(x)</code></td>
<td>Returns the given variable's value with the earliest timestamp over the current time interval.</td>
</tr>
<tr>
<td><code>last(x)</code></td>
<td>Returns the given variable's value with the latest timestamp over the current time interval.</td>
</tr>
<tr>
<td><code>earliest(x)</code></td>
<td>Returns the given variable's value with the earliest timestamp before the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function computes a data point for every time interval, if the input property has at least one data point in its history.</td>
</tr>
<tr>
<td><code>latest(x)</code></td>
<td>Returns the given variable's value with the latest timestamp before the end of the current time interval.</td>
</tr>
<tr>
<td></td>
<td>This function computes a data point for every time interval, if the input property has at least one data point in its history.</td>
</tr>
<tr>
<td><code>statetime(x)</code></td>
<td>Returns the amount of time in seconds that the given variables are positive over the current time interval. You can use the comparison functions (p. 169) to create a transform property for the <code>statetime</code> function to consume.</td>
</tr>
<tr>
<td></td>
<td>For example, if you have an Idle property that is 0 or 1, you can calculate idle time per time interval with this expression: <code>IdleTime = statetime(Idle)</code>. For more information, see the example statetime scenario (p. 176).</td>
</tr>
<tr>
<td></td>
<td>This function doesn't support metric properties as input variables.</td>
</tr>
<tr>
<td></td>
<td>This function computes a data point for every time interval, if the input property has at least one data point in its history.</td>
</tr>
</tbody>
</table>
The following diagram shows how AWS IoT SiteWise computes the temporal functions first, last, earliest, and latest, relative to the current time interval.

**Example Example statetime scenario**

Consider an example where you have an asset with the following properties:

- **Idle** – A measurement that is 0 or 1. When the value is 1, the machine is idle.
- **Idle Time** – A metric that uses the formula $\text{statetime}(\text{Idle})$ to calculate the amount of time in seconds where the machine is idle, per 1 minute interval.

The Idle property has the following data points.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>2:00:00 PM</th>
<th>2:00:30 PM</th>
<th>2:01:15 PM</th>
<th>2:02:45 PM</th>
<th>2:04:00 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AWS IoT SiteWise calculates the Idle Time property every minute from the values of Idle. After this calculation completes, the Idle Time property has the following data points.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>2:00:00 PM</th>
<th>2:01:00 PM</th>
<th>2:02:00 PM</th>
<th>2:03:00 PM</th>
<th>2:04:00 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle Time</td>
<td>N/A</td>
<td>30</td>
<td>60</td>
<td>45</td>
<td>0</td>
</tr>
</tbody>
</table>

AWS IoT SiteWise performs the following calculations for Idle Time at the end of each minute.

- At 2:00 PM (for 1:59 PM to 2:00 PM)
  - There is no data for Idle before 2:00 PM, so no data point is calculated.
- At 2:01 PM (for 2:00 PM to 2:01 PM)
  - At 2:00:00 PM, the machine is active (Idle is 0).
  - At 2:00:30 PM, the machine is idle (Idle is 1).
  - Idle doesn’t change again before the end of the interval at 2:01:00 PM, so Idle Time is 30 seconds.
- At 2:02 PM (for 2:01 PM to 2:02 PM)
  - At 2:01:00 PM, the machine is idle (per the last data point at 2:00:30 PM).
  - At 2:01:15 PM, the machine is still idle.
  - Idle doesn’t change again before the end of the interval at 2:02:00 PM, so Idle Time is 60 seconds.
- At 2:03 PM (for 2:02 PM to 2:03 PM)
  - At 2:02:00 PM, the machine is idle (per the last data point at 2:01:15 PM).
  - At 2:02:45 PM, the machine is active.
  - Idle doesn’t change again before the end of the interval at 2:03:00 PM, so Idle Time is 45 seconds.
- At 2:04 PM (for 2:03 PM to 2:04 PM)
  - At 2:03:00 PM, the machine is active (per the last data point at 2:02:45 PM).
  - Idle doesn’t change again before the end of the interval at 2:04:00 PM, so Idle Time is 0 seconds.
Using temporal functions in transforms

In transforms (p. 154) only, you can use the `pretrigger()` function to retrieve the GOOD quality value for a variable prior to the property update that triggered the current transform calculation.

Consider an example where a manufacturer uses AWS IoT SiteWise to monitor the status of a machine. The manufacturer uses the following measurements and transforms to represent the process:

- A measurement, `current_state`, that can be 0 or 1.
  - If the machine is in the cleaning state, `current_state` equals 1.
  - If the machine is in the manufacturing state, `current_state` equals 0.
- A transform, `cleaning_state_duration`, that equals \( \text{if}(\text{pretrigger}(\text{current_state}) == 1, \text{timestamp}(\text{current_state}) - \text{timestamp}(\text{pretrigger}(\text{current_state})), \text{none}) \). This transform returns how long the machine has been in the cleaning state in seconds, in the Unix epoch format. For more information, see Conditional functions (p. 170) and the `timestamp()` (p. 177) function.

If the machine stays in the cleaning state longer than expected, the manufacturer might investigate the machine.

You can also use the `pretrigger()` function in multivariate transforms. For example, you have two measurements named \( x \) and \( y \), and a transform, \( z \), that equals \( x + y + \text{pretrigger}(y) \). The following table shows the values for \( x \), \( y \), and \( z \) from 9:00 AM to 9:15 AM.

**Note**

- This example assumes that the values for the measurements arrive chronologically. For example, the value of \( x \) for 09:00 AM arrives before the value of \( x \) for 09:05 AM.
- If the data points for 09:05 AM arrive before the data points for 09:00 AM, \( z \) isn't calculated at 09:05 AM.
- If the value of \( x \) for 09:05 AM arrives before the value of \( x \) for 09:00 AM and the values of \( y \) arrive chronologically, \( z \) equals \( 22 = 20 + 1 + 1 \) at 09:05 AM.

<table>
<thead>
<tr>
<th></th>
<th>09:00 AM</th>
<th>09:05 AM</th>
<th>09:10 AM</th>
<th>09:15 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( z = x + y + \text{pretrigger}(y) )</td>
<td>( y ) doesn't receive any data point before 09:00 AM. Therefore, ( z ) isn't calculated at 09:00 AM.</td>
<td>( 23 = 20 + 2 + 1 ) ( \text{pretrigger}(y) ) equals 1.</td>
<td>( 25 = 20 + 3 + 2 ) ( \text{pretrigger}(y) ) equals 2.</td>
<td>( 36 = 30 + 3 + 3 ) ( y ) doesn't receive a new data point. Therefore, ( \text{pretrigger}(y) ) equals 3 at 09:15 AM.</td>
</tr>
</tbody>
</table>

**Date and time functions**

In transforms (p. 154) and metrics (p. 157), you can use the date and time functions in the following ways:

- Retrieve the current timestamp of a data point in UTC or in the local time zone.
• Construct timestamps with arguments, such as `year`, `month`, and `day_of_month`.
• Extract a time period such as a year or month with the `unix_time` argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>now()</code></td>
<td>Returns the current date and time, in seconds, in the Unix epoch format.</td>
</tr>
<tr>
<td><code>timestamp()</code></td>
<td>• In transforms, the function returns the timestamp, in seconds, of the input message in the Unix epoch format.</td>
</tr>
<tr>
<td></td>
<td>In transforms only, you can do one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Provide a variable as an argument to the function. The <code>timestamp(variable-name)</code> function returns the timestamp, in seconds, of the latest GOOD quality value for the specified variable in the Unix epoch format.</td>
</tr>
<tr>
<td></td>
<td>For example, if your asset has a transform property named <code>Temperature_F</code> that uses the <code>9/5 * Temperature_C</code> formula to convert each temperature data point from Celsius to Fahrenheit, you can use the <code>timestamp(Temperature_F)</code> function to get the timestamp of the latest GOOD quality value for the <code>Temperature_F</code> property.</td>
</tr>
<tr>
<td></td>
<td>• Use the <code>pretrigger()</code> function as an argument to the function. The <code>timestamp(pretrigger(variable-name))</code> function returns the timestamp, in seconds, of the GOOD quality value for the specified variable prior to the property update that triggered the current transform calculation in the Unix epoch format. For more information, see Using temporal functions in transforms (p. 177).</td>
</tr>
<tr>
<td><code>mktime(time_zone, year, month, day_of_month, hour, minute, second)</code></td>
<td>Returns the input time in seconds, in the Unix epoch format.</td>
</tr>
<tr>
<td></td>
<td>The following requirements apply for using this function:</td>
</tr>
<tr>
<td></td>
<td>• The time zone argument must be a quoted string (‘UTC’). If not specified, the default time zone is UTC.</td>
</tr>
<tr>
<td></td>
<td>The time zone argument can be the first or last argument.</td>
</tr>
<tr>
<td></td>
<td>• The year, month, day of month, hour, minute, and second arguments must be in order.</td>
</tr>
</tbody>
</table>
### Defining data properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The year, month, and date arguments are required.</td>
<td></td>
</tr>
</tbody>
</table>

The following limits apply for using this function:

- **year** - Valid values are between 1970 and 2250.
- **month** - Valid values are between 1 and 12.
- **day-of-month** - Valid values are between 1 - 31.
- **hour** - Valid values are between 0 and 23.
- **minute** - Valid values are between 0 and 59.
- **second** - Valid values are between 0 and 60. It can be a floating point number.

**Examples:**

- `mktime(2020, 2, 29)`
- `mktime('UTC+3', 2021, 12, 31, 22)`
- `mktime(2022, 10, 13, 2, 55, 13.68, 'PST')`
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localtime(unix_time, time_zone)</td>
<td>Returns the year, the day of the month, the day of the week, the day of the year, the hour, the minute, or the second in the specified time zone from the Unix time.</td>
</tr>
<tr>
<td></td>
<td>The following requirements apply for using this function:</td>
</tr>
<tr>
<td></td>
<td>• The time zone argument must be a quoted string (&quot;UTC&quot;). If not specified, the default time zone is UTC.</td>
</tr>
<tr>
<td></td>
<td>• The Unix time argument is the time in seconds, in the Unix epoch format. The valid range is between 1-31556889864403199. It can be a floating point number.</td>
</tr>
<tr>
<td></td>
<td>Example response:</td>
</tr>
<tr>
<td></td>
<td>2007-12-03T10:15:30+01:00[Europe/Paris]</td>
</tr>
<tr>
<td></td>
<td>localtime(unix_time, time_zone) isn't a standalone function. The year(), mon(), mday, wday(), yday(), hour(), minute(), and sec() functions take localtime(unix_time, time_zone) as an argument.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>• year(localtime('GMT', 1605898608.8113723))</td>
</tr>
<tr>
<td></td>
<td>• now().localtime().year()</td>
</tr>
<tr>
<td></td>
<td>• timestamp().localtime('PST').year()</td>
</tr>
<tr>
<td></td>
<td>• localtime(1605289736, 'Europe/London').year()</td>
</tr>
<tr>
<td>year(localtime(unix_time, time_zone))</td>
<td>Returns the year from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>mon(localtime(unix_time, time_zone))</td>
<td>Returns the month from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>mday(localtime(unix_time, time_zone))</td>
<td>Returns the day of the month from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>wday(localtime(unix_time, time_zone))</td>
<td>Returns the day of the week from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>yday(localtime(unix_time, time_zone))</td>
<td>Returns the day of the year from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>hour(localtime(unix_time, time_zone))</td>
<td>Returns the hour from localtime(unix_time, time_zone).</td>
</tr>
<tr>
<td>minute(localtime(unix_time, time_zone))</td>
<td>Returns the minute from localtime(unix_time, time_zone).</td>
</tr>
</tbody>
</table>
Defining data properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec(localtime(unix_time, time_zone))</td>
<td>Returns the second from localtime(unix_time, time_zone).</td>
</tr>
</tbody>
</table>

### Supported time zone formats

You can specify the time zone argument in the following ways:

- **Time zone offset** - Specify `'Z'` for UTC or an offset (`'+2'` or `'−5'`).
- **Offset IDs** - Combine a time zone abbreviation and an offset. For example, `'GMT+2'` and `'UTC−01:00'`. The time zone abbreviation must contain only three letters. For a list of supported time zone abbreviations, see Supported time zone abbreviations and Region-based IDs: (p. 181).
- **Region based IDs** - For example, `'Etc/GMT+12'` and `'Pacific/Pago_Pago'`. For the list of supported Region-based IDs, see Supported time zone abbreviations and Region-based IDs: (p. 181).

### Supported time zone abbreviations and Region-based IDs:

The date and time functions support the following three-letter time zone abbreviations:

- EST - -05:00
- HST - -10:00
- MST - -07:00
- ACT - Australia/Darwin
- AET - Australia/Sydney
- AGT - America/Argentina/Buenos_Aires
- ART - Africa/Cairo
- AST - America/Anchorage
- BET - America/Sao_Paulo
- BST - Asia/Dhaka
- CAT - Africa/Harare
- CNT - America/St_Johns
- CST - America/Chicago
- CTT - Asia/Shanghai
- EAT - Africa/Addis_Ababa
- CET - Europe/Paris
- IET - America/Indiana/Indianapolis
- IST - Asia/Kolkata
- JST - Asia/Tokyo
- MIT - Pacific/Apia
- NET - Asia/Yerevan
- NST - Pacific/Auckland
- PLT - Asia/Karachi
- PRT - America/Puerto_Rico
- PST - America/Los_Angeles
- SST - Pacific/Guadalcanal
- VST - Asia/Ho_Chi_Minh
The date and time functions support the following Region-based IDs:

- Etc/GMT+12 (UTC-12:00)
- Pacific/Pago_Pago (UTC-11:00)
- Pacific/Samoa (UTC-11:00)
- Pacific/Niue (UTC-11:00)
- US/Samoa (UTC-11:00)
- Etc/GMT+11 (UTC-11:00)
- Pacific/Midway (UTC-11:00)
- Pacific/Honolulu (UTC-10:00)
- Pacific/Rarotonga (UTC-10:00)
- Pacific/Tahiti (UTC-10:00)
- Pacific/Johnston (UTC-10:00)
- US/Hawaii (UTC-10:00)
- SystemV/HST10 (UTC-10:00)
- Etc/GMT+10 (UTC-10:00)
- Pacific/Marquesas (UTC-09:30)
- Etc/GMT+9 (UTC-09:00)
- Pacific/Gambier (UTC-09:00)
- America/Atka (UTC-09:00)
- SystemV/YST9 (UTC-09:00)
- America/Adak (UTC-09:00)
- US/Aleutian (UTC-09:00)
- Etc/GMT+8 (UTC-08:00)
- US/Alaska (UTC-08:00)
- America/Juneau (UTC-08:00)
- America/Metlakatla (UTC-08:00)
- America/Yakutat (UTC-08:00)
- Pacific/Pitcairn (UTC-08:00)
- America/Sitka (UTC-08:00)
- America/Anchorage (UTC-08:00)
- SystemV/PST8 (UTC-08:00)
- America/Nome (UTC-08:00)
- SystemV/YST9YDT (UTC-08:00)
- Canada/Yukon (UTC-07:00)
- US/Pacific-New (UTC-07:00)
- Etc/GMT+7 (UTC-07:00)
- US/Arizona (UTC-07:00)
- America/Dawson_Creek (UTC-07:00)
- Canada/Pacific (UTC-07:00)
- PST8PDT (UTC-07:00)
- SystemV/MST7 (UTC-07:00)
- America/Dawson (UTC-07:00)
- Mexico/BajaNorte (UTC-07:00)
- America/Tijuana (UTC-07:00)
- America/Creston (UTC-07:00)
- America/Hermosillo (UTC-07:00)
- America/Santa_Isabel (UTC-07:00)
- America/Vancouver (UTC-07:00)
- America/Ensenada (UTC-07:00)
- America/Phoenix (UTC-07:00)
- America/Whitehorse (UTC-07:00)
- America/Fort_Nelson (UTC-07:00)
- SystemV/PST8PDT (UTC-07:00)
- America/Los_Angeles (UTC-07:00)
- US/Pacific (UTC-07:00)
- America/EL_Salvador (UTC-06:00)
- America/Guatemala (UTC-06:00)
- America/Belize (UTC-06:00)
- America/Managua (UTC-06:00)
- America/Tegucigalpa (UTC-06:00)
- Etc/GMT+6 (UTC-06:00)
- Pacific/Easter (UTC-06:00)
- Mexico/BajaSur (UTC-06:00)
- America/Regina (UTC-06:00)
- America/Denver (UTC-06:00)
- Pacific/Galapagos (UTC-06:00)
- America/Yellowknife (UTC-06:00)
- America/Swift_Current (UTC-06:00)
- America/Inuvik (UTC-06:00)
- America/Mazatlan (UTC-06:00)
- America/Boise (UTC-06:00)
- America/Costa_Rica (UTC-06:00)
- MST7MDT (UTC-06:00)
- SystemV/CST6 (UTC-06:00)
- America/Chihuahua (UTC-06:00)
- America/Ojinaga (UTC-06:00)
- Chile/EasterIsland (UTC-06:00)
- US/Mountain (UTC-06:00)
- America/Edmonton (UTC-06:00)
- Canada/Mountain (UTC-06:00)
- America/Cambridge_Bay (UTC-06:00)
- Navajo (UTC-06:00)
- SystemV/MST7MDT (UTC-06:00)
- Canada/Saskatchewan (UTC-06:00)
- America/Shiprock (UTC-06:00)
- America/Panama (UTC-05:00)
• America/Chicago (UTC-05:00)
• America/Eirunepe (UTC-05:00)
• Etc/GMT+5 (UTC-05:00)
• Mexico/General (UTC-05:00)
• America/Porto_Acre (UTC-05:00)
• America/Guayaquil (UTC-05:00)
• America/Rankin_Inlet (UTC-05:00)
• US/Central (UTC-05:00)
• America/Rainy_River (UTC-05:00)
• America/Indiana/Knox (UTC-05:00)
• America/North_Dakota/Beulah (UTC-05:00)
• America/Monterrey (UTC-05:00)
• America/Jamaica (UTC-05:00)
• America/Atikokan (UTC-05:00)
• America/Coral_Harbour (UTC-05:00)
• America/North_Dakota/Center (UTC-05:00)
• America/Cayman (UTC-05:00)
• America/Indiana/Tell_City (UTC-05:00)
• America/Mexico_City (UTC-05:00)
• America/Matamoros (UTC-05:00)
• CST6CDT (UTC-05:00)
• America/Knox_IN (UTC-05:00)
• America/Bogota (UTC-05:00)
• America/Menominee (UTC-05:00)
• America/Resolute (UTC-05:00)
• SystemV/EST5 (UTC-05:00)
• Canada/Central (UTC-05:00)
• Brazil/Acre (UTC-05:00)
• America/Cancun (UTC-05:00)
• America/Lima (UTC-05:00)
• America/Bahia_Banderas (UTC-05:00)
• US/Indiana-Starke (UTC-05:00)
• America/Rio_Branco (UTC-05:00)
• SystemV/CST6CDT (UTC-05:00)
• Jamaica (UTC-05:00)
• America/Merida (UTC-05:00)
• America/North_Dakota/New_Salem (UTC-05:00)
• America/Winnipeg (UTC-05:00)
• America/Cuiaba (UTC-04:00)
• America/Marigot (UTC-04:00)
• America/Indiana/Petersburg (UTC-04:00)
• Chile/Continental (UTC-04:00)
• America/Grand_Turk (UTC-04:00)
• Cuba (UTC-04:00)
• Etc/GMT+4 (UTC-04:00)
• America/Manaus (UTC-04:00)
• America/Fort_Wayne (UTC-04:00)
• America/St_Thomas (UTC-04:00)
• America/Anguilla (UTC-04:00)
• America/Havana (UTC-04:00)
• US/Michigan (UTC-04:00)
• America/Barbados (UTC-04:00)
• America/Louisville (UTC-04:00)
• America/Curacao (UTC-04:00)
• America/Guyana (UTC-04:00)
• America/Martinique (UTC-04:00)
• America/Puerto_Rico (UTC-04:00)
• America/Port_of_Spain (UTC-04:00)
• SystemV/AST4 (UTC-04:00)
• America/Indiana/Vevay (UTC-04:00)
• America/Indiana/Vincennes (UTC-04:00)
• America/Kralendijk (UTC-04:00)
• America/Antigua (UTC-04:00)
• America/Indianapolis (UTC-04:00)
• America/Iqaluit (UTC-04:00)
• America/St_Vincent (UTC-04:00)
• America/Kentucky/Louisville (UTC-04:00)
• America/Dominica (UTC-04:00)
• America/Asuncion (UTC-04:00)
• EST5EDT (UTC-04:00)
• America/Nassau (UTC-04:00)
• America/Kentucky/Monticello (UTC-04:00)
• Brazil/West (UTC-04:00)
• America/Aruba (UTC-04:00)
• America/Indiana/Indianapolis (UTC-04:00)
• America/Santiago (UTC-04:00)
• America/La_Paz (UTC-04:00)
• America/Thunder_Bay (UTC-04:00)
• America/Indiana/Marengo (UTC-04:00)
• America/Blanc-Sablon (UTC-04:00)
• America/Santo_Domingo (UTC-04:00)
• US/Eastern (UTC-04:00)
• Canada/Eastern (UTC-04:00)
• America/Port-au-Prince (UTC-04:00)
• America/St_Barthelemy (UTC-04:00)
• America/Nipigon (UTC-04:00)
• US/East-Indiana (UTC-04:00)
• America/St_Lucia (UTC-04:00)
• America/Montserrat (UTC-04:00)
• America/Lower_Princes (UTC-04:00)
• America/Detroit (UTC-04:00)
• America/Tortola (UTC-04:00)
• America/Porto_Velho (UTC-04:00)
• America/Campo_Grande (UTC-04:00)
• America/Virgin (UTC-04:00)
• America/Pangnirtung (UTC-04:00)
• America/Montreal (UTC-04:00)
• America/Indiana/Winamac (UTC-04:00)
• America/Boa_Vista (UTC-04:00)
• America/Grenada (UTC-04:00)
• America/New_York (UTC-04:00)
• America/St_Kitts (UTC-04:00)
• America/Caracas (UTC-04:00)
• America/Guadeloupe (UTC-04:00)
• America/Toronto (UTC-04:00)
• SystemV/EST5EDT (UTC-04:00)
• America/Argentina/Catamarca (UTC-03:00)
• Canada/Atlantic (UTC-03:00)
• America/Argentina/Cordoba (UTC-03:00)
• America/Araguaina (UTC-03:00)
• America/Argentina/Salta (UTC-03:00)
• Etc/GMT+3 (UTC-03:00)
• America/Montevideo (UTC-03:00)
• Brazil/East (UTC-03:00)
• America/Argentina/Mendoza (UTC-03:00)
• America/Argentina/Rio_Gallegos (UTC-03:00)
• America/Catamarca (UTC-03:00)
• America/Cordoba (UTC-03:00)
• America/Sao_Paulo (UTC-03:00)
• America/Argentina/Jujuy (UTC-03:00)
• America/Cayenne (UTC-03:00)
• America/Recife (UTC-03:00)
• America/Buenos_Aires (UTC-03:00)
• America/Paramaribo (UTC-03:00)
• America/Moncton (UTC-03:00)
• America/Mendoza (UTC-03:00)
• America/Santarem (UTC-03:00)
• Atlantic/Bermuda (UTC-03:00)
• America/Maceio (UTC-03:00)
- Atlantic/Stanley (UTC-03:00)
- America/Halifax (UTC-03:00)
- Antarctica/Rothera (UTC-03:00)
- America/Argentina/San_Luis (UTC-03:00)
- America/Argentina/Ushuaia (UTC-03:00)
- Antarctica/Palmer (UTC-03:00)
- America/Punta_Arenas (UTC-03:00)
- America/Glace_Bay (UTC-03:00)
- America/Fortaleza (UTC-03:00)
- America/Thule (UTC-03:00)
- America/Argentina/La_Rioja (UTC-03:00)
- America/Belem (UTC-03:00)
- America/Jujuy (UTC-03:00)
- America/Bahia (UTC-03:00)
- America/Goose_Bay (UTC-03:00)
- America/Argentina/San_Juan (UTC-03:00)
- America/Argentina/ComodRivadavia (UTC-03:00)
- America/Argentina/Tucuman (UTC-03:00)
- America/Rosario (UTC-03:00)
- SystemV/AST4ADT (UTC-03:00)
- America/Argentina/Buenos_Aires (UTC-03:00)
- America/St_Johns (UTC-02:30)
- Canada/Newfoundland (UTC-02:30)
- America/Miquelon (UTC-02:00)
- Etc/GMT+2 (UTC-02:00)
- America/Godthab (UTC-02:00)
- America/Noronha (UTC-02:00)
- Brazil/DeNoronha (UTC-02:00)
- Atlantic/South_Georgia (UTC-02:00)
- Etc/GMT+1 (UTC-01:00)
- Atlantic/Cape_Verde (UTC-01:00)
- Pacific/Kiritimati (UTC+14:00)
- Etc/GMT-14 (UTC+14:00)
- Pacific/Fakaofo (UTC+13:00)
- Pacific/Enderbury (UTC+13:00)
- Pacific/Apia (UTC+13:00)
- Pacific/Tongatapu (UTC+13:00)
- Etc/GMT-13 (UTC+13:00)
- NZ-CHAT (UTC+12:45)
- Pacific/Chatham (UTC+12:45)
- Pacific/Kwajalein (UTC+12:00)
- Antarctica/McMurdo (UTC+12:00)
- Pacific/Wallis (UTC+12:00)
• Pacific/Fiji (UTC+12:00)
• Pacific/Funafuti (UTC+12:00)
• Pacific/Nauru (UTC+12:00)
• Kwajalein (UTC+12:00)
• NZ (UTC+12:00)
• Pacific/Wake (UTC+12:00)
• Antarctica/South_Pole (UTC+12:00)
• Pacific/Tarawa (UTC+12:00)
• Pacific/Auckland (UTC+12:00)
• Asia/Kamchatka (UTC+12:00)
• Etc/GMT-12 (UTC+12:00)
• Asia/Anadyr (UTC+12:00)
• Pacific/Majuro (UTC+12:00)
• Pacific/Ponape (UTC+11:00)
• Pacific/Bougainville (UTC+11:00)
• Antarctica/Macquarie (UTC+11:00)
• Pacific/Pohnpei (UTC+11:00)
• Pacific/Efate (UTC+11:00)
• Pacific/Norfolk (UTC+11:00)
• Asia/Magadan (UTC+11:00)
• Pacific/Kosrae (UTC+11:00)
• Asia/Sakhalin (UTC+11:00)
• Pacific/Noumea (UTC+11:00)
• Etc/GMT-11 (UTC+11:00)
• Asia/Srednekolymsk (UTC+11:00)
• Pacific/Guadacanal (UTC+11:00)
• Australia/Lord_Howe (UTC+10:30)
• Australia/LHI (UTC+10:30)
• Australia/Hobart (UTC+10:00)
• Pacific/Yap (UTC+10:00)
• Australia/Tasmania (UTC+10:00)
• Pacific/Port_Moresby (UTC+10:00)
• Australia/ACT (UTC+10:00)
• Australia/Victoria (UTC+10:00)
• Pacific/Chuuk (UTC+10:00)
• Australia/Queensland (UTC+10:00)
• Australia/Canberra (UTC+10:00)
• Australia/Currie (UTC+10:00)
• Pacific/Guam (UTC+10:00)
• Pacific/Truk (UTC+10:00)
• Australia/NSW (UTC+10:00)
• Asia/Vladivostok (UTC+10:00)
• Pacific/Saipan (UTC+10:00)
• Antarctica/DumontD'Urville (UTC+10:00)
• Australia/Sydney (UTC+10:00)
• Australia/Brisbane (UTC+10:00)
• Etc/GMT-10 (UTC+10:00)
• Asia/Ust-Nera (UTC+10:00)
• Australia/Melbourne (UTC+10:00)
• Australia/Lindeman (UTC+10:00)
• Australia/North (UTC+09:30)
• Australia/Yancowinna (UTC+09:30)
• Australia/Adelaide (UTC+09:30)
• Australia/Broken_Hill (UTC+09:30)
• Australia/South (UTC+09:30)
• Australia/Darwin (UTC+09:30)
• Etc/GMT-9 (UTC+09:00)
• Pacific/Palau (UTC+09:00)
• Asia/Chita (UTC+09:00)
• Asia/Dili (UTC+09:00)
• Asia/Jayapura (UTC+09:00)
• Asia/Yakutsk (UTC+09:00)
• Asia/Pyongyang (UTC+09:00)
• ROK (UTC+09:00)
• Asia/Seoul (UTC+09:00)
• Asia/Khandyga (UTC+09:00)
• Japan (UTC+09:00)
• Asia/Tokyo (UTC+09:00)
• Australia/Eucla (UTC+08:45)
• Asia/Kuching (UTC+08:00)
• Asia/Chungking (UTC+08:00)
• Etc/GMT-8 (UTC+08:00)
• Australia/Perth (UTC+08:00)
• Asia/Macao (UTC+08:00)
• Asia/Macau (UTC+08:00)
• Asia/Choibalsan (UTC+08:00)
• Asia/Shanghai (UTC+08:00)
• Antarctica/Casey (UTC+08:00)
• Asia/Ulan_Bator (UTC+08:00)
• Asia/Chongqing (UTC+08:00)
• Asia/Ulaanbaatar (UTC+08:00)
• Asia/Taipei (UTC+08:00)
• Asia/Manila (UTC+08:00)
• PRC (UTC+08:00)
• Asia/Ujung_Pandang (UTC+08:00)
• Asia/Harbin (UTC+08:00)
• Singapore (UTC+08:00)
• Asia/Brunei (UTC+08:00)
• Australia/West (UTC+08:00)
• Asia/Hong_Kong (UTC+08:00)
• Asia/Makassar (UTC+08:00)
• Hongkong (UTC+08:00)
• Asia/Kuala_Lumpur (UTC+08:00)
• Asia/Irkutsk (UTC+08:00)
• Asia/Singapore (UTC+08:00)
• Asia/Pontianak (UTC+07:00)
• Etc/GMT-7 (UTC+07:00)
• Asia/Phnom_Penh (UTC+07:00)
• Asia/Novosibirsk (UTC+07:00)
• Antarctica/Davis (UTC+07:00)
• Asia/Tomsk (UTC+07:00)
• Asia/Jakarta (UTC+07:00)
• Asia/Barnaul (UTC+07:00)
• Indian/Christmas (UTC+07:00)
• Asia/Ho_Chi_Minh (UTC+07:00)
• Asia/Hovd (UTC+07:00)
• Asia/Bangkok (UTC+07:00)
• Asia/Vientiane (UTC+07:00)
• Asia/Novokuznetsk (UTC+07:00)
• Asia/Krasnoyarsk (UTC+07:00)
• Asia/Saigon (UTC+07:00)
• Asia/Yangon (UTC+06:30)
• Asia/Rangoon (UTC+06:30)
• Indian/Cocos (UTC+06:30)
• Asia/Kashgar (UTC+06:00)
• Etc/GMT-6 (UTC+06:00)
• Asia/Almaty (UTC+06:00)
• Asia/Dacca (UTC+06:00)
• Asia/Omsk (UTC+06:00)
• Asia/Dhaka (UTC+06:00)
• Indian/Chagos (UTC+06:00)
• Asia/Qyzylorda (UTC+06:00)
• Asia/Bishkek (UTC+06:00)
• Antarctica/Vostok (UTC+06:00)
• Asia/Urumqi (UTC+06:00)
• Asia/Thimbu (UTC+06:00)
• Asia/Thimphu (UTC+06:00)
• Asia/Kathmandu (UTC+05:45)
• Asia/Katmandu (UTC+05:45)
• Asia/Kolkata (UTC+05:30)
• Asia/Colombo (UTC+05:30)
• Asia/Calcutta (UTC+05:30)
• Asia/Aqtau (UTC+05:00)
• Etc/GMT-5 (UTC+05:00)
• Asia/Samarkand (UTC+05:00)
• Asia/Karachi (UTC+05:00)
• Asia/Yekaterinburg (UTC+05:00)
• Asia/Dushanbe (UTC+05:00)
• Indian/Maldives (UTC+05:00)
• Asia/Oral (UTC+05:00)
• Asia/Tashkent (UTC+05:00)
• Antarctica/Mawson (UTC+05:00)
• Asia/Aqtobe (UTC+05:00)
• Asia/Ashkhabad (UTC+05:00)
• Asia/Ashgabat (UTC+05:00)
• Asia/Atyrau (UTC+05:00)
• Indian/Kerguelen (UTC+05:00)
• Iran (UTC+04:30)
• Asia/Tehran (UTC+04:30)
• Asia/Kabul (UTC+04:30)
• Asia/Yerevan (UTC+04:00)
• Etc/GMT-4 (UTC+04:00)
• Etc/GMT-4 (UTC+04:00)
• Asia/Dubai (UTC+04:00)
• Indian/Reunion (UTC+04:00)
• Europe/Saratov (UTC+04:00)
• Europe/Samara (UTC+04:00)
• Indian/Mahe (UTC+04:00)
• Asia/Baku (UTC+04:00)
• Asia/Muscat (UTC+04:00)
• Europe/Volgograd (UTC+04:00)
• Europe/Astrakhan (UTC+04:00)
• Asia/Tbilisi (UTC+04:00)
• Europe/Ulyanovsk (UTC+04:00)
• Asia/Aden (UTC+03:00)
• Africa/Nairobi (UTC+03:00)
• Europe/Istanbul (UTC+03:00)
• Etc/GMT-3 (UTC+03:00)
• Europe/Zaporozhye (UTC+03:00)
• Israel (UTC+03:00)
• Indian/Comoro (UTC+03:00)
• Antarctica/Syowa (UTC+03:00)
• Africa/Mogadishu (UTC+03:00)
• Europe/Bucharest (UTC+03:00)
• Africa/Asmera (UTC+03:00)
• Europe/Mariehamn (UTC+03:00)
• Asia/Istanbul (UTC+03:00)
• Europe/Tiraspol (UTC+03:00)
• Europe/Moscow (UTC+03:00)
• Europe/Chisinau (UTC+03:00)
• Europe/Helsinki (UTC+03:00)
• Asia/Beirut (UTC+03:00)
• Asia/Tel_Aviv (UTC+03:00)
• Africa/Djibouti (UTC+03:00)
• Europe/Simferopol (UTC+03:00)
• Europe/Sofia (UTC+03:00)
• Asia/Gaza (UTC+03:00)
• Africa/Asmara (UTC+03:00)
• Europe/Riga (UTC+03:00)
• Asia/Baghdad (UTC+03:00)
• Asia/Damascus (UTC+03:00)
• Africa/Dar_es_Salaam (UTC+03:00)
• Africa/Addis_Ababa (UTC+03:00)
• Europe/Uzhgorod (UTC+03:00)
• Asia/Jerusalem (UTC+03:00)
• Asia/Riyadh (UTC+03:00)
• Asia/Kuwait (UTC+03:00)
• Europe/Kirov (UTC+03:00)
• Africa/Kampala (UTC+03:00)
• Europe/Minsk (UTC+03:00)
• Asia/Qatar (UTC+03:00)
• Europe/Kiev (UTC+03:00)
• Asia/Bahrain (UTC+03:00)
• Europe/Vilnius (UTC+03:00)
• Indian/Antananarivo (UTC+03:00)
• Indian/Mayotte (UTC+03:00)
• Europe/Tallinn (UTC+03:00)
• Turkey (UTC+03:00)
• Africa/Juba (UTC+03:00)
• Asia/Nicosia (UTC+03:00)
• Asia/Famagusta (UTC+03:00)
• W-SU (UTC+03:00)
• EET (UTC+03:00)
• Asia/Hebron (UTC+03:00)
• Asia/Amman (UTC+03:00)
• Europe/Nicosia (UTC+03:00)
• Europe/Athens (UTC+03:00)
• Africa/Cairo (UTC+02:00)
• Africa/Mbabane (UTC+02:00)
• Europe/Brussels (UTC+02:00)
Defining data properties

- Europe/Warsaw (UTC+02:00)
- CET (UTC+02:00)
- Europe/Luxembourg (UTC+02:00)
- Etc/GMT-2 (UTC+02:00)
- Libya (UTC+02:00)
- Africa/Kigali (UTC+02:00)
- Africa/Tripoli (UTC+02:00)
- Europe/Kaliningrad (UTC+02:00)
- Africa/Windhoek (UTC+02:00)
- Europe/Malta (UTC+02:00)
- Europe/Busingen (UTC+02:00)
- Europe/Skopje (UTC+02:00)
- Europe/Sarajevo (UTC+02:00)
- Europe/Rome (UTC+02:00)
- Europe/Zurich (UTC+02:00)
- Europe/Gibraltar (UTC+02:00)
- Africa/Lubumbashi (UTC+02:00)
- Europe/Vaduz (UTC+02:00)
- Europe/Ljubljana (UTC+02:00)
- Europe/Berlin (UTC+02:00)
- Europe/Stockholm (UTC+02:00)
- Europe/Budapest (UTC+02:00)
- Europe/Zagreb (UTC+02:00)
- Europe/Paris (UTC+02:00)
- Africa/Ceuta (UTC+02:00)
- Europe/Prague (UTC+02:00)
- Antarctica/Troll (UTC+02:00)
- Africa/Gaborone (UTC+02:00)
- Europe/Copenhagen (UTC+02:00)
- Europe/Vienna (UTC+02:00)
- Europe/Tirane (UTC+02:00)
- MET (UTC+02:00)
- Europe/Amsterdam (UTC+02:00)
- Africa/Maputo (UTC+02:00)
- Europe/San_Marino (UTC+02:00)
- Poland (UTC+02:00)
- Europe/Andorra (UTC+02:00)
- Europe/Oslo (UTC+02:00)
- Europe/Podgorica (UTC+02:00)
- Africa/Bujumbura (UTC+02:00)
- Atlantic/Jan_Mayen (UTC+02:00)
- Africa/Maseru (UTC+02:00)
- Europe/Madrid (UTC+02:00)
• Africa/Blantyre (UTC+02:00)
• Africa/Lusaka (UTC+02:00)
• Africa/Harare (UTC+02:00)
• Africa/Khartoum (UTC+02:00)
• Africa/Johannesburg (UTC+02:00)
• Europe/Belgrade (UTC+02:00)
• Europe/Bratislava (UTC+02:00)
• Arctic/Longyearbyen (UTC+02:00)
• Egypt (UTC+02:00)
• Europe/Vatican (UTC+02:00)
• Europe/Monaco (UTC+02:00)
• Europe/London (UTC+01:00)
• Etc/GMT-1 (UTC+01:00)
• Europe/Jersey (UTC+01:00)
• Europe/Guernsey (UTC+01:00)
• Europe/Isle_of_Man (UTC+01:00)
• Africa/Tunis (UTC+01:00)
• Africa/Malabo (UTC+01:00)
• GB-Eire (UTC+01:00)
• Africa/Lagos (UTC+01:00)
• Africa/Algiers (UTC+01:00)
• GB (UTC+01:00)
• Portugal (UTC+01:00)
• Africa/Sao_Tome (UTC+01:00)
• Africa/Ndjamena (UTC+01:00)
• Atlantic/Faeroe (UTC+01:00)
• Eire (UTC+01:00)
• Atlantic/Faroe (UTC+01:00)
• Europe/Dublin (UTC+01:00)
• Africa/Libreville (UTC+01:00)
• Africa/EL_Aaiun (UTC+01:00)
• Africa/EL_Aaiun (UTC+01:00)
• Africa/Douala (UTC+01:00)
• Africa/Brazzaville (UTC+01:00)
• Africa/Porto-Novo (UTC+01:00)
• Atlantic/Madeira (UTC+01:00)
• Europe/Lisbon (UTC+01:00)
• Atlantic/Canary (UTC+01:00)
• Africa/Casablanca (UTC+01:00)
• Europe/Belfast (UTC+01:00)
• Africa/Luanda (UTC+01:00)
• Africa/Kinshasa (UTC+01:00)
• Africa/Bangui (UTC+01:00)
• WET (UTC+01:00)
• Africa/Niamey (UTC+01:00)
• GMT (UTC+00:00)
• Etc/GMT-0 (UTC+00:00)
• Atlantic/St_Helena (UTC+00:00)
• Etc/GMT+0 (UTC+00:00)
• Africa/Banjul (UTC+00:00)
• Etc/GMT (UTC+00:00)
• Africa/Freetown (UTC+00:00)
• Africa/Bamako (UTC+00:00)
• Africa/Conakry (UTC+00:00)
• Universal (UTC+00:00)
• Africa/Nouakchott (UTC+00:00)
• UTC (UTC+00:00)
• Etc/Universal (UTC+00:00)
• Atlantic/Azores (UTC+00:00)
• Africa/Abidjan (UTC+00:00)
• Africa/Accra (UTC+00:00)
• Etc/UCT (UTC+00:00)
• GMT0 (UTC+00:00)
• Zulu (UTC+00:00)
• Africa/Ouagadougou (UTC+00:00)
• Atlantic/Reykjavik (UTC+00:00)
• Etc/Zulu (UTC+00:00)
• Iceland (UTC+00:00)
• Africa/Lome (UTC+00:00)
• Greenwich (UTC+00:00)
• Etc/GMT0 (UTC+00:00)
• America/Danmarkshavn (UTC+00:00)
• Africa/Dakar (UTC+00:00)
• Africa/Bissau (UTC+00:00)
• Etc/Greenwich (UTC+00:00)
• Africa/Timbuktu (UTC+00:00)
• UCT (UTC+00:00)
• Africa/Monrovia (UTC+00:00)
• Etc/UTC (UTC+00:00)

**Formula expression tutorials**

You can follow these tutorials to use formula expressions in AWS IoT SiteWise.

**Topics**

- Using strings in formulas (p. 196)
- Filtering data points (p. 196)
- Counting data points that match a condition (p. 196)
- Late data in formulas (p. 197)
• Data quality in formulas (p. 197)
• Undefined, infinite, and overflow values (p. 197)

Using strings in formulas

You can operate on strings in your formula expressions. You also can input strings from variables that reference attribute and measurement properties.

Important

Formula expressions can only output double or string values. Nested expressions can output other data types, such as strings, but the formula as a whole must evaluate to a number or string. You can use the `jp` function (p. 171) to convert a string to a number. The Boolean value must be 1 (true) or 0 (false). For more information, see Undefined, infinite, and overflow values (p. 197).

AWS IoT SiteWise provides the following formula expression features that you can use to operate on strings:

• String literals (p. 163)
• The index operator (p. 166) (`s[index]`)
• The slice operator (p. 166) (`s[start:end:step]`)
• Comparison functions (p. 169), which you can use compare strings by lexicographic order
• String functions (p. 171), which includes the `jp` function that can parse serialized JSON objects and convert strings to numbers

Filtering data points

You can use the `if` function (p. 170) to filter out data points that don't meet a condition. The `if` function evaluates a condition and returns different values for `true` and `false` results. You can use the `none` constant (p. 167) as an output for one case of an `if` function to discard the data point for that case.

To filter out data points that match a condition

• Create a transform that uses the `if` function to define a condition that checks if a condition is met, and returns `none` as either the `result_if_true` or `result_if_false` value.

Example Example: Filter out data points where water isn't boiling

Consider a scenario where you have a measurement, `temp_c`, that provides the temperature (in Celsius) of water in a machine. You can define the following transform to filter out data points where the water isn't boiling:

• Transform: `boiling_temps = if(gte(temp_c, 100), temp_c, none)` – Returns the temperature if it's greater than or equal to 100 degrees Celsius, otherwise returns no data point.

Counting data points that match a condition

You can use comparison functions (p. 169) and `sum()` (p. 174) to count the number of data points for which a condition is true.

To count data points that match a condition

1. Create a transform that uses a comparison function to define a filter condition on another property.
2. Create a metric that sums the data points where that condition is met.

Example Example: Count the number of data points where water is boiling

Consider a scenario where you have a measurement, \( temp_c \), that provides the temperature (in Celsius) of water in a machine. You can define the following transform and metric properties to count the number of data points where the water is boiling:

- **Transform**: \( is\_boiling = gte(temp_c, 100) \) – Returns 1 if the temperature is greater than or equal to 100 degrees Celsius, otherwise returns 0.
- **Metric**: \( boiling\_count = sum(is\_boiling) \) – Returns the number of data points where water is boiling.

Late data in formulas

AWS IoT SiteWise supports late data ingestion of data that is up to 7 days old. When AWS IoT SiteWise receives late data, it recalculates existing values for any metric that inputs the late data in a past window. These recalculations result in data processing charges.

**Note**
When AWS IoT SiteWise computes properties that input late data, it uses each property's current formula expression.

After AWS IoT SiteWise recalculates a past window for a metric, it replaces the previous value for that window. If you enabled notifications for that metric, AWS IoT SiteWise also emits a property value notification. This means that you can receive a new property value update notification for the same property and timestamp for which you previously received a notification. If your applications or data lakes consume property value notifications, you must update the previous value with the new value so that their data is accurate.

Data quality in formulas

In AWS IoT SiteWise, each data point has a quality code, which can be one of the following:

- **GOOD** – The data isn't affected by any issues.
- **BAD** – The data is affected by an issue such as sensor failure.
- **UNCERTAIN** – The data is affected by an issue such as sensor inaccuracy.

AWS IoT SiteWise consumes only **GOOD** quality data when it computes transforms and metrics. AWS IoT SiteWise outputs only **GOOD** quality data for successful computations. If a computation is unsuccessful, then AWS IoT SiteWise doesn't output a data point for that computation. This can occur if a computation results in an undefined, infinite, or overflow value.

For more information about how to query data and filter by data quality, see [Querying asset property values and aggregates](#).

Undefined, infinite, and overflow values

Some formula expressions (such as \( x / 0 \), \( \sqrt{-1} \), or \( \log(0) \)) calculate values that are undefined in a real number system, infinite, or outside the range supported by AWS IoT SiteWise. When an asset property's expression computes an undefined, infinite, or overflow value, AWS IoT SiteWise doesn't output a data point for that computation.

AWS IoT SiteWise also doesn't output a data point if it computes a non-numeric value as the result of a formula expression. This means that if you define a formula that computes a string, array, or the none constant, then AWS IoT SiteWise doesn't output a data point for that computation.
Defining relationships between assets (hierarchies)

You can define asset hierarchies to create logical associations between the assets in your industrial operation. For example, you can define a wind farm composed of several wind turbine assets or an assembly line comprising multiple machine assets.

When you associate a child asset to a parent asset through a hierarchy, the parent asset's metrics can input data from the child asset's metrics. You can use asset hierarchies and metrics to calculate statistics that provide insight to your operation or a subset of your operation. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).

Each hierarchy defines a relationship between a parent asset model and a child asset model. In a parent asset model, you can define multiple hierarchies to the same child asset model. For example, if you have two different types of wind turbine in your wind farms, where all wind turbines use the same asset model, you can define a hierarchy for each type. Then, you can define metrics in the wind farm model to calculate independent and combined statistics for each type of wind turbine.

To define an asset hierarchy, you must have an asset model for a child asset. For more information, see Creating asset models (p. 143).

**Note**

When you define an asset hierarchy, the child asset model must be `ACTIVE` or have a previous `ACTIVE` version. For more information, see Asset and model states (p. 140).

Each asset model can have only one parent asset model. For example, it's not possible to define three asset models (A, B, and C) where A and B both define a hierarchy to C.

After you define hierarchical asset models and create assets, you can associate the assets to complete the parent-child relationship. For more information, see Creating assets (p. 199) and Associating and disassociating assets (p. 207).

**Topics**

- Defining asset hierarchies (console) (p. 198)
- Defining asset hierarchies (CLI) (p. 199)

**Defining asset hierarchies (console)**

When you define a hierarchy for an asset model in the AWS IoT SiteWise console, you specify the following parameters:

- **Hierarchy name** – The hierarchy's name, such as *Wind Turbines*.
- **Hierarchy model** – The child asset model.
For more information, see Creating an asset model (console) (p. 143).

**Example Example hierarchy definition**

The following example demonstrates an asset hierarchy that represents a wind farm’s relationship to wind turbines.

```
{
  ...
  "assetModelHierarchies": [
    {
      "name": "Wind Turbines",
      "childAssetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"
    },
  ]
}
```

**Defining asset hierarchies (CLI)**

When you define a hierarchy for an asset model with the AWS IoT SiteWise API, you specify the following parameters:

- **name** – The hierarchy’s name, such as **Wind Turbines**.
- **childAssetModelId** – The ID of the child asset model for the hierarchy. You can use the ListAssetModels operation to find the ID of an existing asset model.

**Example Example hierarchy definition**

The following example demonstrates an asset hierarchy that represents a wind farm’s relationship to wind turbines. This object is an example of an AssetModelHierarchy. For more information, see Creating an asset model (CLI) (p. 144).

```
{
  ...
  "assetModelHierarchies": [
    {
      "name": "Wind Turbines",
      "childAssetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"
    },
  ]
}
```

**Creating assets**

You can create an asset from an asset model. You must have an asset model before you can create an asset. If you haven’t created an asset model, see Creating asset models (p. 143).

**Note**

You can only create assets from **ACTIVE** models. If your model’s state isn’t **ACTIVE**, you may need to wait for up to a few minutes before you can create assets from that model. For more information, see Asset and model states (p. 140).

**Topics**

- Creating an asset (console) (p. 200)
- Creating an asset (CLI) (p. 201)
- Configuring a new asset (p. 202)
Creating an asset (console)

You can use the AWS IoT SiteWise console to create an asset.

To create an asset (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose Create asset.
4. On the Create asset page, do the following:
   a. For Model, choose the asset model from which to create an asset.
      
      **Note**
      
      If your model isn't **ACTIVE**, you must wait until it's active, or resolve issues if it's **FAILED**.
   b. Enter a Name for your asset.
   c. (Optional) Add tags for your asset. For more information, see Tagging your AWS IoT SiteWise resources (p. 369).
   d. Choose Create asset.
When you create an asset, the AWS IoT SiteWise console navigates to the new asset's page. On this page, you can see the asset's **Status**, which is initially **CREATING**. This page automatically updates, so you can wait for the asset's status to update.

**Note**
The asset creation process can take up to a minute. After the **Status** is **ACTIVE**, you can perform update operations on your asset. For more information, see Asset and model states (p. 140).

After you create an asset, see Configuring a new asset (p. 202).

## Creating an asset (CLI)

You can use the AWS Command Line Interface (AWS CLI) to create an asset from an asset model.

You must have an **assetModelId** to create an asset. If you created an asset model but don't know its **assetModelId**, you can use the **ListAssetModels** operation to view all of your asset models.

To create an asset from an asset model, use the **CreateAsset** operation. Specify the following parameters:

- **assetName** – The new asset's name. Give your asset a unique, friendly name that you can easily identify.
- **assetModelId** – The ID of the model to create the asset from.

### To create an asset (CLI)

- Run the following command to create an asset. Replace **asset-name** with a name for the asset and **asset-model-id** with the ID of the asset model.

  ```bash
  aws iotsitewise create-asset \
  --asset-name asset-name \
  --asset-model-id asset-model-id
  ```

  The operation returns a response that contains your new asset's details and status in the following format.

  ```json
  {
    "assetId": "String",
    "assetArn": "String",
    "assetStatus": {
      "state": "String",
      "error": {
        "code": "String",
        "message": "String"
      }
    }
  }
  ```

  The asset's **state** is **CREATING** until the asset creates.

  **Note**
The asset creation process can take up to a minute. To check your asset's status, use the **DescribeAsset** operation with your asset's ID as the **assetId** parameter. After the asset's **state** is **ACTIVE**, you can perform update operations on your asset. For more information, see Asset and model states (p. 140).
After you create an asset, see Configuring a new asset (p. 202).

Configuring a new asset

Finish configuring your asset with any of the following optional actions:

- Mapping industrial data streams to asset properties (p. 202) if your asset has measurement properties.
- Updating attribute values (p. 204) if your asset has unique attribute values.
- Associating and disassociating assets (p. 207) if your asset is a parent asset.

Mapping industrial data streams to asset properties

If your asset has measurement properties, you can define the property aliases to map your data streams to those properties. You can also set property aliases so that you can easily identify an asset property when you ingest or retrieve asset data.

This process requires that you know your property alias.

- If you ingest data from OPC-UA servers using a gateway (p. 87), your property alias is the path to a variable under the Objects node, starting with /. For example, if the path to your variable is company/windfarm/3/turbine/7/temperature, then your property alias is /company/windfarm/3/turbine/7/temperature. For more information about OPC-UA information architecture, see Information Model and Address Spacing mapping in the OPC UA Online Reference.

  Notes

  - If you configure a data stream prefix for your OPC-UA source, you must include that prefix in the property alias for all data streams from that source. For example, if you use /RentonWA as a prefix, then the previous alias is /RentonWA/company/windfarm/3/turbine/7/temperature.
  - Property aliases can contain up to 1,000 bytes. OPC-UA variables paths can contain up to 4,096 bytes. Currently, AWS IoT SiteWise doesn't support ingesting data from OPC-UA variables with long paths.
  - If you ingest data from other sources, such as using AWS IoT rules (p. 69) or the API (p. 77), you define your property aliases. You can define a property alias naming system that is applicable to your device configuration. For example, if you ingest data from AWS IoT things, you can include the thing name in property aliases to uniquely identify data streams. For more information about this example, see the Ingesting data from AWS IoT things (p. 15) tutorial.

Property aliases must be unique within a Region and AWS account. AWS IoT SiteWise returns an error if you set a property alias to one that already exists on another asset property.

If you have multiple OPC-UA sources with identical data stream paths, you can add a prefix to each source's paths to form unique aliases. For more information, see Configuring data sources (p. 102).

  Note

  This section describes how to set property aliases for measurement properties. For more information about how to set property aliases for external alarm state properties, see Mapping external alarm state streams (p. 254).

Topics

- Setting a property alias (console) (p. 203)
- Setting a property alias (CLI) (p. 203)
Setting a property alias (console)

You can use the AWS IoT SiteWise console to set an alias for an asset property.

To set a property alias (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset for which you want to set a property alias.
   
   Tip
   You can choose the arrow icon to expand an asset hierarchy to find your asset.

   ![Asset Tree](image)

4. Choose Edit.
5. Find the property for which you want to set an alias, and then enter the property alias.

6. Choose Save.

Setting a property alias (CLI)

You can use the AWS Command Line Interface (AWS CLI) to set an alias for an asset property.

You must know your asset's assetId and property's propertyId to complete this procedure. If you created an asset but don't know its assetId, use the ListAssets operation to view all of your assets for a specific model. Then, use the DescribeAsset operation to view your asset's properties including property IDs.

To map a data stream to your asset's property, use the UpdateAssetProperty operation. Specify the following parameters:

- `assetId` – The asset's ID.
- `propertyId` – The asset property's ID.
• **propertyAlias** – The data stream's path to alias to the property.

• **propertyNotificationState** – The property value notification state: ENABLED or DISABLED.

Specify the property's existing notification state when you update the property alias. You can retrieve the existing notification state with the DescribeAssetProperty operation.

If you omit this parameter, the new notification state is DISABLED. For more information about property notifications, see Interacting with other AWS services (p. 291).

**To set a property alias (CLI)**

1. Run the following command to retrieve the property's current notification state. Replace `asset-id` and `property-id` with the asset property's IDs.

```bash
aws iotsitewise describe-asset-property
  --asset-id asset-id
  --property-id property-id
```

The operation returns a response that contains the asset property's details in the following format. The property notification state is in `assetProperty.notification.state` in the JSON object.

```json
{
  "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
  "assetName": "Wind Turbine 7",
  "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
  "assetProperty": {
    "id": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
    "name": "Wind Speed",
    "notification": {
      "topic": "$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
      "state": "DISABLED | ENABLED"
    },
    "dataType": "DOUBLE",
    "unit": "m/s",
    "type": {
      "measurement": {}
    }
  }
}
```

2. Run the following command to set the asset property's alias. Replace `property-alias` with the property alias and `notification-state` with the notification state, or omit `--property-notification-state` to disable notifications.

```bash
aws iotsitewise update-asset-property
  --asset-id asset-id
  --property-id property-id
  --property-alias property-alias
  --property-notification-state notification-state
```

---

**Updating attribute values**

Assets inherit the attributes of their asset model, including the default value of the attribute. In some cases, you will want to keep the asset model's default attribute, such as for an asset manufacturer property. In other cases, you will want to update the inherited attribute, such as for an asset's latitude and longitude.
Updating an attribute value (console)

You can use the AWS IoT SiteWise console to update the value of an attribute asset property.

To update an attribute's value (console)
1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset for which you want to update an attribute.
   Tip
   You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose Edit.
5. Find the attribute to update, and then enter its new value.
6. Choose Save.

Updating an attribute value (CLI)

You can use the AWS Command Line Interface (AWS CLI) to update an attribute value.
You must know your asset’s assetId and property's propertyId to complete this procedure. If you created an asset but don’t know its assetId, use the ListAssets operation to view all of your assets for a specific model. Then, use the DescribeAsset operation to view your asset's properties including property IDs.

Use the BatchPutAssetPropertyValue operation to assign attribute values to your asset. You can use this operation to set multiple attributes at once. This operation's payload contains a list of entries, and each entry contains the asset ID, property ID, and attribute value.

To update an attribute’s value (CLI)

1. Create a file called batch-put-payload.json and copy the following JSON object into the file. This example payload demonstrates how to set a wind turbine's latitude and longitude. Update the IDs, values, and timestamps to modify the payload for your use case.

   ```json
   {
       "entries": [
           {
               "entryId": "windfarm3-turbine7-latitude",
               "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
               "propertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
               "propertyValues": [
                   {
                       "value": {
                           "doubleValue": 47.6204
                       },
                       "timestamp": {
                           "timeInSeconds": 1575691200
                       }
                   }
               ],
           },
           {
               "entryId": "windfarm3-turbine7-longitude",
               "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
               "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE",
               "propertyValues": [
                   {
                       "value": {
                           "doubleValue": 122.3491
                       },
                       "timestamp": {
                           "timeInSeconds": 1575691200
                       }
                   }
               ]
           }
       ]
   }
   ```

Each entry in the payload contains an entryId that you can define as any unique string. If any request entries fail, each error will contain the entryId of the corresponding request so that you know which requests to retry.

To set an attribute value, you can include one timestamp-quality-value (TQV) structure in the list of propertyValues for each attribute property. This structure must contain the new value and the current timestamp.

- value – A structure that contains one of the following fields, depending on the type of the property being set:
  - booleanValue
Associating assets

- doubleValue
- integerValue
- stringValue
- timestamp – A structure that contains the current Unix epoch time in seconds, timeInSeconds. AWS IoT SiteWise rejects any data points with timestamps older than 7 days in the past or newer than 5 minutes in the future.

For more information about how to prepare a payload for BatchPutAssetPropertyValue, see Ingesting data using the AWS IoT SiteWise API (p. 77).

2. Run the following command to send the attribute values to AWS IoT SiteWise.

```
aws iotsitewise batch-put-asset-property-value --cli-input-json file://batch-put-payload.json
```

Associating and disassociating assets

If your asset's model defines any child asset model hierarchies, you can associate child assets to your asset. Parent assets can access and aggregate data from associated assets. For more information about hierarchical asset models, see Defining relationships between assets (hierarchies) (p. 198).

**Topics**

- Associating and disassociating assets (console) (p. 207)
- Associating and disassociating assets (CLI) (p. 209)

## Associating and disassociating assets (console)

You can use the AWS IoT SiteWise console to associate and disassociate assets.

**To associate an asset (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**.
3. Choose the parent asset for which you want to associate a child asset.

  **Tip**
  
  You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose **Edit**.

5. In **Assets associated to this asset**, choose **Add associated asset**.

6. For **Hierarchy**, choose the hierarchy that defines the relationship between the parent asset and the child asset.

7. For **Asset**, choose the child asset to associate.

8. Choose **Save**.

**To disassociate an asset (console)**

1. Navigate to the **AWS IoT SiteWise console**.
2. In the navigation pane, choose **Assets**.
3. Choose the parent asset for which you want to disassociate a child asset.

   **Tip**
   You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose **Edit**.
5. In **Assets associated to this asset**, choose **Disassociate** for the asset.

6. Choose **Save**.

### Associating and disassociating assets (CLI)

You can use the AWS Command Line Interface (AWS CLI) to associate and disassociate assets.

For this procedure, you must know the ID of the hierarchy (`hierarchyId`) in the parent asset model that defines the relationship to the child asset model. Use the `DescribeAsset` operation to find the hierarchy ID in the response.

**To find a hierarchy ID**

- Run the following command to describe the parent asset. Replace `parent-asset-id` with the parent asset's ID.

```bash
aws iotsitewise describe-asset --asset-id parent-asset-id
```

The operation returns a response that contains the asset's details. The response contains an `assetHierarchies` list that has the following structure.

```json
{
  ...
}
```
The hierarchy ID is the id value for a hierarchy in the list of asset hierarchies.

After you have the hierarchy ID, you can associate or disassociate an asset with that hierarchy.

To associate a child asset to a parent asset, use the `AssociateAssets` operation. To disassociate a child asset from a parent asset, use the `DisassociateAssets` operation. Specify the following parameters, which are the same for both operations:

- `assetId` – The parent asset's ID.
- `hierarchyId` – The hierarchy ID in the parent asset.
- `childAssetId` – The child asset's ID.

**To associate an asset (CLI)**

- Run the following command to associate a child asset to a parent asset. Replace `parent-asset-id`, `hierarchy-id`, and `child-asset-id` with the respective IDs.

```
aws iotsitewise associate-assets
   --asset-id parent-asset-id
   --hierarchy-id hierarchy-id
   --child-asset-id child-asset-id
```

**To disassociate an asset (CLI)**

- Run the following command to disassociate a child asset from a parent asset. Replace `parent-asset-id`, `hierarchy-id`, and `child-asset-id` with the respective IDs.

```
aws iotsitewise disassociate-assets
   --asset-id parent-asset-id
   --hierarchy-id hierarchy-id
   --child-asset-id child-asset-id
```

### Updating assets and models

You can update your assets and asset models in AWS IoT SiteWise to modify their names and definitions. These update operations are asynchronous and take time to propagate through AWS IoT SiteWise. Check the status of the asset or asset model before you make additional changes. You must wait until the changes propagate before you can continue to use the updated asset or model.

**Topics**

- Updating assets (p. 211)
- Updating asset models (p. 212)
Updating assets

You can use the AWS IoT SiteWise console or API to update an asset's name.

When you update an asset, the asset's status is UPDATING until the changes propagate. For more information, see Asset and model states (p. 140).

**Topics**
- Updating an asset (console) (p. 211)
- Updating an asset (CLI) (p. 212)

Updating an asset (console)

You can use the AWS IoT SiteWise console to update asset details.

**To update an asset (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to update.

   **Tip**
   You can choose the arrow icon to expand an asset hierarchy to find your asset.

4. Choose Edit.
5. Update the asset's Name.
6. (Optional) On this page, update other information for the asset. For more information, see the following:
   - Mapping industrial data streams to asset properties (p. 202)
   - Updating attribute values (p. 204)
   - Interacting with other AWS services (p. 291)
7. Choose Save.
Updating an asset (CLI)

You can use the AWS CLI to update an asset's name.

Use the `UpdateAsset` operation to update an asset. Specify the following parameters:

- `assetId` – The asset's ID.
- `assetName` – The asset's new name.

To update an asset's name (CLI)

- Run the following command to update an asset's name. Replace `asset-id` with the ID of the asset and `asset-name` with the new name for the asset.

```
aws iotsitewise update-asset \\
   --asset-id asset-id \\
   --asset-name asset-name
```

Updating asset models

You can use the AWS IoT SiteWise console or API to update an asset model.

You can't change the type or data type of an existing property. You also can't change the window of an existing metric.

**Important**

If you remove a property from an asset model, AWS IoT SiteWise deletes all previous data for that property. If you remove a hierarchy definition from an asset model, AWS IoT SiteWise disassociates all assets in that hierarchy.

When you update an asset model, every asset based on that model reflects any changes that you make to the underlying model. Until the changes propagate, each asset has the `UPDATING` state. You must wait until those assets return to the `ACTIVE` state before you interact with them. During this time, the updated asset model's status will be `PROPAGATING`. For more information, see Asset and model states (p. 140).

**Topics**

- Updating an asset model (console) (p. 212)
- Updating an asset model (CLI) (p. 213)

Updating an asset model (console)

You can use the AWS IoT SiteWise console to update an asset model.

**To update an asset model (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Models.
3. Choose the asset model to update.
4. Choose Edit.
5. On the Edit model page, do any of the following:
• In **Model details**, change the **Name** of the model.
• Change any of the **Attribute definitions**. You can't change the **Data type** of existing attributes. For more information, see Defining static data (attributes) (p. 151).
• Change any of the **Measurement definitions**. You can't change the **Data type** of existing measurements. For more information, see Defining data streams from equipment (measurements) (p. 152).
• Change any of the **Transform definitions**. For more information, see Transforming data (transforms) (p. 154).
• Change any of the **Metric definitions**. You can't change the **Time interval** of existing metrics. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).
• Change any of the **Hierarchy definitions**. You can't change the **Hierarchy model** of existing hierarchies. For more information, see Defining relationships between assets (hierarchies) (p. 198).

6. Choose **Save**.

### Updating an asset model (CLI)

You can use the AWS CLI to update an asset model.

Use the **UpdateAssetModel** operation to update an asset model's name, description, properties, and hierarchies. Specify the following parameter:

- **assetModelId** – The asset model's ID.

Specify the updated asset model in the payload. To learn about the expected format of an asset model, see Creating asset models (p. 143).

**Warning**

The **UpdateAssetModel** operation overwrites the existing model with the model that you provide in the payload. To avoid deleting your asset model's properties or hierarchies, you must include their IDs and definitions in the updated asset model payload. To learn how to query your model's existing structure, see the **DescribeAssetModel** operation.

#### To update an asset model (CLI)

1. Run the following command to retrieve the existing asset model definition. Replace **asset-model-id** with the ID of the asset model to update.

   ```
   aws iotsitewise describe-asset-model --asset-model-id asset-model-id
   ```

   The operation returns a response that contains the asset model's details. The response has the following structure.

   ```
   {
   "assetModelId": "String",
   "assetModelArn": "String",
   "assetModelName": "String",
   "assetModelDescription": "String",
   "assetModelProperties": Array of AssetModelProperty,
   "assetModelHierarchies": Array of AssetModelHierarchyDefinition,
   "assetModelCompositeModels": Array of AssetModelCompositeModel,
   "assetModelCreationDate": "String",
   "assetModelLastUpdateDate": "String",
   "assetModelStatus": {
   ```
Updating asset models

For more information, see the DescribeAssetModel operation.

2. Create a file called update-asset-model.json and copy the previous command's response into the file.

3. Remove the following key-value pairs from the JSON object in update-asset-model.json:
   - assetModelId
   - assetModelArn
   - assetModelCreationDate
   - assetModelLastUpdateDate
   - assetModelStatus

The UpdateAssetModel operation expects a payload with the following structure.

```
{
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": Array of AssetModelProperty,
  "assetModelHierarchies": Array of AssetModelHierarchyDefinition,
  "assetModelCompositeModels": Array of AssetModelCompositeModel
}
```

4. In update-asset-model.json, do any of the following:
   - Change the asset model's name (assetModelName).
   - Change, add, or remove the asset model's description (assetModelDescription).
   - Change, add, or remove any of the asset model's properties (assetModelProperties). You can't change the dataType of existing properties or the window of existing metrics. For more information, see Defining data properties (p. 150).
   - Change, add, or remove any of the asset model's hierarchies (assetModelHierarchies). You can't change the childAssetModelId of existing hierarchies. For more information, see Defining relationships between assets (hierarchies) (p. 198).
   - Change, add, or remove any of the asset model's composite models (assetModelCompositeModels), such as alarm definitions. Alarms monitor other properties so that you can identify when equipment or processes require attention. Each alarm definition is a composite model that standardizes the set of properties that the alarm uses. For more information, see Monitoring data with alarms (p. 220) and Defining alarms on asset models (p. 223).

5. Run the following command to update the asset model with the definition stored in update-asset-model.json. Replace asset-model-id with the ID of the asset model.

```
aws iotsitewise update-asset-model \
--asset-model-id asset-model-id \
--cli-input-json file://model-payload.json
```
Deleting assets and models

You can delete your assets and models from AWS IoT SiteWise when you're done with them. The delete operations are asynchronous and take time to propagate through AWS IoT SiteWise.

Topics
- Deleting assets (p. 215)
- Deleting asset models (p. 217)

Deleting assets

You can use the AWS IoT SiteWise console or API to delete an asset.

Before you can delete an asset, you must first disassociate its child assets and disassociate it from its parent asset. For more information, see Associating and disassociating assets (p. 207). If you use the AWS CLI, you can use the ListAssociatedAssets operation to list an asset's children.

When you delete an asset, its status is DELETING until the changes propagate. For more information, see Asset and model states (p. 140). After the asset is deleted, you can't query that asset. If you do, the API returns an HTTP 404 response.

Important
AWS IoT SiteWise deletes all property data for deleted assets.

Topics
- Deleting an asset (console) (p. 215)
- Deleting an asset (CLI) (p. 217)

Deleting an asset (console)

You can use the AWS IoT SiteWise console to delete an asset.

To delete an asset (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to delete.

Tip
You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. If the asset has any Associated assets, delete each asset. You can choose an asset's name to navigate to its page, where you can delete it.

<table>
<thead>
<tr>
<th>Associated assets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hierarchy</strong></td>
</tr>
<tr>
<td>Turbine Asset Model</td>
</tr>
<tr>
<td>Turbine Asset Model</td>
</tr>
<tr>
<td>Turbine Asset Model</td>
</tr>
<tr>
<td>Turbine Asset Model</td>
</tr>
</tbody>
</table>

5. On the asset's page, choose **Delete**.
6. In the **Delete asset** dialog, do the following:
   a. Enter **Delete** to confirm deletion.
   b. Choose **Delete**.
Deleting an asset (CLI)

You can use the AWS Command Line Interface (AWS CLI) to delete an asset.

Use the `DeleteAsset` operation to delete an asset. Specify the following parameter:

- `assetId` – The asset's ID.

To delete an asset (CLI)

1. Run the following command to list the asset's hierarchies. Replace `asset-id` with the ID of the asset.

   ```bash
   aws iotsitewise describe-asset --asset-id asset-id
   ```

   The operation returns a response that contains the asset's details. The response contains an `assetHierarchies` list that has the following structure.

   ```json
   {
     "assetHierarchies": [
       {
         "id": "String",
         "name": "String"
       }
     ],
     ...
   }
   ```

   For more information, see the `DescribeAsset` operation.

2. For each hierarchy, run the following command to list the asset's children that are associated with that hierarchy. Replace `asset-id` with the ID of the asset and `hierarchy-id` with the ID of the hierarchy.

   ```bash
   aws iotsitewise list-associated-assets \
   --asset-id asset-id \
   --hierarchy-id hierarchy-id
   ```

   For more information, see the `ListAssociatedAssets` operation.

3. Run the following command to delete each associated asset and then to delete the asset. Replace `asset-id` with the ID of the asset.

   ```bash
   aws iotsitewise delete-asset --asset-id asset-id
   ```

Deleting asset models

You can use the AWS IoT SiteWise console or API to delete an asset model.

Before you can delete an asset model, you must first delete all assets created from the asset model.

When you delete an asset model, its status is `DELETING` until the changes propagate. For more information, see Asset and model states (p. 140). After the asset model is deleted, you can't query that asset model. If you do, the API returns an HTTP 404 response.
Deleting an asset model (console)

You can use the AWS IoT SiteWise console to delete an asset model.

**To delete an asset model (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model to delete.
4. If the model has any **Assets**, delete each asset. Choose an asset's name to navigate to its page, where you can delete it. For more information, see Deleting an asset (console) (p. 215).

5. On the model's page, choose **Delete**.
6. In the **Delete model** dialog, do the following:
   a. Enter **Delete** to confirm deletion.
   b. Choose **Delete**.

Deleting an asset model (CLI)

You can use the AWS CLI to delete an asset model.

Use the **DeleteAssetModel** operation to delete an asset model. Specify the following parameter:

- **assetModelId** – The asset model's ID.

**To delete an asset model (CLI)**

1. Run the following command to list all assets created from the model. Replace `asset-model-id` with the ID of the asset model.
aws iotsitewise list-assets --asset-model-id asset-model-id

For more information, see the ListAssets operation.

2. If the previous command returns any assets from the model, delete each asset. For more information, see Deleting an asset (CLI) (p. 217).

3. Run the following command to delete the asset model. Replace asset-model-id with the ID of the asset model.

aws iotsitewise delete-asset-model --asset-model-id asset-model-id
Monitoring data with alarms

You can configure alarms for your data to alert your team when equipment or processes perform sub-optimally. Optimal performance of a machine or process means that the values for certain metrics should be within a range of high and low limits. When these metrics are outside their operating range, equipment operators must be notified so they can fix the issue. Use alarms to quickly identify issues and notify operators to maximize performance of your equipment and processes.

Topics
- Alarm types (p. 220)
- Alarm states (p. 221)
- Alarm state properties (p. 221)
- Defining alarms on asset models (p. 223)
- Configuring alarms on assets (p. 247)
- Responding to alarms (p. 249)
- Ingesting external alarm state (p. 254)

Alarm types

You can define alarms that detect in the AWS Cloud and alarms that you detect with external processes. AWS IoT SiteWise supports the following types of alarms:

- **AWS IoT Events alarms**

  AWS IoT Events alarms are alarms that detect in AWS IoT Events. AWS IoT SiteWise sends asset property values to an alarm model in AWS IoT Events. Then, AWS IoT Events sends the alarm state to AWS IoT SiteWise. You can configure options such as when the alarm detects and whom to notify when the alarm state changes. You can also define the AWS IoT Events actions that occur when the alarm state changes.

  Alarms in AWS IoT Events are instances of alarm models. The alarm model specifies the threshold and severity of the alarm, what to do when the alarm state changes, and more. When you configure each trait of the alarm model, you specify an attribute property from the asset model that the alarm monitors. All assets based on the asset model use the value of the attribute when AWS IoT Events evaluates that trait of the alarm. For more information, see Using alarms in the AWS IoT Events Developer Guide.

  You can respond to an AWS IoT Events alarm when it changes state. For example, you can acknowledge or snooze an alarm when it becomes active. You can also enable, disable, and reset alarms.

  SiteWise Monitor users can visualize, configure, and respond to AWS IoT Events alarms in SiteWise Monitor portals. For more information, see Monitoring with alarms in the AWS IoT SiteWise Monitor Application Guide.

  **Note**
  AWS IoT Events charges apply to evaluate these alarms and transfer data between AWS IoT SiteWise and AWS IoT Events. For more information, see AWS IoT Events pricing.

- **External alarms**

  External alarms are alarms that you evaluate outside of AWS IoT SiteWise. Use external alarms if you have a data source that reports alarm state. The external alarm contains a measurement property to which you ingest the alarm state data.
You can't acknowledge or snooze an external alarm when it changes state.

SiteWise Monitor users can see the state of external alarms in SiteWise Monitor portals, but they can't configure or respond to these alarms.

AWS IoT SiteWise doesn't evaluate the state of external alarms.

# Alarm states

Industrial alarms include information about the state of the equipment or process they monitor and (optional) information about the operator's response to the alarm state.

When you define an AWS IoT Events alarm, you specify whether or not to enable the *acknowledge flow*. The acknowledge flow is enabled by default. When you enable this option, operators can acknowledge the alarm and leave a note with details about the alarm or the actions they took to address it. If an operator doesn't acknowledge an active alarm before it becomes inactive, the alarm becomes latched. The latched state indicates that the alarm became active and wasn't acknowledged, so an operator needs to check on the equipment or process and acknowledge the latched alarm.

Alarms have the following states:

- **Normal** (Normal) – The alarm is enabled but inactive. The industrial process or equipment operates as expected.
- **Active** (Active) – The alarm is active. The industrial process or equipment is outside its operating range and needs attention.
- **Acknowledged** (Acknowledged) – An operator acknowledged the state of the alarm. This state applies to only alarms where you enable the acknowledge flow.
- **Latched** (Latched) – The alarm returned to normal but was active and no operator acknowledged it. The industrial process or equipment requires attention from an operator to reset the alarm to normal. This state applies to only alarms where you enable the acknowledge flow.
- **Snoozed** (SnoozeDisabled) – The alarm is disabled because an operator snoozed the alarm. The operator defines the duration for which the alarm snoozes. After that duration, the alarm returns to normal state.
- **Disabled** (Disabled) – The alarm is disabled and won't detect.

# Alarm state properties

AWS IoT SiteWise stores alarm state data as a JSON object serialized to a string. This object contains the state and additional information about the alarm, such as operator response actions and the rule that the alarm evaluates.

You identify the alarm state property by its name and structure type, `AWS/ALARM_STATE`. For more information, see *Defining alarms on asset models* (p. 223).

The alarm state data object contains the following information:

- **stateName**
  - The state of the alarm. For more information, see *Alarm states* (p. 221).

  Data type: STRING
customerAction

(Optional) An object that contains information about an operator's response to the alarm. Operators can enable, disable, acknowledge, and snooze alarms. When they do so, the alarm state data includes their response and the note that they can leave when they respond. This object contains the following information:

actionName

The name of the action that the operator takes to respond to the alarm. This value contains one of the following strings:
- ENABLE
- DISABLE
- SNOOZE
- ACKNOWLEDGE
- RESET

Data type: STRING

enable

(Optional) An object that is present in customerAction when the operator enables the alarm. When an operator enables the alarm, the alarm state changes to Normal. This object contains the following information:

note

(Optional) The note that the customer leaves when they enable the alarm.

Data type: STRING

Maximum length: 128 characters

disable

(Optional) An object that is present in customerAction when the operator disables the alarm. When an operator enables the alarm, the alarm state changes to Disabled. This object contains the following information:

note

(Optional) The note that the customer leaves when they disable the alarm.

Data type: STRING

Maximum length: 128 characters

acknowledge

(Optional) An object that is present in customerAction when the operator acknowledges the alarm. When an operator enables the alarm, the alarm state changes to Acknowledged. This object contains the following information:

note

(Optional) The note that the customer leaves when they acknowledge the alarm.

Data type: STRING

Maximum length: 128 characters

snooze

(Optional) An object that is present in customerAction when the operator snoozes the alarm. When an operator enables the alarm, the alarm state changes to SnoozeDisabled. This object contains the following information:
snoozeDuration

The duration in seconds that the operator snoozes the alarm. The alarm changes to Normal state after this duration.

Data type: INTEGER

note

(Optional) The note that the customer leaves when they snooze the alarm.

Data type: STRING

Maximum length: 128 characters

ruleEvaluation

(Optional) An object that contains information about the rule that evaluates the alarm. This object contains the following information:

simpleRule

An object that contains information about a simple rule, which compares a property value to a threshold value with a comparison operator. This object contains the following information:

inputProperty

The value of the property that this alarm evaluates.

Data type: DOUBLE

operator

The comparison operator that this alarm uses to compare the property with the threshold. This value contains one of the following strings:

- `<` – Less than
- `<=` – Less than or equal
- `==` – Equal
- `!=` – Not equal
- `>=` – Greater than or equal
- `>` – Greater than

Data type: STRING

threshold

The threshold value that this alarm compares the property value against.

Data type: DOUBLE

---

**Defining alarms on asset models**

Asset models drive standardization of your industrial data and alarms. You can define alarm definitions on asset models to standardize the alarms for all assets based on an asset model.

You use composite asset models to define alarms on asset models. Composite asset models are asset models that standardize a specific set of properties on another asset model. Composite asset models ensure that certain properties are present on an asset model. Alarms have type, state, and (optional) source properties, so the alarm composite model enforces that these properties exist.
Each composite asset model has a type that defines the properties for that composite model. Alarm composite models define properties for alarm type, alarm state, and (optional) alarm source. When you create an asset from an asset model with composite models, the asset includes the properties from the composite model alongside the properties that you specify in the asset model.

Each property in a composite model must have the name that identifies it for its type of composite model. Composite model properties support properties with complex data types. These properties have the `STRUCT` data type and a `dataTypeSpec` trait that specifies the complex data type of the property. Complex data type properties contain JSON data serialized as strings.

Alarm composite models have the following properties. Each property must have the name that identifies it for this type of composite model.

**Alarm type**

The type of the alarm. Specify one of the following:
- `IOT_EVENTS` – An AWS IoT Events alarm. AWS IoT SiteWise sends data to AWS IoT Events to evaluate the state of this alarm. You must specify the alarm source property to define the AWS IoT Events alarm model for this alarm definition.
- `EXTERNAL` – An external alarm. You ingest the state of the alarm as a measurement.

Property name: `AWS/ALARM_TYPE`

Property type: attribute (p. 151)

Data type: STRING

**Alarm state**

The time series data for the state of the alarm. This is an object serialized as a string that contains the state and other information about the alarm. For more information, see Alarm state properties (p. 221).

Property name: `AWS/ALARM_STATE`

Property type: measurement (p. 152)

Data type: STRUCT

Data structure type: `AWS/ALARM_STATE`

**Alarm source**

(Optional) The Amazon Resource Name (ARN) of the resource that evaluates the state of the alarm. For AWS IoT Events alarms, this is the ARN of the alarm model.

Property name: `AWS/ALARM_SOURCE`

Property type: attribute (p. 151)

Data type: STRING

**Example Example alarm composite model**

The following asset model represents a boiler that has an alarm to monitor its temperature. AWS IoT SiteWise sends the temperature data to AWS IoT Events to detect the alarm.

```json
{
  "assetModelName": "Boiler",
  "assetModelDescription": "A boiler that alarms when its temperature exceeds its limit."
}
```
Defining AWS IoT Events alarms

When you create an AWS IoT Events alarm, AWS IoT SiteWise sends asset property values to AWS IoT Events to evaluate the state of the alarm. AWS IoT Events alarm definitions depend on an alarm model.
that you define in AWS IoT Events. To define an AWS IoT Events alarm on an asset model, you define an alarm composite model that specifies the AWS IoT Events alarm model as its alarm source property.

AWS IoT Events alarms depend on inputs such as alarm thresholds and alarm notification settings. You define these inputs as attributes on the asset model. You can then customize these inputs on each asset based on the model. The AWS IoT SiteWise console can create these attributes for you. If you define alarms with the AWS CLI or API, you must manually define these attributes on the asset model.

You can also define other actions that happen when your alarm detects, such as custom alarm notification actions. For example, you can configure an action that sends a push notification to an Amazon SNS topic. For more information the actions that you can define, see Working with other AWS services in the AWS IoT Events Developer Guide.

When you update or delete an asset model, AWS IoT SiteWise can check if an alarm model in AWS IoT Events is monitoring an asset property associated with this asset model. This prevents you from deleting an asset property that an AWS IoT Events alarm is currently using. To enable this feature in AWS IoT SiteWise, you must have the iotevents:ListInputRoutings permission. This permission allows AWS IoT SiteWise to make calls to the ListInputRoutings API operation supported by AWS IoT Events. For more information, see (Optional) ListInputRoutings permission (p. 349).

Note
The alarm notifications feature isn't available in the China (Beijing) Region.

Topics
• Requirements for alarm notifications (p. 226)
• Defining an AWS IoT Events alarm (console) (p. 226)
• Defining an AWS IoT Events alarm (CLI) (p. 229)

Requirements for alarm notifications

AWS IoT Events uses an AWS Lambda function in your AWS account to send alarm notifications. You must create this Lambda function in the same AWS Region as your alarms to enable alarm notifications. This Lambda function uses Amazon Simple Notification Service (Amazon SNS) to send text notifications and Amazon Simple Email Service (Amazon SES) to send email notifications. When you create the AWS IoT Events alarm, you configure the protocols and settings that the alarm uses to send notifications.

AWS IoT Events provides an AWS CloudFormation stack template that you can use to create this Lambda function in your account. For more information, see Alarm notification Lambda function in the AWS IoT Events Developer Guide.

Defining an AWS IoT Events alarm (console)

You can use the AWS IoT SiteWise console to define an AWS IoT Events alarm on an existing asset model. To define an AWS IoT Events alarm on a new asset model, create the asset model, and then complete these steps. For more information, see Creating asset models (p. 143).

Important
Each alarm requires an attribute that specifies the threshold value to compare against for the alarm. You must define the threshold value attribute on the asset model before you can define an alarm.

Consider an example where you want to define an alarm that detects when a wind turbine exceeds its maximum wind speed rating of 50 mph. Before you define the alarm, you must define an attribute (Maximum wind speed) with a default value of 50.

To define an AWS IoT Events alarm on an asset model
1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model for which to define an alarm.
4. Choose the **Alarm definitions** tab.
5. Choose **Add alarm**.
6. In **Alarm type options**, choose **AWS IoT Events alarm**.
7. Enter a name for your alarm.
8. (Optional) Enter a description for your alarm.
9. In the **Threshold definitions** pane, you define when the alarm detects and the severity of the alarm. Do the following:
   a. Select the **Property** on which the alarm detects. Each time this property receives a new value, AWS IoT SiteWise sends the value to AWS IoT Events to evaluate the state of the alarm.
   b. Select the **Operator** to use to compare the property with the threshold value. Choose from the following options:
      - `< less than`
      - `<= less than or equal`
      - `== equal`
      - `!= not equal`
      - `>= greater than or equal`
      - `>` greater than
   c. Select the attribute property to use as the threshold **Value**. AWS IoT Events compares the value of the property with the value of this attribute.
   d. Enter the **Severity** of the alarm. Use a number that your team understands to reflect the severity of this alarm.
10. (Optional) In the **Notification settings** pane, you can configure the notification settings for the alarm. The alarm uses an AWS Lambda function in your AWS account to manage alarm notifications. For more information, see [Requirements for alarm notifications](p. 226).

   In this pane, you configure this Lambda function, the message protocol, the message recipient, and the custom message that AWS IoT Events sends when this alarm is invoked. You define the recipient and custom message in an attribute property, so you can later customize these values for each asset based on this model. You can use an existing attribute or create an attribute for each setting. If you create an attribute, you can define its default value for all assets based on this asset model.

   In the **Notification settings** pane, do the following:
   a. Choose **Enabled**.
      
      **Note**
      If you choose **Disabled**, you and your team won't receive any alarm notifications.
   b. For **Recipient**, choose the recipient.
      
      **Important**
      You can send alarm notifications to AWS Single Sign-On users. To use this feature, you must enable AWS SSO. You can only enable AWS SSO in one AWS Region at a time. This means that you can define alarm notifications only in the Region where you enable AWS SSO. For more information, see [Getting started](in the AWS Single Sign-On User Guide).
   c. For **Protocol**, choose from the following options:
      - **Email & text** – The alarm notifies AWS SSO users with an SMS message and an email message.
      - **Email** – The alarm notifies AWS SSO users with an email message.
• **Text** – The alarm notifies AWS SSO users with an SMS message.

d. For **Sender**, choose the sender.

**Important**

You must verify the sender email address in Amazon Simple Email Service (Amazon SES). For more information, see Verifying email addresses in Amazon SES, in the *Amazon Simple Email Service Developer Guide*.

11. Specify the **Default asset state** for this alarm. You can enable or disable this alarm for all assets that you create from this asset model in a later step.

12. In the **Advanced settings** pane, you can configure the permissions, the additional notification settings, the alarm state actions, the alarm model in SiteWise Monitor, and the acknowledge flow.

**Note**

AWS IoT Events alarms require the following service roles:

- A role that AWS IoT Events assumes to send alarm state values to AWS IoT SiteWise.
- A role that AWS IoT Events assumes to send data to Lambda. You only need this role if your alarm sends notifications.

In the **Permissions** pane, do the following:

a. For **AWS IoT Events role**, use an existing role or create a role with the required permissions. This role requires the iotsitewise:BatchPutAssetPropertyValue permission and a trust relationship that allows iotevents.amazonaws.com to assume the role.

b. For the **AWS IoT Events Lambda role**, use an existing role or create a role with the required permissions. This role requires the lambda:InvokeFunction and sso-directory:DescribeUser permissions and a trust relationship that allows iotevents.amazonaws.com to assume the role.

13. (Optional) In the **Additional notification settings** pane, do the following:

a. For **Recipient attribute**, you define an attribute whose value specifies the recipient of the notification. You can choose AWS SSO users as recipients.

You can create an attribute or use an existing attribute on the asset model.

- If you choose to create an attribute, specify the **Attribute name** and **Default value** for the attribute.

- If you choose to use an existing attribute, choose the attribute in **Attribute name**. The alarm uses the default value of the attribute that you choose.

You can override the default value on each asset that you create from this asset model.

b. For **Custom message attribute**, you define an attribute whose value specifies the custom message to send in addition to the default state change message. For example, you can specify a message that helps your team understand how to address this alarm.

You can choose to create an attribute or use an existing attribute on the asset model.

- If you choose to create an attribute, specify the **Attribute name** and **Default value** for the attribute.

- If you choose to use an existing attribute, choose the attribute in **Attribute name**. The alarm uses the default value of the attribute that you choose.

You can override the default value on each asset that you create from this asset model.

c. For **Function name**, choose an existing Lambda function or create a function that manages alarm notifications. For more information, see Managing alarm notifications in the *AWS IoT Events Developer Guide*.

14. (Optional) In the **Set state action** pane, do the following:
a. Choose **Edit action**.
b. On the **Add alarm state actions** page, add actions. You can add up to 10 actions.

AWS IoT Events can perform actions when the alarm is active. You can define built-in actions to use a timer or set a variable, or send data to other AWS resources. For more information, see **Supported actions** in the *AWS IoT Events Developer Guide*.

15. (Optional) In the **Manage alarm model in SiteWise Monitor** pane, choose **Enabled** or **Disabled**.

Use this option so that you can update the alarm model in SiteWise Monitor. This option is enabled by default.

16. In **Acknowledge flow**, you can specify if the acknowledge flow is enabled. For more information about the acknowledge flow, see **Alarm states** (p. 221).

17. Choose **Add alarm**.

**Note**
The AWS IoT SiteWise console makes multiple API requests to add the alarm to the asset model. When you choose **Add alarm**, the console opens a dialog box that shows the progress of these API requests. Stay on this page until each API request succeeds or until an API request fails. If a request fails, close the dialog box, fix the issue, and choose **Add alarm** to try again.

---

**Defining an AWS IoT Events alarm (CLI)**

You can use the AWS Command Line Interface (AWS CLI) to define an AWS IoT Events alarm that monitors an asset property. You can define the alarm on a new or existing asset model. After you define the alarm on the asset model, you create an alarm in AWS IoT Events and connect it to the asset model. In this process, you do the following:

**Steps**

- **Step 1: Defining an alarm on an asset model** (p. 229)
- **Step 2: Defining an AWS IoT Events alarm model** (p. 235)
- **Step 3: Enabling data flow between AWS IoT SiteWise and AWS IoT Events** (p. 244)

**Step 1: Defining an alarm on an asset model**

Add an alarm definition and associated properties to a new or existing asset model.

**To define an alarm on an asset model (CLI)**

1. Create a file called `asset-model-payload.json`. Follow the steps in these other sections to add your asset model's details to the file, but don't submit the request to create or update the asset model. In this section, you add an alarm definition to the asset model details in the `asset-model-payload.json` file.
   - **Note**
     Your asset model must define at least one asset property, including the asset property to monitor with the alarm.
2. Add an alarm composite model (`assetModelCompositeModels`) to the asset model. An AWS IoT Events alarm composite model specifies the `IOT_EVENTS` type and specifies an alarm source property. You add the alarm source property after you create the alarm model in AWS IoT Events.

**Important**
The alarm composite model must have the same name as the AWS IoT Events alarm model you create later. Alarm model names can contain only alphanumeric characters. Specify a unique, alphanumeric name so that you can use the same name for the alarm model.

```json
{
  ...
  "assetModelCompositeModels": [
    {
      "name": "BoilerTemperatureHighAlarm",
      "type": "AWS/ALARM",
      "properties": [
        {
          "name": "AWS/ALARM_TYPE",
          "dataType": "STRING",
          "type": {
            "attribute": {
              "defaultValue": "IOT_EVENTS"
            }
          }
        },
        {
          "name": "AWS/ALARM_STATE",
          "dataType": "STRUCT",
          "dataTypeSpec": "AWS/ALARM_STATE",
          "type": {
            "measurement": {
            }
          }
        }
      ]
    }
  ]
}
```

3. Add an alarm threshold attribute to the asset model. Specify the default value to use for this threshold. You can override this default value on each asset based on this model.

**Note**
The alarm threshold attribute must be an INTEGER or a DOUBLE.

```json
{
  ...
  "assetModelProperties": [
    {
      "name": "Temperature Max Threshold",
      "dataType": "DOUBLE",
      "type": {
        "attribute": {
          "defaultValue": "105.0"
        }
      }
    }
  ]
}
```

4. (Optional) Add alarm notification attributes to the asset model. These attributes specify the AWS SSO recipient and other inputs that AWS IoT Events uses to send notifications when the alarm changes state. You can override these defaults on each asset based on this model.
**Important**
You can send alarm notifications to AWS Single Sign-On users. To use this feature, you must enable AWS SSO. You can only enable AWS SSO in one AWS Region at a time. This means that you can define alarm notifications only in the Region where you enable AWS SSO. For more information, see Getting started in the AWS Single Sign-On User Guide.

Do the following:

a. Add an attribute that specifies the ID of your AWS SSO identity store. You can use the AWS SSO ListInstances API operation to list your identity stores. This operation works only in the Region where you enable AWS SSO.

```bash
aws sso-admin list-instances
```

Then, specify the identity store ID (for example, `d-123EXAMPLE`) as the default value for the attribute.

```json
{
...  "assetModelProperties": [
    ...  {
      "name": "identityStoreId",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "d-123EXAMPLE"
        }
      }
    }
  ]
}
```

b. Add an attribute that specifies the ID of the AWS SSO user who receives notifications. To define a default notification recipient, add an AWS SSO user ID as the default value. Do one of the following to get an AWS SSO user ID:

i. You can use the AWS SSO ListUsers API to get the ID of a user whose user name you know. Replace `d-123EXAMPLE` with the ID of your identity store, and replace `Name` with the user name of the user.

```bash
aws identitystore list-users \\  --identity-store-id d-123EXAMPLE \  --filters AttributePath=UserName,AttributeValue=Name
```

ii. Use the AWS SSO console to browse your users and find a user ID.

Then, specify the user ID (for example, `123EXAMPLE-a1b2c3d4-5678-90ab-cdef-33333EXAMPLE`) as the default value for the attribute, or define the attribute without a default value.

```json
{
...  "assetModelProperties": [
    ...  {
      "name": "userId",
      "dataType": "STRING",
```
Defining AWS IoT Events alarms

c. (Optional) Add an attribute that specifies the default sender ID for SMS (text) message notifications. The sender ID displays as the message sender on messages that Amazon Simple Notification Service (Amazon SNS) sends. For more information, see Requesting sender IDs for SMS messaging with Amazon SNS in the Amazon Simple Notification Service Developer Guide.

```
{ ...
  "assetModelProperties": [ ...
    { "name": "senderId",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "MyFactory"
        }
      }
    }
  ]
}
```

d. (Optional) Add an attribute that specifies the default email address to use as the from address in email notifications.

```
{ ...
  "assetModelProperties": [ ...
    { "name": "fromAddress",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "my.factory@example.com"
        }
      }
    }
  ]
}
```

e. (Optional) Add an attribute that specifies the default subject to use in email notifications.

```
{ ...
  "assetModelProperties": [ ...
    { "name": "emailSubject",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "[ALERT] High boiler temperature"
        }
      }
    }
  ]
}
```
f. (Optional) Add an attribute that specifies an additional message to include in notifications. By default, notification messages include information about the alarm. You can also include an additional message that gives the user more information.

```json
...
"assetModelProperties": [
  ...
  {
    "name": "additionalMessage",
    "dataType": "STRING",
    "type": {
      "attribute": {
        "defaultValue": "Turn off the power before you check the alarm."
      }...
  }
]
```

5. Create the asset model or update the existing asset model. Do one of the following:
   - To create the asset model, run the following command.
     ```bash
     aws iotsitewise create-asset-model --cli-input-json file://asset-model-payload.json
     ```
   - To update the existing asset model, run the following command. Replace `asset-model-id` with the ID of the asset model.
     ```bash
     aws iotsitewise update-asset-model \
     --asset-model-id asset-model-id \
     --cli-input-json file://asset-model-payload.json
     ```

   After you run the command, note the `assetModelId` in the response.

**Example: Boiler asset model**

The following asset model represents a boiler that reports temperature data. This asset model defines an alarm that detects when the boiler overheats.

```json
{
  "assetModelName": "Boiler Model",
  "assetModelDescription": "Represents a boiler.",
  "assetModelProperties": [
    {
      "name": "Temperature",
      "dataType": "DOUBLE",
      "unit": "C",
      "type": {
        "measurement": {}
      }
    },
    {
      "name": "Temperature Max Threshold",
      "dataType": "DOUBLE",
      "type": {
        "attribute": {
          "defaultValue": "Turn off the power before you check the alarm."
        }
      }
    }
  ]
}
"defaultValue": "105.0"
},

"name": "identityStoreId",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "d-123EXAMPLE"
    }
},

"name": "userId",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "123EXAMPLE-a1b2c3d4-5678-90ab-cdef-33333EXAMPLE"
    }
},

"name": "senderId",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "MyFactory"
    }
},

"name": "fromAddress",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "my.factory@example.com"
    }
},

"name": "emailSubject",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "[ALERT] High boiler temperature"
    }
},

"name": "additionalMessage",
"dataType": "STRING",
"type": {
    "attribute": {
        "defaultValue": "Turn off the power before you check the alarm."
    }
},

"assetModelHierarchies": [
],

"assetModelCompositeModels": [
    {
        "name": "BoilerTemperatureHighAlarm",
        "type": "AWS/ALARM",
        "defaultValue": "105.0"
    }
]
Step 2: Defining an AWS IoT Events alarm model

Create the alarm model in AWS IoT Events. In AWS IoT Events, you use expressions to specify values in alarm models. You can use expressions to specify values from AWS IoT SiteWise to evaluate and use as inputs to the alarm. When AWS IoT SiteWise sends asset property values to the alarm model, AWS IoT Events evaluates the expression to get the value of the property or the ID of the asset. You can use the following expressions in the alarm model:

- **Asset property values**

  To get the value of an asset property, use the following expression. Replace `assetModelId` with the ID of the asset model and replace `propertyId` with the ID of the property.

  ```
  $sitewise.assetModel.`assetModelId`.propertyId.propertyValue.value
  ```

- **Asset IDs**

  To get the ID of the asset, use the following expression. Replace `assetModelId` with the ID of the asset model and replace `propertyId` with the ID of the property.

  ```
  $sitewise.assetModel.`assetModelId`.propertyId.assetId
  ```

**Note**

When you create the alarm model, you can define literals instead of expressions that evaluate to AWS IoT SiteWise values. This can reduce the number of attributes that you define on your asset model. However, if you define a value as a literal, you can't customize that value on assets based on the asset model. Your AWS IoT SiteWise Monitor users also can't customize the alarm, because they can configure alarm settings on assets only.

**To create an AWS IoT Events alarm model (CLI)**

1. When you create the alarm model in AWS IoT Events, you must specify the ID of each property that the alarm uses, which includes the following:
   - The alarm state property in the composite asset model
   - The property that the alarm monitors
• The threshold attribute
• (Optional) The AWS SSO identity store ID attribute
• (Optional) The AWS SSO user ID attribute
• (Optional) The SMS sender ID attribute
• (Optional) The email from address attribute
• (Optional) The email subject attribute
• (Optional) The additional message attribute

Run the following command to retrieve the IDs of these properties on the asset model. Replace
`asset-model-id` with the ID of the asset model from the previous step.

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id
```

The operation returns a response that contains the asset model's details. Note the ID of each
property that the alarm uses. You use these IDs when you create the AWS IoT Events alarm model in
the next step.

2. Create the alarm model in AWS IoT Events. Do the following:
   a. Create a file called `alarm-model-payload.json`.
   b. Copy the following JSON object into the file.
   c. Enter a name (`alarmModelName`), description (`alarmModelDescription`), and severity
      (`severity`) for your alarm. For severity, specify an integer that reflects your company's severity
      levels.

```
{  
    "alarmModelName": "BoilerTemperatureHighAlarm",  
    "alarmModelDescription": "Detects when the boiler temperature is high.",  
    "severity": 3  
}
```

   d. Add the comparison rule (`alarmRule`) to the alarm. This rule defines the property to monitor
      (`inputProperty`), the threshold value to compare (`threshold`), and the comparison operator
to use (`comparisonOperator`).

```
• Replace `assetModelId` with the ID of the asset model.
• Replace `alarmModelPropertyId` with the ID of the property that the alarm monitors.
• Replace `thresholdAttributeValueId` with the ID of the threshold attribute property.
• Replace `GREATER` with the operator to use to compare the property values with the threshold.

Choose from the following options:
• LESS
• LESS_OR_EQUAL
• EQUAL
• NOT_EQUAL
• GREATER_OR_EQUAL
• GREATER
```
e. Add an action (alarmEventActions) to send alarm state to the AWS IoT SiteWise when the alarm changes state.

**Note**
For advanced configuration, you can define additional actions to perform when the alarm changes state. For example, you might call an AWS Lambda function or publish to an MQTT topic. For more information, see Working with other AWS services in the AWS IoT Events Developer Guide.

- Replace `assetModelId` with the ID of the asset model.
- Replace `alarmPropertyId` with the ID of the property that the alarm monitors.
- Replace `alarmStatePropertyId` with the ID of the alarm state property in the alarm composite model.

```

```
i. Add the alarm notification configuration (alarmNotification) to the payload in alarm-model-payload.json.

- Replace alarmNotificationFunctionArn with the ARN of the Lambda function that handles alarm notifications.

```json
{
    "alarmModelName": "BoilerTemperatureHighAlarm",
    "alarmModelDescription": "Detects when the boiler temperature is high.",
    "severity": 3,
    "alarmRule": {
        "simpleRule": {
            "inputProperty": "$sitewise.assetModel."assetModelId"."alarmPropertyId".propertyValue.value",
            "comparisonOperator": "GREATER",
            "threshold": "$sitewise.assetModel."assetModelId"."thresholdAttributeId".propertyValue.value"
        }
    },
    "alarmEventActions": {
        "alarmActions": [
            {
                "iotSiteWise": {
                    "assetId": "$sitewise.assetModel."assetModelId"."alarmPropertyId".assetId",
                    "propertyId": "alarmStatePropertyId"
                }
            }
        ],
    }
    "alarmNotification": {
        "notificationActions": [
            {
                "action": {
                    "lambdaAction": {
                        "functionArn": "alarmNotificationFunctionArn"
                    }
                }
            }
        ]
    }
}
```

ii. (Optional) Configure the SMS notifications (smsConfigurations) to send to an AWS SSO user when the alarm changes state.

- Replace identityStoreIdAttributeId with the ID of the attribute that contains the ID of the AWS SSO identity store.
- Replace userIdAttributeId with the ID of the attribute that contains the ID of the AWS SSO user.
- Replace senderIdAttributeId with the ID of the attribute that contains the Amazon SNS sender ID, or remove senderId from the payload.
- Replace additionalMessageAttributeId with the ID of the attribute that contains the additional message, or remove additionalMessage from the payload.

```json
{
    "alarmModelName": "BoilerTemperatureHighAlarm",
    "alarmModelDescription": "Detects when the boiler temperature is high.",
    "severity": 3,
```
iii. (Optional) Configure the email notifications (emailConfigurations) to send to an AWS SSO user when the alarm changes state.

- Replace `identityStoreIdAttributeId` with the ID of the AWS SSO identity store ID attribute property.
- Replace `userIdAttributeId` with the ID of the AWS SSO user ID attribute property.
- Replace `fromAddressAttributeId` with the ID of the "from" address attribute property, or remove from from the payload.
- Replace `emailSubjectAttributeId` with the ID of the email subject attribute property, or remove subject from the payload.
- Replace `additionalMessageAttributeId` with the ID of the additional message attribute property, or remove additionalMessage from the payload.
AWS IoT SiteWise User Guide
Defining AWS IoT Events alarms

```json
{
  "alarmModelName": "BoilerTemperatureHighAlarm",
  "alarmModelDescription": "Detects when the boiler temperature is high.",
  "severity": 3,
  "alarmRule": {
    "simpleRule": {
      "inputProperty": "$sitewise.assetModel."assetModelId"."alarmPropertyId".propertyValue.value",
      "comparisonOperator": "GREATER",
      "threshold": "$sitewise.assetModel."assetModelId"."thresholdAttributeId".propertyValue.value"
    },
    "alarmEventActions": {
      "alarmActions": [
        {
          "iotSiteWise": {
            "assetId": "$sitewise.assetModel."assetModelId"."alarmPropertyId".assetId",
            "propertyId": "'alarmStatePropertyId'"
          }
        }
      ],
      "alarmNotification": {
        "notificationActions": [
          {
            "action": {
              "lambdaAction": {
                "functionArn": "alarmNotificationFunctionArn"
              }
            },
            "smsConfigurations": [
              {
                "recipients": [
                  {
                    "ssoIdentity": {
                      "identityStoreId": "$sitewise.assetModel."assetModelId"."identityStoreIdAttributeId".propertyValue.value",
                      "userId": "$sitewise.assetModel."assetModelId"."userIdAttributeId".propertyValue.value"
                    }
                  }
                ],
                "senderId": "$sitewise.assetModel."assetModelId"."senderIdAttributeId".propertyValue.value",
                "additionalMessage": "$sitewise.assetModel."assetModelId"."additionalMessageAttributeId".propertyValue.value"
              }
            ],
            "emailConfigurations": [
              {
                "from": "$sitewise.assetModel."assetModelId"."fromAddressAttributeId".propertyValue.value",
                "recipients": [
                  {
                    "ssoIdentity": {
                      "identityStoreId": "$sitewise.assetModel."assetModelId"."identityStoreIdAttributeId".propertyValue.value",
                      "userId": "$sitewise.assetModel."assetModelId"."userIdAttributeId".propertyValue.value"
                    }
                  }
                ]
              }
            ]
          } // "action" object
        } // "alarmNotification" object
      } // "alarmEventActions" object
    } // "alarmRule" object
  } // "alarm" object
}
```
g. (Optional) Add the alarm capabilities (alarmCapabilities) to the payload in alarm-model-payload.json. In this object, you can specify if the acknowledge flow is enabled and the default enable state for assets based on the asset model. For more information about the acknowledge flow, see Alarm states (p. 221).

```json
{
    "alarmModelName": "BoilerTemperatureHighAlarm",
    "alarmModelDescription": "Detects when the boiler temperature is high.",
    "severity": 3,
    "alarmRule": {
        "simpleRule": {
            "inputProperty": "sitewise.assetModel.assetModelId.alarmPropertyId.propertyValue.value",
            "comparisonOperator": "GREATER",
            "threshold": "sitewise.assetModel.assetModelId.thresholdAttributeId.propertyValue.value"
        }
    },
    "alarmEventActions": {
        "alarmActions": [
            {"iotSiteWise": {
                "assetId": "sitewise.assetModel.assetModelId.alarmPropertyId.assetId",
                "propertyId": "alarmStatePropertyId"
            }}
        ],
        "alarmNotification": {
            "notificationActions": [
                {"action": {
                    "lambdaAction": {
                        "functionArn": "alarmNotificationFunctionArn"
                    }
                },
                "smsConfigurations": [
                    {"recipients": {
                        "ssoIdentity": {
                            "identityStoreId": "sitewise.assetModel.assetModelId.identityStoreIdAttributeId.propertyValue.value",
                            "userId": "sitewise.assetModel.assetModelId.userIdAttributeId.propertyValue.value"
                        }
                    }
                ]
            }
        }
    }
}```
h. Add the IAM service role (roleArn) that AWS IoT Events can assume to send data to AWS IoT SiteWise. This role requires the iotsitewise:BatchPutAssetPropertyValue permission and a trust relationship that allows iotevents.amazonaws.com to assume the role. To send notifications, this role also requires the lambda:InvokeFunction and ssodirectory:DescribeUser permissions. For more information, see Alarm service roles in the AWS IoT Events Developer Guide.

- Replace the roleArn with the ARN of the role that AWS IoT Events can assume to perform these actions.

```json
{
  "alarmModelName": "BoilerTemperatureHighAlarm",
  "alarmModelDescription": "Detects when the boiler temperature is high.",
  "severity": 3,
  "alarmRule": {
    "simpleRule": {
      "inputProperty": "$sitewise.assetModel."
```
"threshold": "$sitewise.assetModel.\"assetModelId\".\"thresholdAttributeId\".propertyValue.value",
"alarmEventActions": {
  "alarmActions": [
    { 'iotSiteWise': {
      "assetId": "$sitewise.assetModel.\"assetModelId\".\"alarmPropertyId\".assetId",
      "propertyId": "$sitewise.assetModel.\"alarmPropertyId\".propertyId"
    }
  ]
},
"alarmNotification": {
  "notificationActions": [
    { 'action': {
      "lambdaAction": {
        "functionArn": "$alarmNotificationFunctionArn"
      }
    },
    "smsConfigurations": [
      { 'recipients': [
        { 'ssoIdentity': {
          "identityStoreId": "$sitewise.assetModel.\"assetModelId\".\"identityStoreIdAttributeId\".propertyValue.value",
          "userId": "$sitewise.assetModel.\"assetModelId\".\"userIdAttributeId\".propertyValue.value"
        }
      ]
    },
    "emailConfigurations": [
      { 'from': "$sitewise.assetModel.\"assetModelId\".\"fromAddressAttributeId\".propertyValue.value",
        "recipients": {
          "to": [
            { 'ssoIdentity': {
              "identityStoreId": "$sitewise.assetModel.\"assetModelId\".\"identityStoreIdAttributeId\".propertyValue.value",
              "userId": "$sitewise.assetModel.\"assetModelId\".\"userIdAttributeId\".propertyValue.value"
            }
          ]
        },
        "content": {
          "subject": "$sitewise.assetModel.\"assetModelId\".\"emailSubjectAttributeId\".propertyValue.value",
          "additionalMessage": "$sitewise.assetModel.\"assetModelId\".\"additionalMessageAttributeId\".propertyValue.value"
        }
      }
    ]
  ]
}
i. Run the following command to create the AWS IoT Events alarm model from the payload in alarm-model-payload.json.

```
aws iotevents create-alarm-model --cli-input-json file://alarm-model-payload.json
```

j. The operation returns a response that includes the ARN of the alarm model, `alarmModelArn`. Copy this ARN to set in the alarm definition on your asset model in the next step.

### Step 3: Enabling data flow between AWS IoT SiteWise and AWS IoT Events

After you create the required resources in AWS IoT SiteWise and AWS IoT Events, you can enable data flow between the resources to enable your alarm. In this section, you update the alarm definition in the asset model to use the alarm model that you created in the previous step.

#### To enable data flow between AWS IoT SiteWise and AWS IoT Events (CLI)

- Set the alarm model as the alarm's source in the asset model. Do the following:
  
  a. Run the following command to retrieve the existing asset model definition. Replace `asset-model-id` with the ID of the asset model.

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id
```

  The operation returns a response that contains the asset model's details.

  b. Create a file called `update-asset-model-payload.json` and copy the previous command's response into the file.

  c. Remove the following key-value pairs from the `update-asset-model-payload.json` file:

  - `assetModelId`
  - `assetModelArn`
  - `assetModelCreationDate`
  - `assetModelLastUpdateDate`
  - `assetModelStatus`

  d. Add the alarm source property (`AWS/ALARM_SOURCE`) to the alarm composite model that you defined earlier. Replace `alarmModelArn` with the ARN of the alarm model, which sets the value of the alarm source property.

```json
{
...
"assetModelCompositeModels": [
  ...
  {
    "name": "BoilerTemperatureHighAlarm",
    "type": "AWS/ALARM",
```
Defining external alarms

Defining an external alarm (console)

You can use the AWS IoT SiteWise console to define an external alarm on an existing asset model. To define an external alarm on a new asset model, create the asset model, and then complete these steps. For more information, see Creating asset models (p. 143).

To define an alarm on an asset model

1. Navigate to the AWS IoT SiteWise console.

Your asset model now defines an alarm that detects in AWS IoT Events. The alarm monitors the target property in all assets based on this asset model. You can configure the alarm on each asset to customize properties such as the threshold or AWS SSO recipient for each asset. For more information, see Configuring alarms on assets (p. 247).

Defining external alarms

External alarms contain the state of an alarm that you detect outside of AWS IoT SiteWise.

Your asset model now defines an alarm that detects in AWS IoT Events. The alarm monitors the target property in all assets based on this asset model. You can configure the alarm on each asset to customize properties such as the threshold or AWS SSO recipient for each asset. For more information, see Configuring alarms on assets (p. 247).

Defining external alarms

External alarms contain the state of an alarm that you detect outside of AWS IoT SiteWise.

Defining an external alarm (console)

You can use the AWS IoT SiteWise console to define an external alarm on an existing asset model. To define an external alarm on a new asset model, create the asset model, and then complete these steps. For more information, see Creating asset models (p. 143).

To define an alarm on an asset model

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model for which to define an alarm.
4. Choose the **Alarm definitions** tab.
5. Choose **Add alarm**.
6. In **Alarm type options**, choose **External alarm**.
7. Enter a name for your alarm.
8. (Optional) Enter a description for your alarm.
9. Choose **Add alarm**.

### Defining an external alarm (CLI)

You can use the AWS CLI to define an external alarm on a new or existing asset model.

To add an external alarm to an asset model, you add an alarm composite model to the asset model. An external alarm composite model specifies the `EXTERNAL` type and doesn’t specify an alarm source property. The following example composite alarm defines an external temperature alarm.

```
{
  ...
  "assetModelCompositeModels": [
    {
      "name": "BoilerTemperatureHighAlarm",
      "type": "AWS/ALARM",
      "properties": [
        {
          "name": "AWS/ALARM_TYPE",
          "dataType": "STRING",
          "type": {
            "attribute": {
              "defaultValue": "EXTERNAL"
            }
          }
        },
        {
          "name": "AWS/ALARM_STATE",
          "dataType": "STRUCT",
          "dataTypeSpec": "AWS/ALARM_STATE",
          "type": {
            "measurement": {
            }
          }
        }
      ]
    }
  ]
}
```

For more information about how to add a composite model to a new or existing asset model, see the following:

- [Creating an asset model (CLI) (p. 144)](#)
- [Updating an asset model (CLI) (p. 213)](#)

After you define the external alarm, you can ingest alarm state to assets based on the asset model. For more information, see [Ingesting external alarm state (p. 254)](#).
Configuring alarms on assets

After you define an AWS IoT Events alarm on an asset model, you can configure the alarm on each asset based on the asset model. You can edit the threshold value and the notification settings for the alarm. Each of these values is an attribute on the asset, so you can update the default value of the attribute to configure these values.

**Note**
You can configure these values for AWS IoT Events alarms, but not external alarms.

**Topics**
- Configuring threshold values (p. 247)
- Configuring notification settings (p. 248)

**Configuring threshold values**

You can use the AWS IoT SiteWise console or API to configure the threshold value of an alarm for an asset.

**Topics**
- Configuring a threshold value (console) (p. 247)
- Configuring a threshold value (CLI) (p. 248)

**Configuring a threshold value (console)**

You can use the AWS IoT SiteWise console to update the value of the attribute that specifies the threshold value of an alarm.

**To update an alarm's threshold value (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset for which you want to update an alarm threshold value.

**Tip**
You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose **Edit**.
5. Find the attribute that the alarm uses for its threshold value, and then enter its new value.
6. Choose **Save**.

### Configuring a threshold value (CLI)

You can use the AWS Command Line Interface (AWS CLI) to update the value of the attribute that specifies the threshold value of an alarm. For more information, see [Updating an attribute value (CLI) (p. 205)](#).

### Configuring notification settings

You can use the AWS IoT SiteWise console or API to configure the alarm notification settings for an asset.

**Topics**
- Configuring notification settings (console) (p. 248)
- Configuring notification settings (CLI) (p. 249)

### Configuring notification settings (console)

You can use the AWS IoT SiteWise console to update the value of the attributes that specify the notification settings for an alarm.

**To update an alarm's notification settings (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**.
3. Choose the asset to which you want to update the alarm threshold value.

   **Tip**
   You can choose the arrow icon to expand an asset hierarchy to find your asset.
Responding to alarms

4. Choose **Edit**.
5. Find the attribute that the alarm uses for the notification setting that you want to change, and then enter its new value.
6. Choose **Save**.

**Configuring notification settings (CLI)**

You can use the AWS Command Line Interface (AWS CLI) to update the value of the attributes that specify the notification settings for an alarm. For more information, see [Updating an attribute value (CLI)](p. 205).

**Responding to alarms**

When an AWS IoT Events alarm changes state, you can do the following to respond to the alarm:

- Acknowledge an alarm to indicate that you are handling the issue.
- Snooze an alarm to disable it temporarily.
- Disable an alarm to disable it permanently until you enable it again.
- Enable a disabled alarm to detect alarm state.
- Reset an alarm to clear its state and latest value.

You can use the AWS IoT SiteWise console or the AWS IoT Events API to respond to an alarm.

**Note**

You can respond to AWS IoT Events alarms, but not external alarms.

**Topics**

- Responding to an alarm (console) (p. 250)
- Responding to an alarm (API) (p. 253)
Responding to an alarm (console)

You can use the AWS IoT SiteWise console to acknowledge, snooze, disable, or enable an alarm.

Topics
- Acknowledge an alarm (console) (p. 250)
- Snooze an alarm (console) (p. 250)
- Disable an alarm (console) (p. 251)
- Enable an alarm (console) (p. 252)
- Reset an alarm (console) (p. 253)

Acknowledge an alarm (console)

You can acknowledge an alarm to indicate that you’re handling the issue.

Note
You must enable the acknowledge flow on the alarm so that you can acknowledge the alarm. This option is enabled by default if you define the alarm from the AWS IoT SiteWise console.

To acknowledge an alarm (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to for which you want to acknowledge an alarm.
   Tip
   You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose the Alarms tab.
5. Select the alarm to acknowledge, and then choose Actions to open the response action menu.
6. Choose Acknowledge. The alarm’s state changes to Acknowledged.

Snooze an alarm (console)

You can snooze an alarm to disable it temporarily. Specify the duration for which to snooze the alarm.
To snooze an alarm (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to for which you want to snooze an alarm.
   
   **Tip**
   You can choose the arrow icon to expand an asset hierarchy to find your asset.

4. Choose the Alarms tab.
5. Select the alarm to snooze, and then choose Actions to open the response action menu.
6. Choose Snooze. A model opens where you specify the duration to snooze.
7. Choose the Snooze length or enter a Custom snooze length.
8. Choose Save. The alarm's state changes to Snoozed.

Disable an alarm (console)

You can disable an alarm so that it doesn't detect anymore. After you disable the alarm, you must enable it again if you want the alarm to detect.

To disable an alarm (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset to for which you want to disable an alarm.
   
   **Tip**
   You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose the **Alarms** tab.
5. Select the alarm to disable, and then choose **Actions** to open the response action menu.
6. Choose **Disable**. The alarm’s state changes to **Disabled**.

### Enable an alarm (console)

You can enable an alarm to detect again after you disable or snooze it.

**To enable an alarm (console)**

1. Navigate to the **AWS IoT SiteWise** console.
2. In the navigation pane, choose **Assets**.
3. Choose the asset to for which you want to enable an alarm.

**Tip**
You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose the **Alarms** tab.
5. Select the alarm to enable, and then choose **Actions** to open the response action menu.
6. Choose **Enable**. The alarm's state changes to **Normal**.

**Reset an alarm (console)**

You can reset an alarm to clear its state and latest value.

**To reset an alarm (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**.
3. Choose the asset to for which you want to reset an alarm.

    **Tip**
    You can choose the arrow icon to expand an asset hierarchy to find your asset.

4. Choose the **Alarms** tab.
5. Select the alarm to enable, and then choose **Actions** to open the response action menu.
6. Choose **Reset**. The alarm's state changes to **Normal**.

**Responding to an alarm (API)**

You can use the AWS IoT Events API to acknowledge, snooze, disable, enable, or reset an alarm. For more information, see the following operations in the *AWS IoT Events API Reference*:

- BatchAcknowledgeAlarm
- BatchSnoozeAlarm
- BatchDisableAlarm
- BatchEnableAlarm
- BatchResetAlarm

For more information, see *Responding to alarms* in the *AWS IoT Events Developer Guide*. 
Ingesting external alarm state

External alarms are alarms that you evaluate outside of AWS IoT SiteWise. You can use external alarms when you have a data source that reports alarm state that you want to ingest to AWS IoT SiteWise.

Alarm state properties require a specific format for alarm state data values. Each data value must be a JSON object serialized to a string. Then, you ingest the serialized string as a string value. For more information, see Alarm state properties (p. 221).

**Example** Example alarm state data value (not serialized)

```
{
  "stateName": "Active"
}
```

**Example** Example alarm state data value (serialized)

```
{"stateName":"Active"}
```

**Note**

If your data source can't report data in this format, or you can't convert your data to this format before you ingest it, you might choose not to use an alarm property. Instead, you can ingest the data as a measurement property with the string data type, for example. For more information, see Defining data streams from equipment (measurements) (p. 152) and Ingesting data to AWS IoT SiteWise (p. 69).

**Topics**

- Mapping external alarm state streams (p. 254)
- Ingesting alarm state data (p. 255)

**Mapping external alarm state streams**

You can define property aliases to map your data streams to your alarm state properties. This helps you easily identify an alarm state property when you ingest or retrieve data. For more information about property aliases, see Mapping industrial data streams to asset properties (p. 202).

**Topics**

- Mapping external alarm state streams (console) (p. 254)
- Mapping external alarm state streams (CLI) (p. 255)

**Mapping external alarm state streams (console)**

You can use the AWS IoT SiteWise console to set an alias for an alarm state property.

**To set a property alias for an alarm state property (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**.
3. Choose the asset for which you want to set a property alias.

**Tip**

You can choose the arrow icon to expand an asset hierarchy to find your asset.
4. Choose Alarms.
5. Select the external alarm for which you want to set a property alias.
6. Choose View.
7. In the Alarm state details pane, choose Edit.
8. Enter the property alias.
9. Choose Update.

**Mapping external alarm state streams (CLI)**

You can use the AWS Command Line Interface (AWS CLI) to set an alias for an alarm state property.

You must know your asset's `assetId` and property's `propertyId` to complete this procedure. If you created an asset but don't know its `assetId`, use the `ListAssets` operation to view all of your assets for a specific model. Then, use the `DescribeAsset` operation to view your asset's properties including property IDs.

**Note**

The `DescribeAsset` response includes the list of composite asset models for the asset. Each alarm is a composite model. To find the `propertyId`, find the composite model for the alarm, and then find the `AWS/ALARM_STATE` property in that composite model.

For more information about how to set the property alias, see Setting a property alias (CLI) (p. 203).

**Ingesting alarm state data**

Alarm state properties expect alarm state as a serialized JSON string. To ingest alarm state to an external alarm in AWS IoT SiteWise, you ingest this serialized string as a timestamped string value. The following example demonstrates a state data value for an active alarm.

```
{"stateName":"Active"}
```

To identify an alarm state property, you can specify one of the following:

- The `assetId` and `propertyId` of the alarm property that you're sending data to.
• The `propertyAlias`, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature/high`). To use this option, you must first set your alarm property’s alias. To learn how to set property aliases for alarm state properties, see Mapping external alarm state streams (p. 254).

The following example BatchPutAssetPropertyValue API payload demonstrates how to format the state of an external alarm. This external alarm reports when a wind turbine's rotations per minute (RPM) reading is too high.

**Example BatchPutAssetPropertyValue payload for alarm state data**

```json
{
  "entries": [
    {
      "entryId": "unique entry ID",
      "propertyAlias": "/company/windfarm/3/turbine/7/temperature/high",
      "propertyValues": [
        {
          "value": {
            "stringValue": "{"stateName":"Active"}"
          },
          "timestamp": {
            "timeInSeconds": 1607550262
          }
        }
      ]
    }
  ]
}
```

For more information about how to use the BatchPutAssetPropertyValue API to ingest data, see Ingesting data using the AWS IoT SiteWise API (p. 77).

For more information about other ways to ingest data, see Ingesting data to AWS IoT SiteWise (p. 69).
Monitoring data with AWS IoT SiteWise Monitor

You can use AWS IoT SiteWise to monitor the data from your processes, devices, and equipment by creating SiteWise Monitor web portals. SiteWise Monitor is a feature of AWS IoT SiteWise that you can use to create portals in the form of a managed web application. You can then use these portals to view and share your operational data. You can create projects with dashboards to visualize data from your processes, devices, and equipment that are connected to AWS IoT.

Domain experts, such as process engineers, can use these portals to quickly get insights into their operational data to understand device and equipment behavior.

The following is an example dashboard that displays data for a wind farm.

Because AWS IoT SiteWise captures data over time, you can use SiteWise Monitor to view operational data over time, or the last reported values at specific points in time. This lets you uncover insights that might otherwise be difficult to find.

SiteWise Monitor roles

Four roles interact with SiteWise Monitor:
AWS SiteWise Monitor roles

AWS administrator

The AWS administrator uses the AWS IoT SiteWise console to create portals. The AWS administrator can also assign portal administrators and add portal users. Portal administrators later assign portal users to projects as owners or viewers. The AWS administrator works exclusively in the AWS console.

Portal administrator

Each SiteWise Monitor portal has one or more portal administrators. Portal administrators use the portal to create projects that contain collections of assets and dashboards. The portal administrator then assigns assets and owners to each project. By controlling access to the project, portal administrators specify which assets that project owners and viewers can see.

Project owner

Each SiteWise Monitor project has owners. Project owners create visualizations in the form of dashboards to represent operational data in a consistent manner. When dashboards are ready to share, the project owner can invite viewers to the project. Project owners can also assign other owners to the project. Project owners can configure thresholds and notification settings for alarms.

Project viewer

Each SiteWise Monitor project has viewers. Project viewers can connect to the portal to view the dashboards that project owners created. In each dashboard, project viewers can adjust the time range to better understand operational data. Project viewers can only view dashboards in the projects to which they have access. Project viewers can acknowledge and snooze alarms.

Depending on your organization, the same person might perform multiple roles.

The following image illustrates how these four roles interact in the SiteWise Monitor portal.
You can manage who has access to your data by using AWS Single Sign-On or IAM. Your data users can sign in to SiteWise Monitor from a desktop or mobile browser using their AWS SSO or IAM credentials.

**SAML federation**

AWS SSO and IAM support identity federation with SAML (Security Assertion Markup Language) 2.0. SAML 2.0 is an open standard that many external identity providers (IdPs) use to authenticate users and pass their identity and security information to service providers (SPs). SPs are typically applications or services. SAML federation enables your SiteWise Monitor portal administrators and users to sign in to their assigned portals with external credentials, such as their corporate usernames and passwords.

You can configure AWS SSO and IAM to use SAML-based federation for access to your SiteWise Monitor portals.

**AWS SSO**

Your portal administrators and users can sign in to the AWS SSO user portal with their corporate usernames and passwords. They can then navigate to their assigned SiteWise Monitor portals. AWS
SSO uses certificates to set up a SAML trust relationship between your identity provider and AWS. For more information, see SCIM profile and SAML 2.0 implementation in the AWS Single Sign-On User Guide.

IAM

Your portal administrators and users can request temporary security credentials to access their assigned SiteWise Monitor portals. You create a SAML identity provider identity in IAM to set up a trust relationship between your identity provider and AWS. For more information, see Using SAML-based federation for API access to AWS, in the IAM User Guide.

Your portal administrators and users can sign in to your company's portal and select the option to go to the AWS Management console. They can then navigate to their assigned SiteWise Monitor portals. Your company's portal handles the exchange of trust between your identity provider and AWS. For more information, see Enabling SAML 2.0 federated users to access the AWS Management Console in the IAM User Guide.

SiteWise Monitor concepts

To use SiteWise Monitor, you should be familiar with the following concepts:

Portal

An SiteWise Monitor portal is a web application that you can use to visualize and share your AWS IoT SiteWise data. A portal has one or more administrators and contains zero or more projects.

Project

Each SiteWise Monitor portal contains a set of projects. Each project has a subset of your AWS IoT SiteWise assets associated with it. Project owners create one or more dashboards to provide a consistent way to view the data associated with those assets. Project owners can invite viewers to the project to allow them to view the assets and dashboards in the project. The project is the basic unit of sharing within SiteWise Monitor. Project owners can invite users who were given access to the portal by the AWS administrator. A user must have access to a portal before a project in that portal can be shared with that user.

Asset

When data is ingested into AWS IoT SiteWise from your industrial equipment, your devices, equipment, and processes are each represented as assets. Each asset has properties and alarms associated with it. The portal administrator assigns sets of assets to each project.

Property

Properties are time series data associated with assets. For example, a piece of equipment might have a serial number, a location, a make and model, and an install date. It might also have time series values for availability, performance, quality, temperature, pressure, and so on.

Alarm

Alarms monitor properties to identify when equipment is outside of its operating range. Each alarm defines a threshold and a property to monitor. When the property exceeds the threshold, the alarm becomes active and indicates that you or someone on your team should address the issue. Project owners can customize the thresholds and notification settings for alarms. Project viewers can acknowledge and snooze alarms, and they can leave a message with details about the alarm or the action that they took to address it.

Dashboard

Each project contains a set of dashboards. Dashboards provide a set of visualizations for the values of a set of assets. Project owners create the dashboards and the visualizations that it contains. When
a project owner is ready to share the set of dashboards, the owner can invite viewers to the project, which gives them access to all dashboards in the project. If you want a different set of viewers for different dashboards, you must divide the dashboards between projects. When viewers look at dashboards, they can adjust the time range.

Visualizations

In each dashboard, project owners decide how to display the properties and alarms of the assets associated with the project. Availability might be represented as a line chart, while other values might be displayed as bar charts or key performance indicators (KPIs). Alarms are best displayed as status grids and status timelines. Project owners customize each visualization to provide the best understanding of the data for that asset.

Getting started with AWS IoT SiteWise Monitor

If you're the AWS administrator for your organization, you create portals from the AWS IoT SiteWise console. Complete the following steps to create a portal so that members of your organization can view your AWS IoT SiteWise data:

1. Configure and create a portal
2. Add portal administrators and send invitation emails
3. Add portal users

After you create a portal, the portal administrator can view your AWS IoT SiteWise assets and assign them to projects in the portal. Project owners can then create dashboards to visualize the properties of the assets that help project viewers understand how your devices, processes, and equipment are performing.

You can follow a tutorial that walks through the steps required to set up a portal with a project, dashboards, and multiple users for a specific scenario using wind farm data. For more information, see Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 38).

Creating a portal

You create a SiteWise Monitor portal in the AWS IoT SiteWise console.

To create a portal

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose Monitor, Getting started.
3. Choose Create Portal.

Next, you must provide some basic information to configure your portal.

**Configuring your portal**

Your users use portals to view your data. You can customize a portal's name, description, branding, user authentication, support contact email, and permissions.
Portal configuration

Each web portal provides enterprise users with access to your IoT SiteWise assets. Learn more

Portal details

Portal name
Choose a portal name to identify the web portal to your users. Company name is recommended.

example-factory-1

*Note: should be 1-128 characters and only contain A-Z, a-z, 0-9 _, and -*

Description - optional
Create a description of your portal.

Example Corp Factory #1 in Renton, WA

Description should contain a maximum of 2048 characters.

Portal branding

You can provide your logo image to display your brand in this web portal.

Logo image
Upload a square, high-resolution .png file. The image is displayed on a dark background.

Choose file

The file size must be less than 1 MB.

User authentication

Your users can sign in to this portal with their AWS Single Sign-On (AWS SSO) or AWS Identity and Access Management (IAM) credentials. If you choose AWS SSO, you must enable the service for your AWS account.

You haven't enabled AWS SSO in your account yet. When you create your first portal user, this automatically enables AWS SSO in your AWS account.

Create user

- AWS SSO
  - Your users can sign in to the portal with their corporate usernames and passwords.
- IAM
  - Your users can sign in to the portal with their IAM credentials.

Support contact email

You can provide an email address for cases where there’s a problem or issue with this portal and your users need to contact support to resolve.

Email

support@example.com

Tags

This resource doesn’t have any tags.

Add tag

You can add up to 50 more tags.

Permissions

SiteWise Monitor assumes this role to give permissions to your federated users to access AWS IoT SiteWise resources. Learn more

- Create and use a new service role
- Use an existing service role

View details
To configure a portal

1. Enter a name for your portal.
2. (Optional) Enter a description for your portal. If you have multiple portals, use meaningful descriptions to help you keep track of what each portal contains.
3. (Optional) Upload an image to display your brand in the portal. Choose a square, PNG image. If you upload a non-square image, the portal scales the image down to a square.
4. Choose one of the following options:
   - Choose **AWS SSO** if your portal users sign in to this portal with their corporate user names and passwords.
     
     If you haven't enabled AWS SSO in your account, do the following:
     
     a. Choose **Create user**.
     b. On the **Create user** page, to create the first portal, enter the user's email address, first name, and last name, and then choose **Create user**.

     ![Create user](image)

     **Note**

     - AWS automatically enables AWS SSO in your account when you create the first portal user.
     - You can configure AWS SSO in only one Region at a time. SiteWise Monitor connects to the Region that you configured for AWS SSO. This means that you use one Region for AWS SSO access, but you can create portals in any Region.

     - Choose **IAM** if your portal users sign in to this portal with their IAM credentials.

     **Important**

     IAM users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.

5. Enter an email address that portal users can contact when they have an issue with the portal and need help to resolve it.
6. (Optional) Add tags for your portal. For more information, see **Tagging your AWS IoT SiteWise resources** (p. 369).
7. Choose one of the following options:

   - Choose **Create and use a new service role**. By default, SiteWise Monitor automatically creates a service role for each portal. This role allows your portal users to access your AWS IoT SiteWise resources. For more information, see **Using service roles for AWS IoT SiteWise Monitor** (p. 341).
• Choose **Use an existing service role**, and then choose the target role.

8. Choose **Next**

9. (Optional) Enable alarms for your portal. For more information, see Enabling additional features for portals (p. 271).

10. Choose **Create**. AWS IoT SiteWise will create your portal.

    **Note**
    If you close the console, you can finish the setup process by adding administrators and users. For more information, see Adding or removing portal administrators (p. 275). If you don't want to keep this portal, delete it so it doesn't use resources. For more information, see Deleting a portal (p. 279).

A message appears when your portal is created.

Next, you must invite one or more portal administrators to the portal. So far, you created a portal but no one can access it.

**Inviting administrators**

To get started in your new portal, you must assign a portal administrator. The portal administrator creates projects, chooses project owners, and assigns assets to projects. Portal administrators can see all of your AWS IoT SiteWise assets.

Based on the user authentication service, choose one of the following options:

**AWS SSO**

If you're using SiteWise Monitor for the first time, you can choose the user that you created earlier to be the portal administrator. If you want to add another user as a portal administrator, you can create an AWS SSO user from this page. Alternatively, you can connect an external identity provider to AWS SSO. For more information, see the AWS Single Sign-On User Guide.

**To invite administrators**

1. Select the check boxes for the users that you want as your portal administrators. This adds the users to the **Portal administrators** list.

    **Note**
    If you use AWS SSO as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an AWS SSO user. AWS SSO sends the new user an email for them to set their password. You can then assign the user to the portal as an administrator. For more information, see Manage identities in AWS SSO.

2. (Optional) Choose **Send invite to selected users**. Your email client opens, and an invitation is populated in the message body, as shown in the following example.
You can customize the email before you send it to your portal administrators. You can also send the email to your portal administrators later. If you're trying SiteWise Monitor for the first time and adding your new AWS SSO or IAM user or role as the portal administrator, you don't need to email yourself.

3. If you add a user that you don’t want as an administrator, clear the check box for that user.
4. When you’re finished inviting portal administrators, choose Next.

I AM

You can choose an IAM user or role to be the portal administrator. If you want to add another IAM user or role as a portal administrator, you can create an user or role in the IAM console. For more information, see Creating an IAM user in your AWS account and Creating IAM roles in the IAM User Guide.

To invite administrators

1. Do the following:
   - Choose IAM users to add an IAM user as your portal administrator.
   - Choose IAM roles to add an IAM role as your portal administrator.
2. Select the check boxes for the users or roles that you want as your portal administrators. This adds the users or roles to the Portal administrators list.
3. If you add a user or role that you don’t want as an administrator, clear the check box for that user or role.
4. When you’re finished inviting portal administrators, choose **Next**.

**Important**
IAM users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.

**Note**
If you use AWS SSO as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an AWS SSO user. AWS SSO sends the new user an email for them to set their password. You can then assign the user to the portal as an administrator. For more information, see [Manage identities in AWS SSO](#).
You can change the list of portal administrators later. For more information, see Adding or removing portal administrators (p. 275).

**Note**

Because only a portal administrator can create projects and assign assets to them, you should specify at least one portal administrator.

As the last step, you add users who can access your new portal.

**Adding portal users**

You control which users have access to your portals. In each portal, the portal administrators create one or more projects and assign portal users as owners or viewers for each project. Each project owner can invite additional portal users to own or view the project.

Based on the user authentication service, choose one of the following options:

**AWS SSO**

If you want to add a user to the Users list, complete the following steps.
To add portal users

1. Choose users from the Users list to add to the portal. This adds the users to the Portal users list. If you're using SiteWise Monitor for the first time, you don't need to add your portal administrator as a portal user.

   **Note**
   If you use AWS SSO as your identity store, and you're signed in to your AWS Organizations management account, you can choose Create user to create an AWS SSO user. AWS SSO sends the new user an email for them to set their password. You can then assign the user to the portal as a user. For more information, see Manage identities in AWS SSO.

2. If you add a user that you don't want to have access to the portal, clear the check box for that user.

3. When you're finished selecting users, choose Assign users.

IAM

If you see the user or role that you want to add in the IAM users or IAM roles list, complete the following steps.

To add portal users

1. Do the following options:
   - Choose IAM users to add an IAM user as a portal user.
   - Choose IAM roles to add an IAM role as a portal user.

   If you're using SiteWise Monitor for the first time, you don't need to add your portal administrator as a portal user.

2. Select the check boxes for the users or roles that you want as portal users. This adds the users or roles to the Portal users list.

3. If you add a user that you don't want to have access to the portal, clear the check box for that user.

4. When you're finished selecting users, choose Assign users.
Important
IAM users or roles must have the iotsitewise:DescribePortal permission to sign in to the portal.

Assign users
Select the users you want to be able to access and view this portal. Portal administrators will send invitations to these users at a later date. Learn more

IAM users (1)
Manage users in IAM console

<table>
<thead>
<tr>
<th>Name</th>
<th>Date created</th>
</tr>
</thead>
<tbody>
<tr>
<td>raspberryPi-testing</td>
<td>11-08-2019</td>
</tr>
</tbody>
</table>

Portal users (1)

Assign users
Congratulations! You successfully created a portal, assigned portal administrators, and assigned users who can use that portal when invited to do so. Your portal administrators can now create projects and add assets to those projects. Then, your project owners can create dashboards to visualize the data for each project’s assets.

You can change the list of portal users later. For more information, see Adding or removing portal users (p. 277).

If you need to make changes to the portal, see Administering your SiteWise Monitor portals (p. 273).

To get started in the portal, see Getting started in the SiteWise Monitor Application Guide.

Enabling additional features for portals

You can enable the following features for your portals.

**Topics**
- Enabling alarms for your portals (p. 271)
- Enabling your portal at the edge (p. 273)

**Enabling alarms for your portals**

You can enable the alarms feature supported by AWS IoT Events for your portals so that portal administrators can create, edit, and delete AWS IoT Events alarm models in your SiteWise Monitor portals. Project owners can configure alarms. Project viewers can view alarm details. This section explains how you can use the AWS IoT SiteWise console to enable the alarms feature for your portals.

**Important**
- You can’t create external alarms in your portals.
- If you want to send alarm notifications, you must choose AWS SSO for the user authentication service.
- The alarm notifications feature isn’t available in the China (Beijing) Region.

When you configure and create a portal, you can enable alarms and alarm notifications in Step 2 Additional features. Based on the user authentication service, choose one of the following options:
To enable alarms for a portal

1. (Optional) Choose **Enable alarms**.
   - For **AWS IoT SiteWise access role**, use an existing role or create a role with the required permissions. This role requires the `iotevents:BatchPutMessage` permission and a trust relationship that allows `iot.amazonaws.com` and `iotevents.amazonaws.com` to assume the role.

2. (Optional) Choose **Enable alarm notifications**.
   a. For **Sender**, choose the sender.
      - **Important**
        You must verify the sender email address in Amazon SES. For more information, see [Verifying email addresses in Amazon SES](https://docs.aws.amazon.com/AmazonSES/latest/DeveloperGuide/email-addresses.html), in the *Amazon Simple Email Service Developer Guide*.
   b. For **AWS Lambda role**, use an existing role or create a role with the required permissions. This role requires the `lambda:InvokeFunction` and `sso-directory:DescribeUser` permissions and a trust relationship that allows `iotevents.amazonaws.com` and `lambda.amazonaws.com` to assume the role.
   c. For **AWS Lambda functions**, choose an existing Lambda function or create a function that manages alarm notifications. For more information, see [Managing alarm notifications](https://docs.aws.amazon.com/iotevents/latest/developerguide/alarms.html) in the *AWS IoT Events Developer Guide*. 

---

**Additional features - optional**

<table>
<thead>
<tr>
<th>Alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your portal users can create alarms in the portal to monitor equipment or processes. They can also get notified when the equipment or processes perform outside specified range.</td>
</tr>
</tbody>
</table>

- **Enable alarms**
  - If enabled, your portal users can define AWS IoT Events alarms in SiteWise Monitor.
  - **AWS IoT SiteWise access role**
    - Choose an IAM role that allows AWS IoT Events to send data to AWS IoT SiteWise. To edit the role, go to the IAM console.
    - Create a role from an AWS managed template
      - Use an existing role
  - **Enable alarm notifications**
    - If enabled, alarms can send email or SMS notifications.
    - **Sender**
      - Specify the email address that sends alarm notifications. To edit or add a sender, go to the Amazon SES console.
    - **AWS Lambda role**
      - Choose an IAM role that allows AWS Lambda to send data to Amazon SES and Amazon SES. To edit the role, go to the IAM console.
      - Create a role from an AWS managed template
        - Use an existing role
    - **AWS Lambda function**
      - Choose an AWS Lambda function to manage alarm notifications. To edit the function, go to the AWS Lambda console.
IAM

To enable alarms for a portal

- (Optional) Choose Enable alarms.
  - For **AWS IoT SiteWise access role**, use an existing role or create a role with the required permissions. This role requires the `iotevents:BatchPutMessage` permission and a trust relationship that allows `iot.amazonaws.com` and `iotevents.amazonaws.com` to assume the role.

For more information about alarms in SiteWise Monitor, see Monitoring with alarms in the *AWS IoT SiteWise Application Guide*.

Enabling your portal at the edge

After you enable your portal at the edge, this portal is available on all AWS IoT SiteWise gateways with the data processing pack enabled in your account.

To enable the portal at the edge

1. In the **Edge configuration** section, turn on **Enable this portal at the edge**.
2. Choose Create.

Administering your SiteWise Monitor portals

You might need to update portal details, change administrators, or add users to your portals. This section explains how you can complete these basic administrative tasks for your SiteWise Monitor portals.

1. Sign in to the **AWS IoT SiteWise console**.
2. In the navigation pane, choose **Monitor, Portals**.
Changing a portal's name, description, branding, support email, and permissions

3. Choose a portal, and then choose View details (or choose the portal's Name).

4. You can perform any of the following administrative tasks:
   - Changing a portal's name, description, branding, support email, and permissions (p. 274)
   - Adding or removing portal administrators (p. 275)
   - Sending email invitations to portal administrators (p. 277)
   - Adding or removing portal users (p. 277)
   - Deleting a portal (p. 279)

For information about how to create a portal, see Getting started with AWS IoT SiteWise Monitor (p. 261).

Changing a portal's name, description, branding, support email, and permissions

You can change a portal's name, description, branding, support email, and permissions.

1. On the portal details page, in the Portal details section, choose Edit.
2. Update the Name, Description, Portal branding, Support contact email, or Permissions.
3. When you’re finished, choose Save.

Adding or removing portal administrators

In a few steps, you can add or remove users as administrators for a portal. Based on the user authentication service, choose one of the following options.

AWS SSO

To add portal administrators

1. On the portal details page, in the Portal administrators section, choose Assign administrators.
2. On the Assign administrators page, select the check boxes for the users to add to the portal as administrators.
   
   **Note**
   
   If you use AWS SSO as your identity store, and you're signed in to your AWS Organizations management account, you can choose Create user to create an AWS SSO user. AWS SSO sends the new user an email for them to set their password. You can then assign the user to the portal as an administrator. For more information, see Manage identities in AWS SSO.
3. Choose Assign administrators.

To remove portal administrators

- On the portal details page, in the Portal administrators section, select the check box for each user to remove, and then choose Remove from portal.

   **Note**
   
   We recommend that you select at least one portal administrator.

IAM
To add portal administrators

1. On the portal details page, in the **Portal administrators** section, choose **Assign administrators**.
2. On the **Assign administrators** page, do the following:
   - Choose **IAM users**, if you want to add an IAM user as your portal administrator.
   - Choose **IAM roles**, if you want to add an IAM role as your portal administrator.
3. Select the check boxes for the users or roles that you want as your portal administrators. This adds the users or roles to the **Portal administrators** list.
4. Choose **Assign administrators**.

   **Important**
   IAM users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.

To remove portal administrators

- On the portal details page, in the **Portal administrators** section, select the check box for each user to remove, and then choose **Remove from portal**.

   **Note**
   Leaving a portal without a portal administrator is not recommended.
Sending email invitations to portal administrators

You can send email invitations to portal administrators.

1. On the portal details page, in the **Portal administrators** section, select the check boxes for the portal administrators.

![Portal administrators section](Image)

2. Choose **Send invitations**. Your email client opens, and an invitation is populated in the message body, as shown in the following.

**Example**: The invitation email that is sent to AWS SSO users.

![Invitation email](Image)

You can customize the email before you send it to your portal administrators.

Adding or removing portal users

You choose which users have access to your portals. Portal users appear in the list of users within a SiteWise Monitor portal. From this list, portal administrators can add project owners, and project owners can add project viewers.

**Note**

Your portal administrators and portal users might contact you through a portal's support email if they need you to add or remove a user.

Based on the user authentication service, choose one of the following options.
AWS SSO

To add portal users
1. On the portal details page, in the **Portal users** section, choose **Assign users**.
2. On the **Assign users** page, select the check box for the users to add to the portal.
   
   **Note**
   If you use AWS SSO as your identity store, and you’re signed in to your AWS Organizations management account, you can choose **Create user** to create an AWS SSO user. AWS SSO sends the new user an email for them to set their password. You can then assign the user to the portal as a user. For more information, see Manage identities in AWS SSO.
3. Choose **Assign users**.

To remove portal users

- On the portal details page, in the **Portal users** section, select the check box for the users to remove from the portal, and then choose **Remove from portal**.

IAM

To add portal users
1. On the portal details page, in the **Portal users** section, choose **Assign users**.
2. On the **Assign users** page, do the following:
   - Choose **IAM users** to add an IAM user as your portal user.
   - Choose **IAM roles** to add an IAM role as your portal user.
3. Select the check boxes for the users or roles that you want to add as your portal users. This adds the users or roles to the **Portal users** list.
4. Choose **Assign users**.
Deleting a portal

You might delete a portal if you created it for testing purposes or if you created a duplicate of a portal that already exists.

Note
You must first manually delete all dashboards and projects in a portal before you can delete a portal. For more information, see Deleting projects and Deleting dashboards in the SiteWise Monitor Application Guide.

1. On the portal details page, choose Delete.

Important
When you delete a portal, you lose all projects that the portal contains, and all dashboards in each project. This action can't be undone. Your asset data isn't affected.
2. In the **Delete portals** dialog box, choose **Remove admins and users**.

You must remove the administrators and users from a portal before you can delete it. If your portal doesn't have administrators or users, the button doesn't appear, and you can skip to the next step.

3. If you're sure that you want to delete the entire portal, enter **delete** in the field to confirm deletion.

4. Choose **Delete**.

---

**Creating dashboards (AWS Command Line Interface)**

When you define visualizations (or widgets) in dashboards using the AWS CLI, you must specify the following information in the `dashboardDefinition` JSON document. This definition is a parameter of the `CreateDashboard` and `UpdateDashboard` operations.
widgets

A list of widget definition structures that each contain the following information:

type

The type of widget. AWS IoT SiteWise provides the following widget types:

- `sc-line-chart` – A line chart. For more information, see Line charts in the AWS IoT SiteWise Monitor Application Guide.
- `sc-scatter-chart` – A scatter chart. For more information, see Scatter charts in the AWS IoT SiteWise Monitor Application Guide.
- `sc-bar-chart` – A bar chart. For more information, see Bar charts in the AWS IoT SiteWise Monitor Application Guide.
- `sc-status-grid` – A status widget that shows the latest value of asset properties as a grid. For more information, see Status widgets in the AWS IoT SiteWise Monitor Application Guide.
- `sc-status-timeline` – A status widget that shows the historical values of asset properties as a timeline. For more information, see Status widgets in the AWS IoT SiteWise Monitor Application Guide.
- `sc-kpi` – A key performance indicator (KPI) visualization. For more information, see KPI widgets in the AWS IoT SiteWise Monitor Application Guide.
- `sc-table` – A table widget. For more information, see Table widgets in the AWS IoT SiteWise Monitor Application Guide.

title

The title of the widget.

x

The horizontal position of the widget, starting from the left of the grid. This value refers to the widget's position in the dashboard's grid.

y

The vertical position of the widget, starting from the top of the grid. This value refers to the widget's position in the dashboard's grid.

width

The width of the widget, expressed in number of spaces on the dashboard's grid.

height

The height of the widget, expressed in number of spaces on the dashboard's grid.

metrics

A list of metric structures that each define a data stream for this widget. Each structure in the list must contain the following information:

label

A label to display for this metric.

type

The type of data source for this metric. AWS IoT SiteWise provides the following metric types:
• iotsitewise – The dashboard fetches data for an asset property in AWS IoT SiteWise. If you choose this option, you must define assetId and propertyId for this metric.

    assetId
    (Optional) The ID of an asset in AWS IoT SiteWise.

    This field is required if you choose iotsitewise for type in this metric.

    propertyId
    (Optional) The ID of an asset property in AWS IoT SiteWise.

    This field is required if you choose iotsitewise for type in this metric.

    analysis
    (Optional) A structure that defines the analysis, such as trend lines, to display for the widget. For more information, see Configuring trend lines in the AWS IoT SiteWise Monitor Application Guide. You can add one of each type of trend line per property in the widget. The analysis structure contains the following information:

    trends
    (Optional) A list of trend structures that each define a trend analysis for this widget. Each structure in the list contains the following information:

        type
        The type of trend line. Choose the following option:

        • linear-regression – Display a linear regression line. SiteWise Monitor uses the least squares method to calculate the linear regression.

    annotations
    (Optional) An annotations structure that defines thresholds for the widget. For more information, see Configuring thresholds in the AWS IoT SiteWise Monitor Application Guide. You can add up to six annotations per widget. The annotations structure contains the following information:

        y
        (Optional) A list of annotation structures that each define a horizontal threshold for this widget. Each structure in the list contains the following information:

            comparisonOperator
            The comparison operator for the threshold. Choose one of the following:

            • LT – Highlight properties that have at least one data point less than the value.
            • GT – Highlight properties that have at least one data point greater than the value.
            • LTE – Highlight properties that have at least one data point less than or equal to the value.
            • GTE – Highlight properties that have at least one data point greater than or equal to the value.
            • EQ – Highlight properties that have at least one data point equal to the value.

            value
            The threshold value to compare data points with the comparisonOperator.

            color
            (Optional) The 6-digit hexadecimal code of the threshold color. The visualization displays property legends in this color for properties with at least one data point that meets the threshold rule. Defaults to black (#000000).
Creating dashboards (CLI)

showValue

(Optional) Whether or not to show the value of the threshold in the margins of the widget. Defaults to true.

properties

(Optional) A flat dictionary of properties for the widget. The members of this structure are context-dependent. AWS IoT SiteWise provides the following widgets that use properties:

- Line charts (p. 281), scatter charts (p. 281), and bar charts (p. 281) have the following property:
  colorDataAcrossThresholds

  (Optional) Whether or not to change the color of the data that crosses the thresholds in this widget. When you enable this option, the data that crosses a threshold appears in the color that you choose. Defaults to true.

- Status grids (p. 281) have the following property:
  labels

  (Optional) A structure that defines the labels to display on the status grid. The labels structure contains the following information:

  showValue

  (Optional) Whether or not to display the unit and value for each asset property in this widget. Defaults to true.

Example Example dashboard definition

The following example defines a dashboard from a payload stored in a JSON file.

```
aws iotsitewise create-dashboard \
  --project-id a1b2c3d4-5678-90ab-cdef-eeeeeeEXAMPLE \
  --dashboard-name "Wind Farm Dashboard" \
  --dashboard-definition file://dashboard-definition.json
```

The following JSON example for `dashboard-definition.json` defines dashboard with the following visualization widgets:

- A line chart that visualizes total wind farm power in the upper left of the dashboard. This line chart includes a threshold that indicates when the wind farm outputs less power than its minimum expected output. This line chart also includes a linear regression trend line.
- A bar chart that visualizes wind speed for four turbines in the upper right of the dashboard.

**Note**

This example represents line and bar chart visualizations on a dashboard. This dashboard is similar to the example wind farm dashboard (p. 257).

```json
{
  "widgets": [
    {
      "type": "sc-line-chart",
      "title": "Total Average Power",
      "x": 0,
      "y": 0,
      "height": 3,
      "width": 3,
      "metrics": [
```


```
{
  "label": "Power",
  "type": "iotsitewise",
  "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
  "propertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
  "analysis": {
    "trends": [
      {
        "type": "linear-regression"
      }
    ]
  },
  "annotations": {
    "y": [
      {
        "comparisonOperator": "LT",
        "value": 20000,
        "color": 
"#D13212",
        "showValue": true
      }
    ]
  },
  "type": "sc-bar-chart",
  "title": "Wind Speed",
  "x": 3,
  "y": 3,
  "height": 3,
  "width": 3,
  "metrics": [
    {
      "label": "Turbine 1",
      "type": "iotsitewise",
      "assetId": "a1b2c3d4-5678-90ab-cdef-2a2a2EXAMPLE",
      "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
    },
    {
      "label": "Turbine 2",
      "type": "iotsitewise",
      "assetId": "a1b2c3d4-5678-90ab-cdef-2b2b2EXAMPLE",
      "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
    },
    {
      "label": "Turbine 3",
      "type": "iotsitewise",
      "assetId": "a1b2c3d4-5678-90ab-cdef-2c2c2EXAMPLE",
      "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
    },
    {
      "label": "Turbine 4",
      "type": "iotsitewise",
      "assetId": "a1b2c3d4-5678-90ab-cdef-2d2d2EXAMPLE",
      "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
    }
  ]
}
```
Querying asset property values and aggregates

You can use the AWS IoT SiteWise API operations to query your asset properties' current values, historical values, and aggregates over specific time intervals. You can use these features to gain quick insights or develop software solutions that integrate with the industrial data stored in your AWS IoT SiteWise assets.

You can also explore your asset data live in AWS IoT SiteWise Monitor. To learn how to configure SiteWise Monitor, see Monitoring data with AWS IoT SiteWise Monitor (p. 257).

The operations described in this section return property value objects that contain timestamp, quality, value (TQV) structures.

- The timestamp contains the current Unix epoch time in seconds with nanosecond offset.
- The quality contains one of the following strings that indicate the quality of the data point:
  - GOOD – The data isn't affected by any issues.
  - BAD – The data is affected by an issue such as sensor failure.
  - UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.
- The value contains one of the following fields, depending on the type of the property:
  - booleanValue
  - doubleValue
  - integerValue
  - stringValue

Topics
• Querying current asset property values (p. 285)
• Querying historical asset property values (p. 287)
• Querying asset property aggregates (p. 288)

Querying current asset property values

You can use the AWS IoT SiteWise console or API to get the current value of an asset property.

Topics
• Querying an asset property's current value (console) (p. 285)
• Querying an asset property's current value (CLI) (p. 286)

Querying an asset property's current value (console)

You can use the AWS IoT SiteWise console to view the current value of an asset property.

To get the current value of an asset property (console)
1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Assets.
3. Choose the asset with the property to query.  

**Tip**
You can choose the arrow icon to expand an asset hierarchy to find your asset.

4. Choose the tab for the type of property. For example, choose **Measurements** to view the current value of a measurement property.

5. Find the property to view. The current value appears in the **Latest value** column.

---

**Querying an asset property's current value (CLI)**

You can use the AWS Command Line Interface (AWS CLI) to query the current value of an asset property.

Use the **GetAssetPropertyValue** operation to query an asset property's current value.

To identify an asset property, you can specify one of the following:

- The **assetId** and **propertyId** of the asset property that you are sending data to.
- The **propertyAlias**, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see **Mapping industrial data streams to asset properties** (p. 202).

**To get the current value of an asset property (CLI)**

- Run the following command to get the current value of the asset property. Replace **asset-id** with the ID of the asset and **property-id** with the ID of the property.

```bash
aws iotsitewise get-asset-property-value
  --asset-id asset-id
  --property-id property-id
```

The operation returns a response that contains the current TQV of the property in the following format.
Querying historical asset property values

You can use the AWS IoT SiteWise API to get the value history for an asset property.

Use the `GetAssetPropertyValueHistory` operation to query the historical values of an asset property.

To identify an asset property, you can specify one of the following:

- The `assetId` and `propertyId` of the asset property that you are sending data to.
- The `propertyAlias`, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 202).

You can also pass any of the following parameters to refine your results:

- `startDate` – The exclusive start of the range from which to query historical data, expressed in seconds in Unix epoch time.
- `endDate` – The inclusive end of the range from which to query historical data, expressed in seconds in Unix epoch time.
- `maxResults` – The maximum number of results to return in one request. Defaults to 20 results.
- `nextToken` – A pagination token returned from a previous call of this operation.
- `timeOrdering` – The ordering to apply to the returned values: `ASCENDING` or `DESCENDING`.
- `qualities` – The quality to filter results by: `GOOD`, `BAD`, or `UNCERTAIN`.

To query the value history for an asset property (CLI)

1. Run the following command to get the value history for the asset property. This command queries the property's history over a specific 10 minute interval. Replace `asset-id` with the ID of the asset and `property-id` with the ID of the property. Replace the date parameters with the interval to query.

```bash
aws iotsitewise get-asset-property-value-history \
   --asset-id asset-id \n   --property-id property-id \n   --start-date 1575216000 \n   --end-date 1575216600
```
The operation returns a response that contains the historical TQVs of the property in the following format.

```json
{
    "assetPropertyValueHistory": [
        {
            "value": {
                "booleanValue": Boolean,
                "doubleValue": Number,
                "integerValue": Number,
                "stringValue": String
            },
            "timestamp": {
                "timeInSeconds": Number,
                "offsetInNanos": Number
            },
            "quality": "String"
        }
    ],
    "nextToken": "String"
}
```

2. If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to the `GetAssetPropertyValueHistory` operation.

## Querying asset property aggregates

AWS IoT SiteWise automatically computes aggregated asset property values, which are a set of basic metrics calculated over multiple time intervals. AWS IoT SiteWise computes the following aggregates every minute, hour, and day for your asset properties:

- **average** – The average (mean) of a property's values over a time interval.
- **count** – The number of data points for a property over a time interval.
- **maximum** – The maximum of a property's values over a time interval.
- **minimum** – The minimum of a property's values over a time interval.
- **standard deviation** – The standard deviation of a property's values over a time interval.
- **sum** – The sum of a property's values over a time interval.

For non-numeric properties, such as strings and Booleans, AWS IoT SiteWise computes only the count aggregate.

You can also compute custom metrics for your asset data. Metric properties let you define aggregations specific to your operation. Metric properties offer additional aggregation functions and time intervals that aren't precomputed for the AWS IoT SiteWise API. For more information, see Aggregating data from properties and other assets (metrics) (p. 157).

You can use the AWS IoT SiteWise API to get aggregates for an asset property.

Use the `GetAssetPropertyAggregates` operation to query aggregates of an asset property.

To identify an asset property, you can specify one of the following:

- The `assetId` and `propertyId` of the asset property that you are sending data to.
- The `propertyAlias`, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 202).
You must also pass the following required parameters:

- **aggregateTypes** – The list of aggregates to retrieve. You can specify any of AVERAGE, COUNT, MAXIMUM, MINIMUM, STANDARD_DEVIATION, and SUM.
- **resolution** – The time interval for which to retrieve the metric: 1m (1 minute), 1h (1 hour), or 1d (1 day).
- **startDate** – The exclusive start of the range from which to query historical data, expressed in seconds in Unix epoch time.
- **endDate** – The inclusive end of the range from which to query historical data, expressed in seconds in Unix epoch time.

You can also pass any of the following parameters to refine your results:

- **maxResults** – The maximum number of results to return in one request. Defaults to 20 results.
- **nextToken** – A pagination token returned from a previous call of this operation.
- **timeOrdering** – The ordering to apply to the returned values: ASCENDING or DESCENDING.
- **qualities** – The quality to filter results by: GOOD, BAD, or UNCERTAIN.

**Note**
The `GetAssetPropertyAggregates` operation returns a TQV with a different format than other operations described in this section. The value structure contains a field for each of the aggregateTypes in the request. The timestamp contains the time that the aggregation occurred, in seconds in Unix epoch time.

**To query aggregates for an asset property (CLI)**

1. Run the following command to get aggregates for the asset property. This command queries the average and sum with a 1 hour resolution for a specific 1 hour interval. Replace `asset-id` with the ID of the asset and `property-id` with the ID of the property. Replace the parameters with the aggregates and interval to query.

```bash
aws iotsitewise get-asset-property-aggregates \
  --asset-id asset-id \
  --property-id property-id \
  --start-date 1575216000 \
  --end-date 1575219600 \
  --aggregate-types AVERAGE SUM \
  --resolution 1h
```

The operation returns a response that contains the historical TQVs of the property in the following format. The response includes only the requested aggregates.

```json
{
  "aggregatedValues": [
    {
      "timestamp": Number,
      "quality": "String",
      "value": {
        "average": Number,
        "count": Number,
        "maximum": Number,
        "minimum": Number,
        "standardDeviation": Number,
        "sum": Number
      }
    }
  ]
}
```
2. If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to the `GetAssetPropertyAggregates` operation.
Interacting with other AWS services

AWS IoT SiteWise can publish asset data to the AWS IoT MQTT publish-subscribe message broker, so that you can interact with your asset data from other AWS services. AWS IoT SiteWise assigns each asset property a unique MQTT topic that you can use to route your asset data to other AWS services using AWS IoT Core rules. For example, you can configure AWS IoT Core rules to do the following tasks:

- Identify equipment failure and notify appropriate personnel by sending data to AWS IoT Events.
- Historize select asset data for use in external software solutions by sending data to Amazon DynamoDB.
- Generate weekly reports by triggering an AWS Lambda function.

You can follow a tutorial that walks through the steps required to set up a rule that stores property values in DynamoDB. For more information, see Publishing property value updates to Amazon DynamoDB (p. 55).

For more information about how to configure a rule, see Rules in the AWS IoT Developer Guide.

You can also consume data from other AWS services back into AWS IoT SiteWise. To ingest data through the AWS IoT SiteWise rule action, see Ingesting data using AWS IoT Core rules (p. 69).

Topics
- Understanding asset properties' MQTT topics (p. 291)
- Enabling asset property notifications (p. 291)
- Querying asset property notification messages (p. 294)

Understanding asset properties' MQTT topics

Every asset property has a unique MQTT topic path in the following format.

```
/aws/sitewise/asset-models/assetModelId/assets/assetId/properties/propertyId
```

Note
AWS IoT SiteWise doesn't support the # (multi-level) topic filter wildcard in the AWS IoT Core rules engine. You can use the + (single-level) wildcard. For example, you can use the following topic filter to match all updates for a particular asset model.

```
/aws/sitewise/asset-models/assetModelId/assets/+/properties/+ 
```

To learn more about topic filter wildcards, see Topics in the AWS IoT Core Developer Guide.

Enabling asset property notifications

By default, AWS IoT SiteWise doesn't publish property value updates. You can use the AWS IoT SiteWise console or API to enable notifications for each asset property.

Topics
Enabling asset property notifications (console)

You can use the AWS IoT SiteWise console to enable notifications for an asset property.

**To enable or disable notifications for an asset property (console)**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**.
3. Choose the asset to enable a property's notifications.
   
   **Tip**
   
   You can choose the arrow icon to expand an asset hierarchy to find your asset.

4. Choose **Edit**.
5. For the asset property's **Notification status**, choose **ENABLED**.

   ![Edit Asset Property](image_url)

   You can also choose **DISABLED** to disable notifications for the asset property.

6. Choose **Save**.

Enabling asset property notifications (CLI)

You can use the AWS Command Line Interface (AWS CLI) to enable or disable notifications for an asset property.

You must know your asset's **assetId** and property's **propertyId** to complete this procedure. If you created an asset but don't know its **assetId**, use the **ListAssets** operation to view all of your assets for a
specific model. Then, use the DescribeAsset operation to view your asset's properties including property IDs.

Use the UpdateAssetProperty operation to enable or disable notifications for an asset property. Specify the following parameters:

- **assetId** – The asset's ID.
- **propertyId** – The asset property's ID.
- **propertyNotificationState** – The property value notification state: ENABLED or DISABLED.
- **propertyAlias** – The alias of the property. Specify the property's existing alias when you update the notification state. If you omit this parameter, the property's existing alias is removed.

**To enable or disable notifications for an asset property (CLI)**

1. Run the following command to retrieve the asset property's alias. Replace `asset-id` with the ID of the asset and `property-id` with the ID of the property.

   ```bash
   aws iotsitewise describe-asset-property
   --asset-id asset-id
   --property-id property-id
   ```

   The operation returns a response that contains the asset property's details in the following format. The property alias is in `assetProperty.alias` in the JSON object.

   ```json
   {
   "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
   "assetName": "Wind Turbine 7",
   "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
   "assetProperty": {
       "id": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
       "name": "Wind Speed",
       "alias": "/company/windfarm/3/turbine/7/windspeed",
       "notification": {
           "topic": "$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
           "state": "DISABLED"
       },
       "dataType": "DOUBLE",
       "unit": "m/s",
       "type": { "measurement": {} }
   }
   }
   ```

2. Run the following command to enable notifications for the asset property. Replace `property-alias` with the property alias from the previous command's response, or omit `--property-alias` to update the property without an alias.

   ```bash
   aws iotsitewise update-asset-property
   --asset-id asset-id
   --property-id property-id
   --property-notification-state ENABLED
   --property-alias property-alias
   ```

   You can also pass `--property-notification-state DISABLED` to disable notifications for the asset property.
Querying asset property notification messages

AWS IoT SiteWise publishes asset property data updates to AWS IoT Core in the following format.

```json
{
  "type": "PropertyValueUpdate",
  "payload": {
    "assetId": "String",
    "propertyId": "String",
    "values": [
      {
        "timestamp": {
          "timeInSeconds": Number,
          "offsetInNanos": Number
        },
        "quality": "String",
        "value": {
          "booleanValue": Boolean,
          "doubleValue": Number,
          "integerValue": Number,
          "stringValue": "String"
        }
      }
    ]
  }
}
```

Each structure in the `values` list is a timestamp-quality-value (TQV) structure.

- The `timestamp` contains the current Unix epoch time in seconds with nanosecond offset.
- The `quality` contains one of the following strings that indicate the quality of the data point:
  - `GOOD` – The data isn’t affected by any issues.
  - `BAD` – The data is affected by an issue such as sensor failure.
  - `UNCERTAIN` – The data is affected by an issue such as sensor inaccuracy.
- The `value` contains one of the following fields, depending on the type of the property:
  - `booleanValue`
  - `doubleValue`
  - `integerValue`
  - `stringValue`

To parse values out of the `values` array, you need to use complex nested object queries in your rules' SQL statements. For more information, see Nested object queries in the AWS IoT Developer Guide, or see the Publishing property value updates to Amazon DynamoDB (p. 55) tutorial for a specific example of parsing asset property notification messages.

### Example Example query to extract the array of values

The following statement demonstrates how to query the array of updated property values for a specific double-type property on all assets with that property.

```sql
SELECT (SELECT VALUE (value.doubleValue) FROM payload.values) AS windspeed
FROM 'aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE'
WHERE
```
type = 'PropertyValueUpdate'

The previous rule query statement outputs data in the following format.

```json
{
  "windspeed": [
    26.32020195042838,
    26.28584572975477,
    26.3526697732508,
    26.28308436111442,
    26.5718373959322,
    26.60684414074005,
    26.628738636715045,
    26.273486932802125,
    26.436379105473964,
    26.600590095377303
  ]
}
```

**Example Example query to extract a single value**

The following statement demonstrates how to query the first value from the array of property values for a specific double-type property on all assets with that property.

```sql
SELECT get((SELECT VALUE (value.doubleValue) FROM payload.values), 0) AS windspeed
FROM '$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE'
WHERE type = 'PropertyValueUpdate'
```

The previous rule query statement outputs data in the following format.

```json
{
  "windspeed": 26.32020195042838
}
```

**Important**

This rule query statement ignores value updates other than the first in each batch. Each batch can contain up to 10 values. If you need to include the remaining values, you must set up a more complex solution to output asset property values to other services. For example, you can set up a rule with an AWS Lambda action to republish each value in the array to another topic, and set up another rule to query that topic and publish each value to the desired rule action.
Exporting data to Amazon S3

AWS IoT SiteWise can export raw data and metadata to an Amazon S3 bucket in your account. Raw data includes measurements (equipment data), metrics, and transforms. AWS IoT SiteWise exports raw data to Amazon S3 once every six hours. Metadata includes asset and asset hierarchy metadata. AWS IoT SiteWise exports metadata to Amazon S3 when you change asset model definitions or asset definitions.

After you export your data to an Amazon S3 bucket, you can use the following AWS services to create historical reports or analyze and query your data:

- Run SQL queries on your data using Amazon Athena.
- Perform big data analysis using Amazon EMR.
- Search and analyze your data using Amazon OpenSearch Service.

For more information about other AWS services that can interact with your data in Amazon S3, see the list under Analytics in the AWS Management Console.

You can also use asset property notifications to export data from asset properties that you select to Amazon S3 in near-real time. For more information, see Exporting data to Amazon S3 by using asset property notifications (p. 309).

Configure storage settings

AWS IoT SiteWise saves your data in a service-managed database by default. You can use the AWS IoT SiteWise console or AWS CLI to configure the storage settings in your account so that AWS IoT SiteWise can export a copy of your data to an Amazon S3 bucket.

**Topics**

- Configure storage settings (console) (p. 296)
- Configure storage for data replication (AWS CLI) (p. 299)
- (Optional) Create an AWS IoT Analytics data store (AWS CLI) (p. 301)
- Troubleshoot (p. 302)

Configure storage settings (console)

The following procedure shows you how to configure the storage settings to export data to Amazon S3 in the AWS IoT SiteWise console.

**To configure storage settings in the console**

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, under Settings, choose Storage.
3. In the upper-right corner, choose Edit.
4. On the Edit storage page, do the following:
   a. For Storage settings, choose Enabled. The Storage settings is disabled by default.
   b. For S3 bucket location, enter the name of an existing Amazon S3 bucket and a prefix.
Configure storage settings (console)

Note

- Amazon S3 uses the prefix as a folder name in the Amazon S3 bucket. The prefix must have 1-255 characters and end with a forward slash (/). Your AWS IoT SiteWise data is saved in this folder.
- If you don't have an Amazon S3 bucket, choose View, and then create one in the Amazon S3 console. For more information, see Create your first S3 bucket in the Amazon S3 User Guide.

c. For S3 access role, do one of the following:

- Choose Create a role from an AWS managed template, AWS automatically creates an IAM role that allows AWS IoT SiteWise to send data to Amazon S3.
- Choose Use an existing role, and then choose the role that you created from the list.

Note

- You must use the same Amazon S3 bucket name for the S3 bucket location that you used in the previous step and in your IAM policy.
- Make sure that your role has the permissions shown in the following example.

Example permissions policy

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name",
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

Replace bucket-name with the name of your Amazon S3 bucket.

d. (Optional) If you want to use AWS IoT Analytics to query your data, enable AWS IoT Analytics data store. Do the following:

   i. Choose Enabled.
   ii. AWS IoT SiteWise generates a name for your data store or you can enter a different name.

AWS IoT SiteWise automatically creates a data store in AWS IoT Analytics to save your data. To query the data, you can use AWS IoT Analytics to create datasets. For more information, see Working with AWS IoT SiteWise data in the AWS IoT Analytics User Guide.

e. Choose Save.
In the **Export new data to S3** section, the **Status** can be one of the following:

- **Enabled** – AWS IoT SiteWise exports your data to the specified Amazon S3 bucket.
- **Enabling** – AWS IoT SiteWise is enabling this feature. This process can take several minutes to complete.
- **Enable_Failed** – AWS IoT SiteWise couldn't enable this feature. If you enabled AWS IoT SiteWise to send logs to Amazon CloudWatch Logs, you can use these logs to troubleshoot issues. For more information, see Monitoring AWS IoT SiteWise with Amazon CloudWatch Logs (p. 360).
- **Disabled** - This feature is disabled.
Configure storage for data replication (AWS CLI)

The following procedure shows you how to configure the storage settings to export data to Amazon S3 using AWS CLI.

To configure storage settings using AWS CLI

1. To export data to an Amazon S3 bucket in your account, run the following command to configure the storage settings. Replace `file-name` with the name of the file that contains the AWS IoT SiteWise storage configuration.

   ```bash
   aws iotsitewise put-storage-configuration --cli-input-json file://file-name.json
   ```

   **Example AWS IoT SiteWise storage configuration**

   Replace `bucket-name`, `prefix`, `aws-account`, and `role-name` with your Amazon S3 bucket name, prefix, AWS account ID, and IAM role name.

   ```json
   {
     "storageType": "MULTI_LAYER_STORAGE",
     "multiLayerStorage": {
       "customerManagedS3Storage": {
         "s3ResourceArn": "arn:aws:s3:::bucket-name/prefix/",
         "roleArn": "arn:aws:iam::aws-account:role/role-name"
       }
     }
   }
   ```

   **Note**

   - You must use the same Amazon S3 bucket name in the AWS IoT SiteWise storage configuration and IAM policy.
   - Make sure that your role has the permissions shown in the following example.

   **Example permissions policy**

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:PutObject",
           "s3:GetObject",
           "s3:DeleteObject",
           "s3:GetBucketLocation",
           "s3:ListBucket"
         ],
         "Resource": ["arn:aws:s3:::bucket-name",
                       "arn:aws:s3:::bucket-name/**"]
       }
     ]
   }
   ```

   Replace `bucket-name` with the name of your Amazon S3 bucket.
Configure storage for data replication (AWS CLI)

Example response

```json
{
   "storageType": "MULTI_LAYER_STORAGE",
   "configurationStatus": {
      "state": "UPDATE_IN_PROGRESS"
   }
}
```

**Note**
It can take a few minutes for AWS IoT SiteWise to update the storage configuration.

2. To retrieve the storage configuration information, run the following command.

```
aws iotsitewise describe-storage-configuration
```

Example response

```json
{
   "storageType": "MULTI_LAYER_STORAGE",
   "multiLayerStorage": {
      "customerManagedS3Storage": {
         "s3ResourceArn": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/torque/",
         "roleArn": "arn:aws:iam::123456789012:role/SWAccessS3Role"
      }
   },
   "configurationStatus": {
      "state": "ACTIVE"
   },
   "lastUpdateDate": "2021-03-30T15:54:14-07:00"
}
```

3. To stop exporting data to the Amazon S3 bucket, run the following command to configure storage settings.

```
aws iotsitewise put-storage-configuration --storage-type SITEWISE_DEFAULT_STORAGE
```

**Note**
By default, AWS IoT SiteWise only saves data to a service-managed database.

Example response

```json
{
   "storageType": "SITEWISE_DEFAULT_STORAGE",
   "configurationStatus": {
      "state": "UPDATE_IN_PROGRESS"
   }
}
```

4. To retrieve the storage configuration information, run the following command.

```
aws iotsitewise describe-storage-configuration
```

Example response

```json
{
}
```
(Optional) Create an AWS IoT Analytics data store (AWS CLI)

An AWS IoT Analytics data store is a scalable and queryable repository that receives and stores data. You can use the AWS IoT SiteWise console or AWS IoT Analytics APIs to create an AWS IoT Analytics data store to save your AWS IoT SiteWise data. To query the data, you create datasets by using AWS IoT Analytics. For more information, see Working with AWS IoT SiteWise data in the AWS IoT Analytics User Guide.

The following steps use AWS CLI to create a data store in AWS IoT Analytics.

To create a data store, run the following command. Replace `file-name` with the name of the file that contains the data store configuration.

```
aws iotanalytics create-datastore --cli-input-json file://file-name.json
```

**Note**

- You must specify the name of an existing Amazon S3 bucket. If you don’t have an Amazon S3 bucket, create one first. For more information, see Create your first S3 bucket in Amazon S3 User Guide.
- You must use the same Amazon S3 bucket name in the AWS IoT SiteWise storage configuration, IAM policy, and AWS IoT Analytics data store configuration.

**Example AWS IoT Analytics data store configuration**

Replace `data-store-name` and `s3-bucket-name` with your AWS IoT Analytics data store name and Amazon S3 bucket name.

```json
{
    "datastoreName": "data-store-name",
    "datastoreStorage": {
        "iotSiteWiseMultiLayerStorage": {
            "customerManagedS3Storage": {
                "bucket": "s3-bucket-name"
            }
        }
    },
    "retentionPeriod": {
        "numberOfDays": 90
    }
}
```

**Example response**

```
{
    "datastoreName": "datastore_IoTSiteWise_demo",
    "configurationStatus": {
        "state": "ACTIVE"
    },
    "lastUpdateDate": "2021-03-30T15:57:14-07:00"
}
```
Troubleshoot

Use the following information to troubleshoot and resolve issues with the storage configuration.

Issues

- Error: Bucket doesn't exist (p. 302)
- Error: Access denied to S3 path (p. 302)
- Error: Role ARN can't be assumed (p. 303)
- Error: Failed to access cross-Region S3 bucket (p. 303)

Error: Bucket doesn't exist

Solution: AWS IoT SiteWise couldn't find your Amazon S3 bucket. Make sure you enter the name of an existing Amazon S3 bucket in the current Region.

Error: Access denied to S3 path

Solution: AWS IoT SiteWise couldn't access your Amazon S3 bucket. Do the following:

- Make sure that you use the same Amazon S3 bucket that you specified in the IAM policy.
- Make sure that your role has the permissions shown in the following example.

Example permissions policy

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:GetBucketLocation",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name",
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

Replace `bucket-name` with the name of your Amazon S3 bucket.
Error: Role ARN can't be assumed

Solution: AWS IoT SiteWise couldn't assume the IAM role on your behalf. Make sure that your role trusts the following service: iotsitewise.amazonaws.com. For more information, see I can't assume a role see IAM User Guide.

Error: Failed to access cross-Region S3 bucket

Solution: The Amazon S3 bucket that you specified is in a different AWS Region. Make sure that your Amazon S3 bucket and AWS IoT SiteWise assets are in the same Region.

File paths and schemas

AWS IoT SiteWise exports raw data, asset metadata, and asset hierarchy metadata to Amazon S3. The following describes the file paths and schemas of data that is exported to your Amazon S3 bucket.

Topics
- Equipment data, metrics, and transforms (raw data) (p. 303)
- Asset metadata (p. 305)
- Asset hierarchy metadata (p. 308)

Equipment data, metrics, and transforms (raw data)

AWS IoT SiteWise exports raw data that includes equipment data, metrics, and transforms to Amazon S3 once every six hours. Raw data is saved in Amazon S3 in the Apache AVRO (.avro) format.

File path

To generate file paths of raw data in Amazon S3, AWS IoT SiteWise uses the following template.

```
keyPrefix/raw/startYear=startYear/startMonth=startMonth/startDay=startDay/
seriesBucket=seriesBucket/raw_{timeseriesId}_startTimestamp_{quality}.avro
```

Every file path to raw data in Amazon S3 contains the following components.

<table>
<thead>
<tr>
<th>Path component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyPrefix</td>
<td>The Amazon S3 prefix that you specified in the AWS IoT SiteWise storage configuration. Amazon S3 uses the prefix as a folder name in the bucket.</td>
</tr>
<tr>
<td>raw</td>
<td>The folder that stores time series data from equipment, metrics, or transforms. The raw folder is saved in the prefix folder.</td>
</tr>
<tr>
<td>seriesBucket</td>
<td>A hexadecimal number between 00 and ff. This number is derived from timeSeriesId. This partition is used to increase throughput when AWS IoT SiteWise writes to Amazon S3. When you</td>
</tr>
</tbody>
</table>
Path component | Description
---|---
 | use Amazon Athena to run queries, you can use the partition for fine-grain partitioning to improve query performance.
 | seriesBucket and timeSeriesBucket in the asset metadata are the same number.

| startYear | The year of the exclusive start time associated with the time series data. |
| startMonth | The month of the exclusive start time associated with the time series data. |
| startDay | The day of the month of the exclusive start time associated with the time series data. |

| fileName | The file name uses the underscore (_) character as a delimiter to separate the following:
- The raw prefix.
- The timeSeriesId value.
- The epoch timestamp of the exclusive start time associated with the time series data.
- The quality of the data. Valid values: GOOD, BAD, and UNCERTAIN. For more information, see AssetPropertyValue in the AWS IoT SiteWise API Reference. |

The file is saved in the .avro format by using the Snappy compression.

**Example file path to raw data in Amazon S3**

`keyPrefix/raw/startYear=2021/startMonth=1/startDay=2/seriesBucket=a2/raw_7020c8e2-e6db-40fa-9845-ed0ddd4c77d_95e63da7-d34e-43e1-bc6f-1b490154b07a_1609577700_GOOD.avro`

**Fields**

The schema of raw data that is exported to Amazon S3 contains the following fields.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Supported types</th>
<th>Default type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seriesId</td>
<td>string</td>
<td>N/A</td>
<td>The ID that identifies the time series data from equipment, metrics, or transforms. You can use this field to join raw data and asset metadata in queries.</td>
</tr>
<tr>
<td>timeInSeconds</td>
<td>long</td>
<td>N/A</td>
<td>The timestamp date, in seconds, in the Unix epoch format.</td>
</tr>
</tbody>
</table>
### Field name | Supported types | Default type | Description
--- | --- | --- | ---
offsetInNanos | long | N/A | The nanosecond offset from timeInSeconds.
quality | string | N/A | The quality of the time series value.
doubleValue | double or null | null | Time series data of type double (floating point number).
stringValue | string or null | null | Time series data of type string (sequence of characters).
integerValue | int or null | null | Time series data of type integer (whole number).
booleanValue | boolean or null | null | Time series data of type Boolean (true or false).
jsonValue | string or null | null | Time series data of type JSON (complex data types stored as a string).
recordVersion | long or null | null | The version number for the record. You can use the version number to select the latest record. Newer records have larger version numbers.

**Example raw data in Amazon S3**

```json
"seriesId":"e9687d2a-0dbe-4f65-9ed6-6f443cba41f7_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeInSeconds":1625675887,"offsetInNanos":0,"quality":"GOOD","doubleValue":
{"double":0.75},"stringValue":null,"integerValue":null,"booleanValue":null,"jsonValue":null,"recordVersion":null

{"seriesId":"e9687d2a-0dbe-4f65-9ed6-6f443cba41f7_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeInSeconds":1625675889,"offsetInNanos":0,"quality":"GOOD","doubleValue":
{"double":0.69},"stringValue":null,"integerValue":null,"booleanValue":null,"jsonValue":null,"recordVersion":null

{"seriesId":"e9687d2a-0dbe-4f65-9ed6-6f443cba41f7_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeInSeconds":1625675890,"offsetInNanos":0,"quality":"GOOD","doubleValue":
{"double":0.66},"stringValue":null,"integerValue":null,"booleanValue":null,"jsonValue":null,"recordVersion":null

{"seriesId":"e9687d2a-0dbe-4f65-9ed6-6f443cba41f7_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeInSeconds":1625675891,"offsetInNanos":0,"quality":"GOOD","doubleValue":
{"double":0.92},"stringValue":null,"integerValue":null,"booleanValue":null,"jsonValue":null,"recordVersion":null

{"seriesId":"e9687d2a-0dbe-4f65-9ed6-6f443cba41f7_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeInSeconds":1625675892,"offsetInNanos":0,"quality":"GOOD","doubleValue":
{"double":0.73},"stringValue":null,"integerValue":null,"booleanValue":null,"jsonValue":null,"recordVersion":null

```

**Asset metadata**

When you enable AWS IoT SiteWise to export data to Amazon S3 for the first time, asset metadata is exported to Amazon S3. After the initial configuration, AWS IoT SiteWise exports asset metadata to
Asset metadata

Amazon S3 only when you change asset model definitions or asset definitions. Asset metadata is saved in Amazon S3 in the Newline Delimited JSON (.ndjson) format.

**File path**

To generate file paths of asset metadata in Amazon S3, AWS IoT SiteWise uses the following template.

```
keyPrefix/asset_metadata/asset_{assetId}.ndjson
```

Every file path to asset metadata in Amazon S3 contains the following components.

**File path**

<table>
<thead>
<tr>
<th>Path component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyPrefix</td>
<td>The Amazon S3 prefix that you specified in the AWS IoT SiteWises storage configuration. Amazon S3 uses the prefix as a folder name in the bucket.</td>
</tr>
<tr>
<td>asset_metadata</td>
<td>The folder that stores asset metadata. The asset_metadata folder is saved in the prefix folder.</td>
</tr>
<tr>
<td>fileName</td>
<td>The file name uses the underscore (_) character as a delimiter to separate the following:</td>
</tr>
<tr>
<td></td>
<td>• The asset prefix.</td>
</tr>
<tr>
<td></td>
<td>• The assetId value.</td>
</tr>
<tr>
<td></td>
<td>The file is saved in the .ndjson format.</td>
</tr>
</tbody>
</table>

**Example file path to asset metadata in Amazon S3**

keyPrefix/asset_metadata/asset_35901915-d476-4dca-8637-d9ed4df939ed.ndjson

**Fields**

The schema of asset metadata that is exported to Amazon S3 contains the following fields.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assetId</td>
<td>The ID of the asset.</td>
</tr>
<tr>
<td>assetName</td>
<td>The name of the asset.</td>
</tr>
<tr>
<td>assetModelId</td>
<td>The ID of the asset model used to create this asset.</td>
</tr>
<tr>
<td>assetModelName</td>
<td>The name of the asset model.</td>
</tr>
<tr>
<td>assetPropertyId</td>
<td>The ID of the asset property.</td>
</tr>
<tr>
<td>assetPropertyName</td>
<td>The name of the asset property.</td>
</tr>
<tr>
<td>assetPropertyDataType</td>
<td>The data type of the asset property.</td>
</tr>
</tbody>
</table>
### Asset metadata

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assetPropertyUnit</td>
<td>The unit of the asset property (for example, Newtons and RPM).</td>
</tr>
<tr>
<td>assetPropertyAlias</td>
<td>The alias that identifies the asset property, such as an OPC-UA server data stream path (for example, /company/windfarm/3/turbine/7/temperature).</td>
</tr>
<tr>
<td>timeSeriesId</td>
<td>The ID that identifies the time series data from equipment, metrics, or transforms. You can use this field to join raw data and asset metadata in queries.</td>
</tr>
<tr>
<td>timeSeriesBucket</td>
<td>A hexadecimal number between 00 and ff. This number is derived from timeSeriesId. This partition is used to increase throughput when AWS IoT SiteWise writes to Amazon S3. When you use Amazon Athena to run queries, you can use the partition for fine-grain partitioning to improve query performance. The timeSeriesBucket and seriesBucket in the file path to raw data are the same number.</td>
</tr>
<tr>
<td>assetCompositeModelDescription</td>
<td>The description of the composite model.</td>
</tr>
<tr>
<td>assetCompositeModelName</td>
<td>The name of the composite model.</td>
</tr>
<tr>
<td>assetCompositeModelType</td>
<td>The type of the composite model. For alarm composite models, this type is AWS/ALARM.</td>
</tr>
<tr>
<td>assetCreationDate</td>
<td>The date the asset was created, in Unix epoch time.</td>
</tr>
<tr>
<td>assetLastUpdateDate</td>
<td>The date the asset was last updated, in Unix epoch time.</td>
</tr>
<tr>
<td>assetStatusErrorCode</td>
<td>The error code.</td>
</tr>
<tr>
<td>assetStatusErrorMessage</td>
<td>The error message.</td>
</tr>
<tr>
<td>assetStatusState</td>
<td>The current status of the asset.</td>
</tr>
</tbody>
</table>

### Example asset metadata in Amazon S3

```json
{"assetId":"7020c8e2-e6db-40fa-9845-ed0d6d4c77d","assetName":"Wind Turbine Asset 2","assetModelId":"ec1d924f-f07d-444f-b072-e2994c165d35","assetModelName":"Wind Turbine Asset Model","assetPropertyId":"95e63da7-d34e-43e1-bc6f-1b490154b07a","assetPropertyName":"Temperature","assetPropertyDataType":"DOUBLE","assetPropertyUnit":"Celsius","assetPropertyAlias":"USA/Washington/Seattle/WT2/temp","timeSeriesId":"7020c8e2-e6db-40fa-9845-ed0d6d4c77d_95e63da7-d34e-43e1-bc6f-1b490154b07a","timeSeriesBucket":"f6","assetArn":null,"assetCompositeModelDescription":null,"assetCompositeModelName":null,"assetCompositeModelType":null,"assetCreationDate":null,"assetLastUpdateDate":null,"assetStatusErrorCode":null,"assetStatusErrorMessage":null,"assetStatusState":null}
```

```json
{"assetId":"7020c8e2-e6db-40fa-9845-ed0d6d4c77d","assetName":"Wind Turbine Asset 2","assetModelId":"ec1d924f-f07d-444f-b072-e2994c165d35","assetModelName":"Wind Turbine Asset Model","assetPropertyId":"c706d54d-4c11-42dc-9a01-63662fc697b4","assetPropertyName":"Pressure","assetPropertyDataType":"DOUBLE","assetPropertyUnit":"KiloPascal","assetPropertyAlias":"USA/Washington/Seattle/WT2/pressure","timeSeriesId":"7020c8e2-e6db-40fa-9845-ed0d6d4c77d_c706d54d-4c11-42dc-9a01-63662fc697b4","timeSeriesBucket":"1e","assetArn":null,"assetCompositeModelDescription":null,"assetCompositeModelName":null,"assetCompositeModelType":null,"assetCreationDate":null,"assetLastUpdateDate":null,"assetStatusErrorCode":null,"assetStatusErrorMessage":null,"assetStatusState":null}
```
Asset hierarchy metadata

When you enable AWS IoT SiteWise to export data to Amazon S3 for the first time, asset hierarchy metadata is exported to Amazon S3. After the initial configuration, AWS IoT SiteWise exports asset hierarchy metadata to Amazon S3 only when you make changes to asset model or asset definitions. Asset hierarchy metadata is saved in Amazon S3 in the Newline Delimited JSON (.ndjson) format.

File path

To generate file paths of asset hierarchy metadata in Amazon S3, AWS IoT SiteWise uses the following template.

```
keyPrefix/asset_hierarchy_metadata/(parentAssetId)
```

Every file path to asset hierarchy metadata in Amazon S3 contains the following components.

<table>
<thead>
<tr>
<th>Path component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyPrefix</td>
<td>The Amazon S3 prefix that you specified in the AWS IoT SiteWise storage configuration. Amazon S3 uses the prefix as a folder name in the bucket.</td>
</tr>
<tr>
<td>asset_hierarchy_metadata</td>
<td>The folder that stores asset hierarchy metadata. The asset_hierarchy_metadata folder is saved in the prefix folder.</td>
</tr>
<tr>
<td>fileName</td>
<td>The file name uses the underscore (_) character as a delimiter to separate the following:</td>
</tr>
<tr>
<td></td>
<td>• The parentAssetId value.</td>
</tr>
<tr>
<td></td>
<td>• The hierarchyId value.</td>
</tr>
<tr>
<td></td>
<td>The file is saved in the .ndjson format.</td>
</tr>
</tbody>
</table>

**Example file path to asset hierarchy metadata in Amazon S3**

keyPrefix/asset_hierarchy_metadata/35901915-d476-4dca-8637-d9ed4df939ed_c5b3ced8-589a-48c7-9998-cdcccfc9747a0.ndjson

**Fields**

The schema of asset hierarchy metadata that is exported to Amazon S3 contains the following fields.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceAssetId</td>
<td>The ID of the source asset in this asset relationship.</td>
</tr>
</tbody>
</table>
### Using asset property notifications

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetAssetId</td>
<td>The ID of the target asset in this asset relationship.</td>
</tr>
<tr>
<td>hierarchyId</td>
<td>The ID of the hierarchy.</td>
</tr>
<tr>
<td>associationType</td>
<td>The association type of this asset relationship.</td>
</tr>
<tr>
<td></td>
<td>The value must be CHILD. The target asset is a child asset of the source asset.</td>
</tr>
</tbody>
</table>

**Example asset hierarchy metadata in Amazon S3**

```
{"sourceAssetId":"80388e72-2284-44fb-9c89-bfba0dfedd2","targetAssetId":"2b866c25-0c74-4750-bdf5-b73683c8a2a2","hierarchyId":"bbed9f59-0412-4585-a61d-6044db526aee","associationType":"CHILD"}
{"sourceAssetId":"80388e72-2284-44fb-9c89-bfba0dfedd2","targetAssetId":"6b51246e-984d-460d-bc0b-470ea47d1e31","hierarchyId":"bbed9f59-0412-4585-a61d-6044db526aee","associationType":"CHILD"}
```

**To view your data in Amazon S3**

1. Navigate to the Amazon S3 console.
2. In the navigation pane, choose **Buckets**, and then choose your Amazon S3 bucket.
3. Navigate to the folder that contains the raw data, asset metadata, or asset hierarchy metadata.
4. Select the files, and then from **Actions**, choose **Download**.

---

**Exporting data to Amazon S3 by using asset property notifications**

You can export incoming data from AWS IoT SiteWise to an Amazon S3 bucket in your account. You can back up your data in a format that you can use to create historical reports or to analyze your data with complex methods.

AWS IoT SiteWise provides this feature as an AWS CloudFormation template. When you create a stack from the template, AWS CloudFormation creates the required AWS resources to stream incoming data from AWS IoT SiteWise to an S3 bucket.

Then, the S3 bucket receives all of your asset property data sent from AWS IoT SiteWise property value update messages. The S3 bucket also receives your asset metadata, which includes asset and property names and other information.

For more information about how to enable property value update messages for the asset properties to export to Amazon S3, see [Interacting with other AWS services (p. 291)](https://docs.aws.amazon.com/iot-sitewise/latest/userguide/interacting-with-other-aws-services.html).

This feature stores your asset property data and asset metadata in the [Apache Parquet](https://parquet.apache.org/) format in Amazon S3. Parquet is a columnar data format that saves space and enables faster queries compared to row-oriented formats like JSON.
Note
When this feature retrieves asset metadata, it supports up to approximately 1,500 assets. This limitation applies only to asset metadata. This limitation doesn't apply to the number of assets supported when the feature exports asset property data.

Each resource's name includes a prefix that you can customize when you create the stack. Resources include the following:

- An Amazon S3 bucket
- AWS Lambda functions
- An AWS IoT Core rule
- AWS Identity and Access Management roles
- An Amazon Kinesis Data Firehose stream
- An AWS Glue database

For a complete list, see Resources created from the template (p. 318).

Important
You will be charged for the resources that this AWS CloudFormation template creates and consumes. These charges include data storage and data transfer for multiple AWS services.

Topics
- Creating the AWS CloudFormation stack (p. 310)
- Viewing your data in Amazon S3 (p. 311)
- Analyzing exported data with Amazon Athena (p. 312)
- Resources created from the template (p. 318)

Creating the AWS CloudFormation stack

You can create a stack in AWS CloudFormation to export your asset data to Amazon S3.

To export data to Amazon S3

1. Open the AWS CloudFormation template and sign in to the AWS Management Console.
2. On the Create stack page, choose Next at the bottom of the page.
3. On the Specify stack details page, enter a BucketName for the S3 bucket that this template creates in order to receive asset data. This bucket name must be globally unique. For more information, see Rules for bucket naming in the Amazon Simple Storage Service Developer Guide.
4. (Optional) Change any of the template's other parameters:
   - GlobalResourcePrefix – A prefix for names of global resources, such as IAM roles, created from this template.
   - LocalResourcePrefix – A prefix for names of resources created from this template in the current Region.

   Note
   If you create this template multiple times, you should change the bucket name and resource prefix parameters in order to avoid resource name conflicts.
5. Choose Next.
6. On the Configure stack options page, choose Next.
7. At the bottom of the page, select the check box that says I acknowledge that AWS CloudFormation might create IAM resources.

8. Choose Create stack.

The stack takes a few minutes to create. If the stack fails to create, your account might have insufficient permissions, or you might have entered a bucket name that already exists. Use the following steps to delete the stack and try again:

a. Choose Delete in the upper-right corner.

   The stack takes a few minutes to delete.

   Note
   AWS CloudFormation doesn't delete S3 buckets or CloudWatch log groups. You can delete these resources in the consoles for those services.

b. If the stack fails to delete, choose Delete again.

c. If the stack fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.

9. After the AWS CloudFormation stack creates successfully, follow the next procedure to explore your asset property data in Amazon S3.

Important
After you create the stack, you can see the new resources in your AWS account. The feature might stop working correctly if you delete or modify these resources. We recommend that you don't modify these resources unless you want to stop sending data to the bucket or want to customize this feature.

Viewing your data in Amazon S3

After you create the feature, you can view your asset property data and asset metadata in Amazon S3.

Note
Asset metadata updates every six hours. You might need to wait up to six hours to see asset metadata appear in the S3 bucket.

This feature stores asset property data in the following columns, where each row contains a data point:

- **type** – The type of property notification (PropertyValueUpdate).
- **asset_id** – The ID of the asset that received a data point.
- **asset_property_id** – The ID of the property that received a data point for the asset.
- **time_in_seconds** – The time at which the data was received, expressed in seconds in Unix epoch time.
- **offset_in_nanos** – The nanosecond offset from timeInSeconds.
- **asset_property_quality** – The quality of the data point: GOOD, UNCERTAIN, or BAD.
- **asset_property_value** – The value of the data point.
- **asset_property_data_type** – The data type of the asset property: boolean, double, integer, or string.

This feature stores asset metadata in the following columns, where each row contains an asset property:

- **asset_id** – The ID of the asset.
- **asset_name** – The name of the asset.
- **asset_model_id** – The ID of the asset's model.
- **asset_property_id** – The ID of the asset property.
Analyzing exported data

- **asset_property_name** – The name of the asset property.
- **asset_property_data_type** – The data type of the asset property: BOOLEAN, DOUBLE, INTEGER, or STRING.
- **asset_property_unit** – The unit of the asset property.
- **asset_property_alias** – The alias of the asset property.

To view your AWS IoT SiteWise data in Amazon S3

1. Navigate to the Amazon S3 console.
2. From the list of buckets, choose the bucket with the name you chose when you created the template.
3. In the bucket, choose one of the following folders:
   - **asset-property-updates** – This folder contains asset property data exported from AWS IoT SiteWise.
   - **asset-metadata** – This folder contains asset details exported from AWS IoT SiteWise.
4. Choose the object that you want to view.
5. On the object's page, do the following:
   a. Choose the Select from tab.
      In this panel, you can preview records from Parquet files.
   b. For File format, choose Parquet.
   c. To show the contents of the file in JSON format, choose Show file preview.

**Note**
If new data doesn't appear in the bucket, check that you enabled property value update notifications for your asset properties. For more information, see Interacting with other AWS services (p. 291).

For more information about how to analyze your asset data stored in the S3 bucket, see Analyzing exported data with Amazon Athena (p. 312).

Analyzing exported data with Amazon Athena

After you have your asset property data in Amazon S3, you can use several AWS services to generate reports or analyze and query your data:

- Run SQL queries on your data using Amazon Athena.
- Perform big data analysis using Amazon EMR.
- Search and analyze your data using Amazon OpenSearch Service.

You can find other AWS services that can interact with your data in Amazon S3 listed under Analytics in the AWS Management Console.

**Note**
The stack creates an AWS Glue database to format asset property data. You can't query this database for asset data. Follow the steps in this section to create an AWS Glue database that you can query.

In this tutorial, you learn how to configure the prerequisites to use Amazon Athena and how to use Athena to run SQL queries on your exported AWS IoT SiteWise asset data. To query data with Athena,
you must first populate the AWS Glue Data Catalog with your asset data. The Data Catalog contains databases and tables, and Athena can access data in the Data Catalog. You can create an AWS Glue crawler that regularly updates the Data Catalog with your exported asset data.

**Topics**
- Configuring a crawler to populate the AWS Glue Data Catalog (p. 313)
- Querying data with Athena (p. 315)

**Configuring a crawler to populate the AWS Glue Data Catalog**

AWS Glue crawlers crawl data stores to populate tables in the AWS Glue Data Catalog. In this procedure, you create and run an AWS Glue crawler for your S3 bucket that contains exported asset data. The crawler creates a table for asset property updates and a table for asset metadata. Then, you can perform SQL queries on these tables with Athena. For more information, see Populating the AWS Glue Data Catalog and Defining crawlers in the *AWS Glue Developer Guide*.

**To create an AWS Glue crawler**

1. Navigate to the AWS Glue console.
2. In the navigation pane, choose **Crawlers**.
3. Choose **Add crawler**.
4. On the **Add crawler** page, do the following:
   a. Enter a name for your crawler, such as `IotSiteWiseDataCrawler`, and then choose **Next**.
   b. For **Crawler source type**, choose **Data stores**, and then choose **Next**.
   c. On the **Add a data store page**, do the following:
      i. For **Choose a data store**, choose **S3**.
      ii. In **Include path**, enter `s3://DOC-EXAMPLE-BUCKET1` to add your asset data bucket as a data store. Replace `DOC-EXAMPLE-BUCKET1` with the bucket name that you chose when you created the stack.
      iii. Choose **Next**.
d. On the **Add another data store** page, choose **No**, and then choose **Next**.

e. On the **Choose an IAM role** page, do the following:

   i. To create a new service role that allows AWS Glue to access the S3 bucket, choose **Create an IAM role**.

   ii. Enter a suffix for your role's name, such as **IoTSiteWiseDataCrawler**.

   iii. Choose **Next**.

f. For **Frequency**, choose **Hourly**, and then choose **Next**. The crawler updates the tables with new data each time it runs, so you can choose any frequency that fits your use case.

g. On the **Configure the crawler's output** page, do the following:

   i. Choose **Add database** to create an AWS Glue database for your asset data.

   ii. Enter a name for the database, such as **iot_sitewise_asset_database**.

   iii. Choose **Create**.

   iv. Choose **Next**.

h. Review the crawler details, and then choose **Finish**.

   ![](image)

   By default, your new crawler doesn't immediately run. You must manually run it or wait until it runs on its configured schedule.

**To run a crawler**

1. On the **Crawlers** page, select the check box for your new crawler, and then choose **Run crawler**.
2. Wait until the crawler finishes and has a status of **Ready**.

   The crawler can take several minutes to run, and its status updates automatically.

3. In the navigation pane, choose **Tables**.

   You should see two new tables: **asset_metadata** and **asset_property_updates**.

### Querying data with Athena

Athena automatically discovers your asset data tables in the AWS Glue Data Catalog. To perform queries on the intersection of these tables, you can create a view, which is a logical data table. For more information, see **Working with views** in the *Amazon Athena User Guide*.

After you create a view that combines asset property data and metadata, you can run queries that output property values with asset and property names attached. For more information, see **Running SQL queries using Amazon Athena** in the *Amazon Athena User Guide*.

### To query asset data with Athena

1. Navigate to the **Athena console**.

   If the **Getting started** page appears, choose **Get Started**.

2. If you're using Athena for the first time, complete the following steps to configure an S3 bucket for query results. Athena stores the results of your queries in this bucket.

   **Important**
   
   Use a different bucket than your asset data bucket, so the crawler that you created earlier doesn't crawl query results. We recommend that you create a bucket to use only for Athena query results. For more information, see **How do I create an S3 bucket?** in the *Amazon Simple Storage Service Console User Guide*.

   a. Choose **Settings**.
   
   b. In **Query result location**, enter the S3 bucket for Athena query results. The bucket must end with `/`.  

---

315
c. Choose Save.

3. The left panel contains the data source to query. Do the following:
   
   a. For **Data source**, choose **AwsDataCatalog** to use the AWS Glue Data Catalog.
   
   b. For **Database**, choose the AWS Glue database that you created with the crawler.

   ![Athena Query Editor](image)

   You should see two tables: **asset_metadata** and **asset_property_updates**.

4. To create a view from the combination of asset property data and metadata, enter the following query, and then choose **Run query**.

   ```sql
   CREATE OR REPLACE VIEW iot_sitewise_asset_data AS
   ```
SELECT "from_unixtime"("time_in_seconds" + ("offset_in_nanos" / 1000000000))
"timestamp",
"metadata"."asset_name",
"metadata"."asset_property_name",
"data"."asset_property_value",
"metadata"."asset_property_unit",
"metadata"."asset_property_alias"
FROM ( "iot_sitewise_asset_database"."asset_property_updates" data
    INNER JOIN "iot_sitewise_asset_database"."asset_metadata" metadata
    ON ( "data"."asset_id" = "metadata"."asset_id" )
    AND ("data"."asset_property_id" = "metadata"."asset_property_id") ) );

This query joins the asset property data and metadata tables on asset ID and property ID to create a view. You can run this query multiple times because it replaces the existing view if the view already exists.

5. To add a new query, choose the + icon.
6. To view a sample of asset data, enter the following query, and then choose Run query. Replace the timestamps with an interval for which your bucket has data.

```
SELECT *
FROM "iot_sitewise_asset_database"."iot_sitewise_asset_data"
WHERE "timestamp"
    BETWEEN TIMESTAMP '2020-05-14 12:00:00.000'
    AND TIMESTAMP '2020-05-14 13:00:00.000'
ORDER BY  "timestamp" DESC LIMIT 50;
```

This query outputs up to 50 data points between two timestamps, with the most recent entries shown first.

Your query output might look similar to the following results.

You can now run queries useful to your AWS IoT SiteWise application. For more information, see SQL reference for Amazon Athena in the *Amazon Athena User Guide*. 

317
Resources created from the template

When you create a stack from the template, AWS CloudFormation creates the following resources. Most resources' names include a prefix that you can customize when you create the stack.

Resource name parameters

- **BucketName** – The name of the S3 bucket created from this template that receives asset data.
- **GlobalResourcePrefix** – A prefix for names of global resources created from this template. Defaults to `sitewise-export-to-s3`.
- **LocalResourcePrefix** – A prefix for names of resources created from this template in the current Region. Defaults to `sitewise_export_to_s3`.

Resources created by the AWS CloudFormation template

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S3 bucket for processed data</strong></td>
<td>This bucket contains two folders. One folder receives the flattened, formatted data from the Kinesis Data Firehose delivery stream, and the other folder receives asset metadata.</td>
<td><code>${BucketName}</code></td>
</tr>
<tr>
<td><strong>AWS Glue database</strong></td>
<td>This database contains the AWS Glue table that this stack creates.</td>
<td><code>${LocalResourcePrefix}_firehose_glue_database</code></td>
</tr>
<tr>
<td><strong>AWS Glue table</strong></td>
<td>The Kinesis Data Firehose delivery stream uses this table to format data to Parquet format.</td>
<td><code>${LocalResourcePrefix}_firehose_glue_table</code></td>
</tr>
<tr>
<td><strong>AWS Lambda function that transforms data</strong></td>
<td>This function flattens the array of values in property value notification messages sent from AWS IoT SiteWise.</td>
<td><code>${LocalResourcePrefix}_lambda_transform_function</code></td>
</tr>
<tr>
<td><strong>IAM role for the transform Lambda function</strong></td>
<td>This role allows Lambda to store runtime logs for the transform function.</td>
<td><code>${GlobalResourcePrefix}-lambda-transform-role</code></td>
</tr>
<tr>
<td><strong>IAM policy for the transform Lambda function role</strong></td>
<td>This policy allows Lambda to store execution logs for the transform function.</td>
<td><code>${GlobalResourcePrefix}-lambda-transform-policy</code></td>
</tr>
<tr>
<td><strong>CloudWatch Logs log group for the transform function</strong></td>
<td>This log group contains logs for the transform function.</td>
<td><code>/aws/lambda/${LocalResourcePrefix}_lambda_transform_function</code></td>
</tr>
<tr>
<td><strong>Lambda function that collects asset metadata</strong></td>
<td>This function retrieves details about assets in AWS IoT SiteWise and stores the details in an Amazon S3 bucket that this stack creates.</td>
<td><code>${LocalResourcePrefix}_lambda_metadata_function</code></td>
</tr>
<tr>
<td><strong>Lambda layer for the metadata function</strong></td>
<td>This layer provides an AWS SDK that contains AWS IoT SiteWise</td>
<td><code>${LocalResourcePrefix}_lambda_metadata_layer</code></td>
</tr>
<tr>
<td>Resource</td>
<td>Description</td>
<td>Name</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>IAM role for the metadata Lambda function</td>
<td>This role allows Lambda to retrieve details about assets in AWS IoT SiteWise.</td>
<td>${GlobalResourcePrefix}-lambda-metadata-role</td>
</tr>
<tr>
<td>IAM policy for the metadata Lambda function role</td>
<td>This policy allows Lambda to retrieve details about assets in AWS IoT SiteWise.</td>
<td>${GlobalResourcePrefix}-lambda-metadata-policy</td>
</tr>
<tr>
<td>EventBridge scheduled event for the metadata Lambda function</td>
<td>This scheduled event runs the metadata Lambda every 6 hours to update the asset metadata bucket.</td>
<td>${LocalResourcePrefix}-metadata-event</td>
</tr>
<tr>
<td>CloudWatch Logs log group for the metadata function</td>
<td>This log group contains logs for the metadata function.</td>
<td>/aws/lambda/${LocalResourcePrefix}_lambda_metadata_function</td>
</tr>
<tr>
<td>AWS IoT rule</td>
<td>This rule queries property value notification messages and sends asset data to an Amazon Kinesis Data Firehose delivery stream.</td>
<td>${LocalResourcePrefix}_iot_topic_rule</td>
</tr>
<tr>
<td>IAM role for the AWS IoT rule</td>
<td>This role allows AWS IoT to send data to the Kinesis Data Firehose delivery stream.</td>
<td>${GlobalResourcePrefix}-core-firehose-role</td>
</tr>
<tr>
<td>IAM policy for the AWS IoT rule role</td>
<td>This policy allows AWS IoT to send data to the Kinesis Data Firehose delivery stream.</td>
<td>${GlobalResourcePrefix}-core-firehose-policy</td>
</tr>
<tr>
<td>Kinesis Data Firehose delivery stream</td>
<td>This delivery stream consumes data from the AWS IoT rule, flattens the data with a Lambda function, and delivers the data to Amazon S3.</td>
<td>${LocalResourcePrefix}_firehose_delivery_stream</td>
</tr>
<tr>
<td>IAM role for the delivery stream</td>
<td>This role allows Kinesis Data Firehose to perform operations on the S3 bucket, AWS Glue table, Lambda functions, and CloudWatch Logs log group.</td>
<td>${GlobalResourcePrefix}-firehose-delivery-role</td>
</tr>
<tr>
<td>CloudWatch Logs log group for the delivery stream</td>
<td>This log group contains a log stream, S3 Delivery, that receives logs about the Kinesis Data Firehose delivery stream.</td>
<td>/aws/kinesissfirehose/${LocalResourcePrefix}_firehose_delivery_stream</td>
</tr>
</tbody>
</table>
Security in AWS IoT SiteWise

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 SiteWise, 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 SiteWise. The following topics show you how to configure AWS IoT SiteWise 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 SiteWise resources.

**Topics**
- Data protection in AWS IoT SiteWise (p. 320)
- Identity and access management for AWS IoT SiteWise (p. 325)
- Compliance validation for AWS IoT SiteWise (p. 353)
- Resilience in AWS IoT SiteWise (p. 353)
- Infrastructure security in AWS IoT SiteWise (p. 354)
- Configuration and vulnerability analysis in AWS IoT SiteWise (p. 354)
- AWS IoT SiteWise and interface VPC endpoints (AWS PrivateLink) (p. 355)
- Security best practices for AWS IoT SiteWise (p. 358)

Data protection in AWS IoT SiteWise

The AWS shared responsibility model applies to data protection in AWS IoT SiteWise. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
Data encryption

Data encryption refers to protecting data while in-transit (as it travels to and from AWS IoT SiteWise, and between gateways and servers), and at rest (while it is stored on local devices or in AWS services). You can protect data in transit using Transport Layer Security (TLS) or at rest using client-side encryption.

AWS IoT SiteWise edge processing exposes APIs that are hosted within SiteWise gateways and accessible over the local network. These APIs are exposed over a TLS connection backed by a server-certificate owned by the AWS IoT SiteWise Edge connector. For client authentication, these APIs use an access-control password. The server-certificate private-key and the access-control password are both stored on disk. AWS IoT SiteWise edge processing relies on file-system encryption for the security of these credentials at rest.

For more information about server-side encryption and client-side encryption, review the topics listed below.

Topics
- Encryption at rest (p. 321)
- Encryption in transit (p. 323)
- Key management (p. 324)

Encryption at rest

AWS IoT SiteWise stores your data in the AWS Cloud and on gateways.

Data at rest in the AWS Cloud

AWS IoT SiteWise stores data in other AWS services that encrypt data at rest by default. Encryption at rest integrates with AWS Key Management Service (AWS KMS) for managing the encryption key that is used to encrypt your asset property values and aggregate values in AWS IoT SiteWise. You can choose to use a customer managed key to encrypt asset property values and aggregate values in AWS IoT SiteWise. You can create, manage, and view your encryption key through AWS KMS.

You can choose an AWS owned key to encrypt your data, or choose a customer managed key to encrypt your asset property values and aggregate values:
How it works

Encryption at rest integrates with AWS KMS for managing the encryption key that is used to encrypt your data.

- AWS owned key – Default encryption key. AWS IoT SiteWise owns this key. You can’t view this key in your AWS account. You also can’t see operations on the key in AWS CloudTrail logs. You can use this key at no additional charge.
- Customer managed key – The key is stored in your account, which you create, own, and manage. You have full control over the KMS key. Additional AWS KMS charges apply.

AWS owned keys

AWS owned keys aren’t stored in your account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You can’t view, manage, use AWS owned keys, or audit their use. However, you don’t need to do any work or change any programs to protect the keys that encrypt your data.

You’re not charged a monthly fee or a usage fee if you use AWS owned keys, and they don't count against AWS KMS quotas for your account.

Customer managed keys

Customer managed keys are KMS keys in your account that you create, own, and manage. You have full control over these KMS keys, such as the following:

- Establishing and maintaining their key policies, IAM policies, and grants
- Enabling and disabling them
- Rotating their cryptographic material
- Adding tags
- Creating aliases that refer to them
- Scheduling them for deletion

You can also use CloudTrail and Amazon CloudWatch Logs to track the requests that AWS IoT SiteWise sends to AWS KMS on your behalf.

If you're using customer managed keys, you need to grant AWS IoT SiteWise access to the KMS key stored in your account. AWS IoT SiteWise uses envelope encryption and key hierarchy to encrypt data. Your AWS KMS encryption key is used to encrypt the root key of this key hierarchy. For more information, see Envelope encryption in the AWS Key Management Service Developer Guide.

The following example policy grants AWS IoT SiteWise permissions to a create customer managed key on your behalf. When you create your key, you need to allow the kms:CreateGrant and kms:DescribeKey actions.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1603902045292",
      "Action": [
        "kms:CreateGrant",
        "kms:DescribeKey"
      ],
      "Effect": "Allow",
    }
  ]
}
```
The encryption context for your created grant uses your `aws:iotsitewise:subscriberId` and account ID.

**Data at rest on gateways**

AWS IoT SiteWise gateways store the following data on the local file system:

- OPC-UA source configuration information
- The set of OPC-UA data stream paths from connected OPC-UA sources
- Industrial data cached when the gateway loses connection to the internet

AWS IoT SiteWise gateways run on AWS IoT Greengrass. AWS IoT Greengrass relies on Unix file permissions and full-disk encryption (if enabled) to protect data at rest on the core. It's your responsibility to secure the file system and device.

However, AWS IoT Greengrass does encrypt local copies of your OPC-UA server secrets retrieved from Secrets Manager. For more information, see Secrets encryption in the *AWS IoT Greengrass Version 1 Developer Guide*.

For more information about encryption at rest on AWS IoT Greengrass cores, see Encryption at rest in the *AWS IoT Greengrass Version 1 Developer Guide*.

**Encryption in transit**

AWS IoT SiteWise has three modes of communication where data is in transit:

- **Over the internet (p. 323)** – Communication between local devices (including gateways) and AWS IoT SiteWise is encrypted.
- **Over the local network (p. 323)** – Communication between OpsHub for SiteWise application and gateways is always encrypted. Communication between the SiteWise monitor application running within your browser and gateways is always encrypted. Communication between gateways and OPC-UA sources can be encrypted.
- **Between components on gateways (p. 324)** – Communication between AWS IoT Greengrass components on AWS IoT SiteWise gateways isn't encrypted.

**Data in transit over the internet**

AWS IoT SiteWise uses Transport Layer Security (TLS) to encrypt all communication over the internet. All data sent to the AWS Cloud is sent over a TLS connection using MQTT or HTTPS protocols, so it's secure by default. Gateways, which run on AWS IoT Greengrass, and property value notifications use the AWS IoT transport security model. For more information, see Transport security in the *AWS IoT Developer Guide*.

**Data in transit over the local network**

AWS IoT SiteWise gateways follow OPC-UA specifications for communication with local OPC-UA sources. It's your responsibility to configure your sources to use a message security mode that encrypts data in transit.

If you choose a `sign` message security mode, data in transit between gateways and sources is signed but not encrypted. If you choose a `sign and encrypt` message security mode, the data in transit between...
gateways and sources is signed and encrypted. For more information about configuring sources, see Configuring data sources (p. 102).

The communication between the edge console application and gateways is always encrypted by TLS. The Sitewise Edge connector on the gateway generates and stores a self-signed certificate to be able to establish a TLS connection with the edge console for AWS IoT SiteWise application. You will need to copy this certificate from your gateway to the edge console for AWS IoT SiteWise application before you connect the application to the gateway. This ensures that the edge console for AWS IoT SiteWise application is able to verify that it has connected to your trusted gateway.

In addition to TLS for secrecy and server authenticity, Sitewise Edge uses the SigV4 protocol to establish the authenticity of the edge console application. The Sitewise Edge connector on the gateway accepts and stores a password to be able to verify incoming connections from the edge console application, SiteWise Monitor application running within browsers, and other clients based on the AWS IoT SiteWise SDK.

For more information about generating the password and server certificate, see the section called “Managing gateways” (p. 134).

Data in transit between local components on gateways

AWS IoT SiteWise gateways run on AWS IoT Greengrass, which doesn’t encrypt data exchanged locally on the AWS IoT Greengrass core because the data doesn’t leave the device. This includes communication between AWS IoT Greengrass components such as the AWS IoT SiteWise connector. For more information, see Data on the core device in the AWS IoT Greengrass Version 1 Developer Guide.

Key management

AWS IoT SiteWise cloud key key management

By default, AWS IoT SiteWise uses AWS managed keys to protect your data in the AWS Cloud. You can update your settings to use a customer managed key to encrypt some data in AWS IoT SiteWise. You can create, manage, and view your encryption key through AWS Key Management Service (AWS KMS).

AWS IoT SiteWise supports server-side encryption with customer managed keys stored in AWS KMS to encrypt the following data:

- Asset property values
- Aggregate values

Note
Other data and resources are encrypted using the default encryption with keys managed by AWS IoT SiteWise. This key is stored in the AWS IoT SiteWise account.

For more information, see What is AWS Key Management Service? in the AWS Key Management Service Developer Guide.

Enable encryption using customer managed keys

To use customer managed keys with AWS IoT SiteWise, you need to update your AWS IoT SiteWise settings.

To enable encryption using KMS keys

1. Navigate to the AWS IoT SiteWise console.
2. Choose Account Settings and choose Edit to open the Edit account settings page.
3. For **Encryption key type**, choose **Choose a different AWS KMS key**. This enables encryption with customer managed keys stored in AWS KMS.
   
   **Note**
   Currently, you can only use customer managed key encryption for asset property values and aggregate values.

4. Choose your KMS key with one of the following options:
   
   - **To use an existing KMS key** – Choose your KMS key alias from the list.
   - **To create a new KMS key** – Choose **Create an AWS KMS key**.
   
   **Note**
   This opens the AWS KMS dashboard. For more information about creating a KMS key, see [Creating keys](https://docs.aws.amazon.com/kms/latest/DeveloperGuide/creating-keys.html) in the *AWS Key Management Service Developer Guide*.

5. Choose **Save** to update your settings.

### AWS IoT Greengrass gateway key management

AWS IoT SiteWise gateways run on AWS IoT Greengrass, and AWS IoT Greengrass core devices use public and private keys to authenticate with the AWS Cloud and encrypt local secrets, such as OPC-UA authentication secrets. For more information, see [Key management](https://docs.aws.amazon.com/greengrass/latest/developerguide/security-key-management.html) in the *AWS IoT Greengrass Version 1 Developer Guide*.

### Internetwork traffic privacy

Connections between AWS IoT SiteWise and on-premises applications, such as gateways, are secured over Transport Layer Security (TLS) connections. For more information, see [Encryption in transit](p. 323).

AWS IoT SiteWise doesn’t support connections between Availability Zones within an AWS Region or connections between AWS accounts.

You can configure AWS SSO in only one Region at a time. SiteWise Monitor connects to the Region that you configured for AWS SSO. This means that you use one Region for AWS SSO access, but you can create portals in any Region.

### Identity and access management for AWS IoT SiteWise

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 SiteWise resources. IAM is an AWS service that you can use with no additional charge.

**Topics**

- **Audience** (p. 326)
- **Authenticating with identities** (p. 326)
- **Managing access using policies** (p. 328)
- **How AWS IoT SiteWise works with IAM** (p. 329)
- **AWS IoT SiteWise identity-based policy examples** (p. 336)
- **Using service-linked roles for AWS IoT SiteWise** (p. 339)
- **Using service roles for AWS IoT SiteWise Monitor** (p. 341)
Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS IoT SiteWise.

**Service user** – If you use the AWS IoT SiteWise service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT SiteWise 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 SiteWise, see Troubleshooting AWS IoT SiteWise identity and access (p. 351).

**Service administrator** – If you're in charge of AWS IoT SiteWise resources at your company, you probably have full access to AWS IoT SiteWise. It's your job to determine which AWS IoT SiteWise features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT SiteWise, see How AWS IoT SiteWise works with IAM (p. 329).

**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 SiteWise. To view example AWS IoT SiteWise identity-based policies that you can use in IAM, see AWS IoT SiteWise identity-based policy examples (p. 336).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.

IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for AWS IoT SiteWise in the Service Authorization Reference.
- **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...
Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the IAM User Guide.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.
Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

**Access control lists (ACLs)**

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.

**Other policy types**

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of the entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the **Principal** field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.

**Multiple policy types**

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

**How AWS IoT SiteWise works with IAM**

Before you use IAM to manage access to AWS IoT SiteWise, you should understand what IAM features are available to use with AWS IoT SiteWise.

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>Supported by AWS IoT SiteWise?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies with resource-level permissions (p. 330)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
To get a high-level view of how AWS IoT SiteWise and other AWS services work with IAM, see AWS services that work with IAM in the IAM User Guide.

Contents

- AWS IoT SiteWise identity-based policies (p. 330)
  - Actions (p. 330)
    - BatchPutAssetPropertyValue authorization (p. 331)
  - Resources (p. 331)
  - Condition keys (p. 332)
  - Examples (p. 334)
- AWS IoT SiteWise resource-based policies (p. 335)
- Access control lists (ACLs) (p. 335)
- Authorization based on AWS IoT SiteWise tags (p. 335)
- AWS IoT SiteWise IAM roles (p. 335)
  - Using temporary credentials with AWS IoT SiteWise (p. 335)
  - Service-linked roles (p. 335)
  - Service roles (p. 336)
  - Choosing an IAM role in AWS IoT SiteWise (p. 336)

AWS IoT SiteWise identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. AWS IoT SiteWise supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the IAM User Guide.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don’t have a matching API operation. There are also

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>Supported by AWS IoT SiteWise?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource-based policies (p. 335)</td>
<td>No</td>
</tr>
<tr>
<td>Access control lists (ACLs) (p. 335)</td>
<td>No</td>
</tr>
<tr>
<td>Tags-based authorization (p. 335)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials (p. 335)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles (p. 335)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles (p. 335)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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 SiteWise use the following prefix before the action: iotsitewise:. For example, to grant someone permission to upload asset property data to AWS IoT SiteWise with the BatchPutAssetPropertyValue API operation, you include the iotsitewise:BatchPutAssetPropertyValue action in their policy. Policy statements must include either an Action or NotAction element. AWS IoT SiteWise 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": [  "iotsitewise:action1",  "iotsitewise:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action.

```
"Action": "iotsitewise:Describe*"
```

To see a list of AWS IoT SiteWise actions, see Actions Defined by AWS IoT SiteWise in the IAM User Guide.

**BatchPutAssetPropertyValue** authorization

AWS IoT SiteWise authorizes access to the BatchPutAssetPropertyValue action in an unusual way. For most actions, when you allow or deny access to an action, that action returns an error if permissions aren't granted. When you use BatchPutAssetPropertyValue, you can send multiple data entries to different assets and asset properties in a single API request, and AWS IoT SiteWise authorizes each data entry independently. For any individual entry that fails authorization in the request, AWS IoT SiteWise includes an AccessDeniedException in the returned list of errors. AWS IoT SiteWise receives the data for any entry that authorizes and succeeds, even if another entry in the same request fails.

**Important**

If one entry is denied permissions, all entries for the same asset are also denied. For example, consider a scenario where you allow access to a property Property1 for any asset using the propertyId condition key. If you send a BatchPutAssetPropertyValue request that contains entries for Asset1.Property1, Asset1.Property2, Asset2.Property1, and Asset3.Property3, then the only entry that succeeds is Asset2.Property1. If you send those entries in separate BatchPutAssetPropertyValue requests, then Asset1.Property1 and Asset2.Property1 succeed.

**Resources**

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as resource-level permissions.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.
Each IAM policy statement applies to the resources that you specify using their ARNs. An ARN has the following general syntax.

`arn:${Partition}:${Service}:${Region}:${Account}:${ResourceType}/${ResourcePath}`

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS service namespaces.

For example, to specify the asset with ID `a1b2c3d4-5678-90ab-cdef-22222EXAMPLE` in your statement, use the following ARN:

```
"Resource": "arn:aws:iotsitewise:region:123456789012:asset/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE"
```

To specify all assets that belong to a specific account, use the wildcard (`*`):

```
"Resource": "arn:aws:iotsitewise:region:123456789012:asset/*"
```

Some AWS IoT SiteWise actions, such as those for creating resources, can't be performed on a specific resource. In those cases, you must use the wildcard (`*`).

```
"Resource": "*
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
  "resource1",
  "resource2"
]
```

To see a list of AWS IoT SiteWise resource types and their ARNs, see Resources Defined by AWS IoT SiteWise in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS IoT SiteWise.

### Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the IAM User Guide.
AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

**Important**

Many condition keys are specific to a resource, and some API actions use multiple resources. If you write a policy statement with a condition key, use the Resource element of the statement to specify the resource to which the condition key applies. If you don't do so, the policy might prevent users from performing the action at all, because the condition check fails for the resources to which the condition key doesn't apply. If you don't want to specify a resource, or if you've written the Action element of your policy to include multiple API actions, then you must use the ...IfExists condition type to ensure that the condition key is ignored for resources that don't use it. For more information, see ...IfExists conditions in the IAM User Guide.

AWS IoT SiteWise defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

**AWS IoT SiteWise condition keys**

<table>
<thead>
<tr>
<th>Condition key</th>
<th>Description</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotsitewise:assetHierarchyPath</td>
<td>The asset's hierarchy path, which is a string of asset IDs each separated by a forward slash. Use this condition key to define permissions based on a subset of your hierarchy of all assets in your account. Example value: /a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/a1b2c3d4-5678-90ab-cdef-66666EXAMPLE</td>
<td>String</td>
</tr>
<tr>
<td>iotsitewise:propertyId</td>
<td>The ID of an asset property. Use this condition key to define permissions based on a specified property of an asset model. This condition key applies to all assets of that model. Example value: a1b2c3d4-5678-90ab-cdef-33333EXAMPLE</td>
<td>String</td>
</tr>
<tr>
<td>iotsitewise:childAssetId</td>
<td>The ID of an asset being associated as a child to another asset. Use this condition key to define permissions based on child assets. To define permissions based on parent assets, use the resource section of a policy statement. Example value: a1b2c3d4-5678-90ab-cdef-66666EXAMPLE</td>
<td>String</td>
</tr>
<tr>
<td>iotsitewise:iamArn</td>
<td>The ARN of an IAM identity when listing access policies.</td>
<td>String, Null</td>
</tr>
<tr>
<td>Condition key</td>
<td>Description</td>
<td>Types</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>iotsitewise:user</td>
<td>The ID of an AWS SSO user when listing access policies. Use this condition key to define access policy permissions for an AWS SSO user.</td>
<td>String, Null</td>
</tr>
<tr>
<td></td>
<td>Example value: a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE</td>
<td></td>
</tr>
<tr>
<td>iotsitewise:group</td>
<td>The ID of an AWS SSO group when listing access policies. Use this condition key to define access policy permissions for an AWS SSO group.</td>
<td>String, Null</td>
</tr>
<tr>
<td></td>
<td>Example value: a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE</td>
<td></td>
</tr>
<tr>
<td>iotsitewise:portal</td>
<td>The ID of a portal in an access policy. Use this condition key to define access policy permissions based on a portal.</td>
<td>String, Null</td>
</tr>
<tr>
<td></td>
<td>Example value: a1b2c3d4-5678-90ab-cdef-77777EXAMPLE</td>
<td></td>
</tr>
<tr>
<td>iotsitewise:project</td>
<td>The ID of a project in an access policy, or the ID of a project for a dashboard. Use this condition key to define dashboard or access policy permissions based on a project.</td>
<td>String, Null</td>
</tr>
<tr>
<td></td>
<td>Example value: a1b2c3d4-5678-90ab-cdef-88888EXAMPLE</td>
<td></td>
</tr>
</tbody>
</table>

To learn with which actions and resources you can use a condition key, see Actions Defined by AWS IoT SiteWise.

**Examples**

To view examples of AWS IoT SiteWise identity-based policies, see AWS IoT SiteWise identity-based policy examples (p. 336).
AWS IoT SiteWise resource-based policies

AWS IoT SiteWise doesn't support resource-based policies (p. 328).

Access control lists (ACLs)

AWS IoT SiteWise doesn't support ACLs (p. 329).

Authorization based on AWS IoT SiteWise tags

You can attach tags to AWS IoT SiteWise resources or pass tags in a request to AWS IoT SiteWise. To control access based on tags, you provide tag information in the condition element of a policy using the iotsitewise:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. For more information about tagging AWS IoT SiteWise resources, see Tagging your AWS IoT SiteWise resources (p. 369).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Viewing AWS IoT SiteWise assets based on tags (p. 338).

AWS IoT SiteWise IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with AWS IoT SiteWise

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

AWS IoT SiteWise supports using temporary credentials.

SiteWise Monitor supports federated users to access portals. Portal users authenticate with their AWS SSO or IAM credentials.

Important

IAM users or roles must have the iotsitewise:DescribePortal permission to sign in to the portal.

When a user signs in to a portal, SiteWise Monitor generates a session policy that provides the following permissions:

- Read-only access to the assets and asset data in AWS IoT SiteWise in your account to which that portal's role provides access.
- Access to projects in that portal to which the user has administrator (project owner) or read-only (project viewer) access.

For more information about federated portal user permissions, see Using service roles for AWS IoT SiteWise Monitor (p. 341).

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

AWS IoT SiteWise supports service-linked roles. For details about creating or managing AWS IoT SiteWise service-linked roles, see Using service-linked roles for AWS IoT SiteWise (p. 339).
Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

AWS IoT SiteWise uses a service role to allow SiteWise Monitor portal users to access some of your AWS IoT SiteWise resources on your behalf. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 341).

You must have required permissions before you can create AWS IoT Events alarm models in AWS IoT SiteWise. For more information, see Setting up permissions for AWS IoT Events alarms (p. 347).

Choosing an IAM role in AWS IoT SiteWise

When you create a portal resource in AWS IoT SiteWise, you must choose a role to allow the federated users of your SiteWise Monitor portal to access AWS IoT SiteWise on your behalf. If you have previously created a service role, then AWS IoT SiteWise provides you with a list of roles to choose from. Otherwise, you can create a role with the required permissions when you create a portal. It's important to choose a role that allows access to your assets and asset data. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 341).

AWS IoT SiteWise identity-based policy examples

By default, IAM users and roles don’t have permission to create or modify AWS IoT SiteWise 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 (p. 336)
- Using the AWS IoT SiteWise console (p. 337)
- Allowing users to view their own permissions (p. 337)
- Allowing users to ingest data to assets in one hierarchy (p. 338)
- Viewing AWS IoT SiteWise assets based on tags (p. 338)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS IoT SiteWise resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- Get started using AWS managed policies – To start using AWS IoT SiteWise quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.
- Grant least privilege – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.
• **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

• **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

**Using the AWS IoT SiteWise console**

To access the AWS IoT SiteWise console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT SiteWise 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 SiteWise console, attach the AWSIoTSiteWiseConsoleFullAccess managed policy to those entities or define equivalent permissions for those entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

**Allowing 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.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"],
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*
    }
  ]
}
```
### Allowing users to ingest data to assets in one hierarchy

In this example, you want to grant an IAM user in your AWS account access to write data to all asset properties in a specific hierarchy of assets, starting from the root asset `a1b2c3d4-5678-90ab-cdef-22222EXAMPLE`. The policy grants the `iotsitewise:BatchPutAssetPropertyValue` permission to the user. This policy uses the `iotsitewise:assetHierarchyPath` condition key to restrict access to assets whose hierarchy path matches the asset or its descendants.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PutAssetPropertyValuesForHierarchy",
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "arn:aws:iotsitewise:*:*:asset/*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
                        "/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/*"
                    ]
                }
            }
        }
    ]
}
```

### Viewing AWS IoT SiteWise assets based on tags

You can use conditions in your identity-based policy to control access to AWS IoT SiteWise resources based on tags. This example shows how you might create a policy that allows viewing an asset. However, permission is granted only if the asset 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.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListAllAssets",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:ListAssets",
                "iotsitewise:ListAssociatedAssets"
            ],
            "Resource": "*
        },
        {
            "Sid": "DescribeAssetIfOwner",
            "Effect": "Allow",
            "Action": "iotsitewise:DescribeAsset",
            "Resource": "arn:aws:iotsitewise::*:*:asset/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```
You can attach this policy to the IAM users in your account. If a user named richard-roe attempts to view an AWS IoT SiteWise asset, the asset must be tagged Owner=richard-roe or owner=richard-roe. Otherwise, he is denied access. The condition tag key Owner matches both Owner and owner because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the IAM User Guide.

Using service-linked roles for AWS IoT SiteWise

AWS IoT SiteWise uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS IoT SiteWise. Service-linked roles are predefined by AWS IoT SiteWise and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS IoT SiteWise easier because you don't have to manually add the necessary permissions. AWS IoT SiteWise defines the permissions of its service-linked roles, and unless defined otherwise, only AWS IoT SiteWise can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy can't be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS IoT SiteWise resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS IoT SiteWise

AWS IoT SiteWise uses the service-linked role named AWSServiceRoleForIoTSiteWise – AWS IoT SiteWise uses this service-linked role to deploy gateways (which run on AWS IoT Greengrass) and perform logging.

The AWSServiceRoleForIoTSiteWise service-linked role trusts the following services to assume the role:

- iotsitewise.amazonaws.com

The role uses the following permissions policy to allow AWS IoT SiteWise to complete actions on other services' resources in your account:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:GetAssociatedRole",
        "greengrass:GetCoreDefinition",
        "greengrass:GetCoreDefinitionVersion",
        "greengrass:GetGroup",
        "greengrass:GetGroupVersion",
        "greengrass:CreateDeployment",
        "greengrass:ListDeployments",
        "greengrass:ListEffectiveDeployments",
        "greengrass:ListInstalledComponents",
        "iot:DescribeThing",
        "iot:CancelJob",
        "iot:CreateJob",
```
You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the IAM User Guide.

Creating a service-linked role for AWS IoT SiteWise

You don't need to manually create a service-linked role. When you perform any operation in the AWS IoT SiteWise console, AWS IoT SiteWise creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you perform any operation in the AWS IoT SiteWise console, AWS IoT SiteWise creates the service-linked role for you again.

You can also use the IAM console or API to create a service-linked role for AWS IoT SiteWise.

- To do so in the IAM console, create a role with the AWSServiceRoleForIoTSiteWise policy and a trust relationship with iotsitewise.amazonaws.com.
- To do so using the AWS CLI or IAM API, create a role with the arn:aws:iam::aws:policy/ aws-service-role/AWSServiceRoleForIoTSiteWise policy and a trust relationship with iotsitewise.amazonaws.com.
Important
The name of your service-linked role must be `AWSServiceRoleForIoTSiteWise`.

For more information, see Creating a service-linked role in the IAM User Guide.

If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for AWS IoT SiteWise

AWS IoT SiteWise doesn’t allow you to edit the `AWSServiceRoleForIoTSiteWise` service-linked role. After you create a service-linked role, you can’t change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the IAM User Guide.

Deleting a service-linked role for AWS IoT SiteWise

If you no longer need a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that isn’t actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note
If the AWS IoT SiteWise service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try again.

To delete AWS IoT SiteWise resources used by the `AWSServiceRoleForIoTSiteWise`

1. Disable logging for AWS IoT SiteWise. For more information, see Changing your logging level (console) (p. 361) or Changing your logging level (CLI) (p. 362).
2. Delete any active gateways.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForIoTSiteWise` service-linked role. For more information, see Deleting a Service-Linked Role in the IAM User Guide.

Supported Regions for AWS IoT SiteWise service-linked roles

AWS IoT SiteWise supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS IoT SiteWise Endpoints and Quotas.

Using service roles for AWS IoT SiteWise Monitor

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.

To allow federated SiteWise Monitor portal users to access your AWS IoT SiteWise and AWS Single Sign-On resources, you must attach a service role to each portal that you create. The service role must specify SiteWise Monitor as a trusted entity and include the `AWSIoTSiteWiseMonitorPortalAccess` managed policy or define equivalent permissions (p. 342). This policy is maintained by AWS and defines the set of permissions that SiteWise Monitor uses to access your AWS IoT SiteWise and AWS SSO resources.

When you create a SiteWise Monitor portal, you must choose a role that allows users of that portal to access your AWS IoT SiteWise and AWS SSO resources. The AWS IoT SiteWise console can create and configure the role for you. You can edit the role in IAM later. Your portal users will have issues using their SiteWise Monitor portals if you remove the required permissions from the role or delete the role.
Note
Portals created before April 29, 2020 didn’t require service roles. If you created portals before this date, you must attach service roles to continue using them. To do so, navigate to the Portals page in the AWS IoT SiteWise console, and then choose Migrate all portals to use IAM roles.

The following sections describe how to create and manage the SiteWise Monitor service role in the AWS Management Console or the AWS Command Line Interface.

Contents
• Service role permissions for SiteWise Monitor (p. 342)
• Managing the SiteWise Monitor service role (console) (p. 344)
  • Finding a portal's service role (console) (p. 344)
  • Creating a SiteWise Monitor service role (AWS IoT SiteWise console) (p. 344)
  • Creating a SiteWise Monitor service role (IAM console) (p. 345)
  • Changing a portal's service role (console) (p. 345)
• Managing the SiteWise Monitor service role (CLI) (p. 345)
  • Finding a portal's service role (CLI) (p. 346)
  • Creating the SiteWise Monitor service role (CLI) (p. 346)

Service role permissions for SiteWise Monitor

When you create a portal, AWS IoT SiteWise lets you create a role whose name starts with AWSIoTSiteWiseMonitorServiceRole. This role allows federated SiteWise Monitor users to access your portal configuration, assets, asset data, and AWS SSO configuration.

The role trusts the following service to assume the role:

• monitor.iotsitewise.amazonaws.com

The role uses the following permissions policy, whose name starts with AWSIoTSiteWiseMonitorServicePortalPolicy, to allow SiteWise Monitor users to complete actions on resources in your account. The AWSIoTSiteWiseMonitorPortalAccess managed policy defines equivalent permissions.

```json
{
   "Version": "2012-10-17",
   "Statement": [
   {
      "Effect": "Allow",
   ...
```
"iotsitewise:DescribeAccessPolicy",
"iotsitewise:UpdateAccessPolicy",
"iotsitewise:DeleteAccessPolicy",
"iotsitewise:ListAccessPolicies",
"iotsitewise:DescribeAsset",
"iotsitewise:ListAssets",
"iotsitewise:ListAssociatedAssets",
"iotsitewise:DescribeAssetProperty",
"iotsitewise:GetAssetPropertyValue",
"iotsitewise:GetAssetPropertyValueHistory",
"iotsitewise:GetAssetPropertyAggregates",
"iotsitewise:BatchPutAssetPropertyValue",
"iotsitewise:ListAssetRelationships",
"iotsitewise:DescribeAssetModel",
"iotsitewise:ListAssetModels",
"iotsitewise:UpdateAssetModel",
"iotsitewise:UpdateAssetModelPropertyRouting",
"sso-directory:DescribeUsers",
"sso-directory:DescribeUser",
"iotevents:DescribeAlarmModel",
"iotevents:ListTagsForResource",

"Effect": "Allow",
"Action": [
  "iotevents:BatchAcknowledgeAlarm",
  "iotevents:BatchSnoozeAlarm",
  "iotevents:BatchEnableAlarm",
  "iotevents:BatchDisableAlarm"
],
"Resource": "*",
"Condition": { "Null": { "iotevents:keyValue": "false" }}
},

"Effect": "Allow",
"Action": [
  "iotevents:CreateAlarmModel",
  "iotevents:TagResource"
],
"Resource": "*",
"Condition": { "Null": { "aws:RequestTag/iotsitewisemonitor": "false" }}
},

"Effect": "Allow",
"Action": [
  "iotevents:UpdateAlarmModel",
  "iotevents:DeleteAlarmModel"
],
"Resource": "*",
"Condition": { "Null": { "aws:ResourceTag/iotsitewisemonitor": "false" }}
}
]
Using service roles for SiteWise Monitor

```
"Effect": "Allow",
"Action": [
  "iam:PassRole"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "iam:PassedToService": [
      "iotevents.amazonaws.com"
    ]
  }
}
```

For more information about the required permissions for alarms, see Setting up permissions for AWS IoT Events alarms (p. 347).

When a portal user signs in, SiteWise Monitor creates a session policy based on the intersection of the service role and that user's access policies. Access policies define identities' level of access to your portals and projects. For more information about portal permissions and access policies, see Administering your SiteWise Monitor portals (p. 273) and CreateAccessPolicy.

Managing the SiteWise Monitor service role (console)

You can use the AWS IoT SiteWise console to easily manage the SiteWise Monitor service role for your portals. When you create a portal, the console checks if you have any existing roles that can be attached to that portal. If not, the console can create and configure a service role for you. For more information, see Creating a portal (p. 261).

Topics
- Finding a portal's service role (console) (p. 344)
- Creating a SiteWise Monitor service role (AWS IoT SiteWise console) (p. 344)
- Creating a SiteWise Monitor service role (IAM console) (p. 345)
- Changing a portal's service role (console) (p. 345)

Finding a portal's service role (console)

Use the following steps to find the service role attached to a SiteWise Monitor portal.

To find a portal's service role

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Portals.
3. Choose the portal for which you want to find the service role.

   The role attached to the portal appears under Permissions, Service role.

Creating a SiteWise Monitor service role (AWS IoT SiteWise console)

When you create a SiteWise Monitor portal, you can create a service role for your portal. For more information, see Creating a portal (p. 261).

You can also create a service role for an existing portal in the AWS IoT SiteWise console. This replaces the portal’s existing service role.
To create a service role for an existing portal

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Portals.
3. Choose the portal for which you want to create a new service role.
5. Under Permissions, choose Create and use a new service role from the list.
6. Enter a name for your new role.
7. Choose Save.

Creating a SiteWise Monitor service role (IAM console)

You can create a service role from the service role template in the IAM console. This role template includes the AWSIoTSiteWiseMonitorPortalAccess managed policy and specifies SiteWise Monitor as a trusted entity.

To create a service role from the portal service role template

1. Navigate to the IAM console.
2. In the navigation pane, choose Roles.
3. Choose Create role.
4. In Choose a use case, choose IoT SiteWise.
5. In Select your use case, choose IoT SiteWise Monitor - Portal.
6. Choose Next: Permissions.
7. Choose Next: Tags.
8. Choose Next: Review.
9. Enter a Role name for the new service role.
10. Choose Create role.

Changing a portal's service role (console)

Use the following procedure to choose a different SiteWise Monitor service role for a portal.

To change a portal's service role

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose Portals.
3. Choose the portal for which you want to change the service role.
5. Under Permissions, choose Use an existing role.
6. Choose an existing role to attach to this portal.
7. Choose Save.

Managing the SiteWise Monitor service role (CLI)

You can use the AWS CLI for the following portal service role management tasks:

Topics
- Finding a portal's service role (CLI) (p. 346)
- Creating the SiteWise Monitor service role (CLI) (p. 346)
Finding a portal's service role (CLI)

To find the service role attached to a SiteWise Monitor portal, run the following command to list all of your portals in the current Region.

```bash
aws iotsitewise list-portals
```

The operation returns a response that contains your portal summaries in the following format.

```json
{
   "portalSummaries": [
      {
         "id": "a1b2c3d4-5678-90ab-cdef-aaaaEXAMPLE",
         "name": "WindFarmPortal",
         "description": "A portal that contains wind farm projects for Example Corp.",
         "roleArn": "arn:aws:iam::123456789012:role/service-role/role-name",
         "startUrl": "https://a1b2c3d4-5678-90ab-cdef-aaaaEXAMPLE.app.iotsitewise.aws",
         "creationDate": "2020-02-04T23:01:52.90248068Z",
         "lastUpdateDate": "2020-02-04T23:01:52.90248078Z"
      }
   ]
}
```

You can also use the `DescribePortal` operation to find your portal's role if you know the ID of your portal.

Creating the SiteWise Monitor service role (CLI)

Use the following steps to create a new SiteWise Monitor service role.

**To create a SiteWise Monitor service role**

1. Create a role with a trust policy that allows SiteWise Monitor to assume the role. This example creates a role named `MySiteWiseMonitorPortalRole` from a trust policy stored in a JSON string.

   **Linux, macOS, or Unix**
   ```bash
   aws iam create-role --role-name MySiteWiseMonitorPortalRole --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [ {
         "Effect": "Allow",
         "Principal": { "Service": "monitor.iotsitewise.amazonaws.com" },
         "Action": "sts:AssumeRole"
      } ]
   }
   ```

   **Windows command prompt**
   ```bash
   ```

2. Copy the role ARN from the role metadata in the output. When you create a portal, you use this ARN to associate the role with your portal. For more information about creating a portal, see `CreatePortal` in the *AWS IoT SiteWise API Reference*. 

346
3. Attach the AWSIoTSiteWiseMonitorPortalAccess policy to the role, or attach a policy that defines equivalent permissions.

```
```

To attach a service role to an existing portal

1. To retrieve the portal's existing details, run the following command. Replace `portal-id` with the ID of the portal.

```
aws iotsitewise describe-portal --portal-id portal-id
```

The operation returns a response that contains the portal's details in the following format.

```json
{
    "portalId": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalArn": "arn:aws:iotsitewise:region:account-id:portal/a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalName": "WindFarmPortal",
    "portalDescription": "A portal that contains wind farm projects for Example Corp.",
    "portalClientId": "E-1a2b3c4d5e6f_sn6tbqHVZvLWVEXAMPLE",
    "portalStartUrl": "https://a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE.app.iotsitewise.aws",
    "portalContactEmail": "support@example.com",
    "portalStatus": {
        "state": "ACTIVE"
    },
    "portalCreationDate": "2020-04-29T23:01:52.90248068Z",
    "portalLastUpdateDate": "2020-04-29T00:28:26.103548287Z",
    "roleArn": "arn:aws:iam::123456789012:role/service-role/AWSIoTSiteWiseMonitorServiceRole_1aEXAMPLE"
}
```

2. To attach a service role to a portal, run the following command. Replace `role-arn` with the service role ARN, and replace the remaining parameters with the portal's existing values.

```
aws iotsitewise update-portal \
    --portal-id portal-id \
    --role-arn role-arn \
    --portal-name portal-name \
    --portal-description portal-description \
    --portal-contact-email portal-contact-email
```

Setting up permissions for AWS IoT Events alarms

When you use an AWS IoT Events alarm model to monitor an AWS IoT SiteWise asset property, you must have the following IAM permissions:

- An AWS IoT Events service role that allows AWS IoT Events to send data to AWS IoT SiteWise. For more information, see Identity and access management for AWS IoT Events in the AWS IoT Events Developer Guide.
- You must have the following AWS IoT SiteWise action permissions: iotsitewise:DescribeAssetModel and iotsitewise:UpdateAssetModelPropertyRouting. These permissions allow AWS IoT SiteWise to send asset property values to AWS IoT Events alarm models.
For more information, see Resource-based policies in the IAM User Guide.

Required action permissions

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.

Before you define an AWS IoT Events alarm model, you must grant the following permissions that allow AWS IoT SiteWise to send asset property values to the alarm model.

- iotsitewise:DescribeAssetModel – Allows AWS IoT Events to check if an asset property exists.
- iotsitewise:UpdateAssetModelPropertyRouting – Allows AWS IoT SiteWise to automatically create subscriptions that enable AWS IoT SiteWise to send data to AWS IoT Events.

For more information about AWS IoT SiteWise supported actions, see Actions defined by AWS IoT SiteWise in the Service Authorization Reference.

Example Example permissions policy 1

The following policy allows AWS IoT SiteWise to send asset property values to any AWS IoT Events alarm models.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotevents:CreateAlarmModel",
        "iotevents:UpdateAlarmModel"
      ],
      "Resource": "arn:aws:iotevents:us-east-1:123456789012:alarmModel/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:UpdateAssetModelPropertyRouting"
      ],
      "Resource": "arn:aws:iotsitewise:us-east-1:123456789012:asset-model/*"
    }
  ]
}
```

Example Example permissions policy 2

The following policy allows AWS IoT SiteWise to send values of a specified asset property to a specified AWS IoT Events alarm model.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotevents:CreateAlarmModel",
        "iotevents:UpdateAlarmModel"
      ],
      "Resource": "arn:aws:iotevents:us-east-1:123456789012:alarmModel/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:UpdateAssetModelPropertyRouting"
      ],
      "Resource": "arn:aws:iotsitewise:us-east-1:123456789012:asset-model/*"
    }
  ]
}
```
Setting up permissions for alarms

```json
{
   "Effect": "Allow",
   "Action": [
      "iotsitewise:DescribeAssetModel"
   ],
   "Resource": "arn:aws:iotsitewise:us-east-1:123456789012:asset-model/*"
},
{
   "Effect": "Allow",
   "Action": [
      "iotsitewise:UpdateAssetModelPropertyRouting"
   ],
   "Resource": [
      "arn:aws:iotsitewise:us-east-1:123456789012:asset-model/12345678-90ab-cdef-1234-567890abcdef"
   ],
   "Condition": {
      "StringLike": {
         "iotsitewise:propertyId": "abcdef12-3456-7890-abcd-ef1234567890",
         "iotevents:alarmModelArn": "arn:aws:iotevents:us-east-1:123456789012:alarmModel/MyAlarmModel"
      }
   }
}
```

(Optional) ListInputRoutings permission

When you update or delete an asset model, AWS IoT SiteWise can check if an alarm model in AWS IoT Events is monitoring an asset property associated with this asset model. This prevents you from deleting an asset property that an AWS IoT Events alarm is currently using. To enable this feature in AWS IoT SiteWise, you must have the `iotevents:ListInputRoutings` permission. This permission allows AWS IoT SiteWise to make calls to the ListInputRoutings API operation supported by AWS IoT Events.

**Note**

We strongly recommend that you add the ListInputRoutings permission.

**Example permissions policy**

The following policy allows you to update and delete asset models, and use the ListInputRoutings API in AWS IoT SiteWise.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "iotsitewise:UpdateAssetModel",
            "iotsitewise:DeleteAssetModel",
            "iotevents:ListInputRoutings"
         ],
         "Resource": "arn:aws:iotsitewise:us-east-1:123456789012:asset-model/*"
      }
   ]
}
```

**Required permissions for SiteWise Monitor**

If you want to use the alarms feature in SiteWise Monitor portals, you must update the SiteWise Monitor service role (p. 341) with the following policy:
Setting up permissions for alarms

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotsitewise:DescribePortal",
        "iotsitewise:CreateProject",
        "iotsitewise:DescribeProject",
        "iotsitewise:UpdateProject",
        "iotsitewise:DeleteProject",
        "iotsitewise:ListProjects",
        "iotsitewise:BatchAssociateProjectAssets",
        "iotsitewise:BatchDisassociateProjectAssets",
        "iotsitewise:ListProjectAssets",
        "iotsitewise:CreateDashboard",
        "iotsitewise:DescribeDashboard",
        "iotsitewise:UpdateDashboard",
        "iotsitewise:DeleteDashboard",
        "iotsitewise:ListDashboards",
        "iotsitewise:CreateAccessPolicy",
        "iotsitewise:DescribeAccessPolicy",
        "iotsitewise:UpdateAccessPolicy",
        "iotsitewise:DeleteAccessPolicy",
        "iotsitewise:ListAccessPolicies",
        "iotsitewise:DescribeAsset",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:GetAssetPropertyValueHistory",
        "iotsitewise:GetAssetPropertyAggregates",
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:ListAssetModels",
        "iotsitewise:UpdateAssetModel",
        "iotsitewise:UpdateAssetModelPropertyRouting",
        "sso-directory:DescribeUsers",
        "sso-directory:DescribeUser",
        "iotevents:DescribeAlarmModel",
        "iotevents:ListTagsForResource"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotevents:BatchAcknowledgeAlarm",
        "iotevents:BatchSnoozeAlarm",
        "iotevents:BatchEnableAlarm",
        "iotevents:BatchDisableAlarm"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "iotevents:keyValue": "false"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotevents:CreateAlarmModel",
        "iotevents:TagResource"
      ]
    }
  ]
}
```
Troubleshooting AWS IoT SiteWise identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT SiteWise and IAM.

Topics
- I am not authorized to perform an action in AWS IoT SiteWise (p. 351)
- I am not authorized to perform iam:PassRole (p. 352)
- I want to view my access keys (p. 352)
- I'm an administrator and want to allow others to access AWS IoT SiteWise (p. 352)
- I want to allow people outside of my AWS account to access my AWS IoT SiteWise resources (p. 353)

I am not authorized to perform an action in AWS IoT SiteWise

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 an asset but does not have iotsitewise:DescribeAsset permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iotsitewise:DescribeAsset on resource: a1b2c3d4-5678-90ab-cdef-22222EXAMPLE
```

In this case, Mateo asks his administrator to update his policies to allow him to access the asset resource with ID a1b2c3d4-5678-90ab-cdef-22222EXAMPLE using the iotsitewise:DescribeAsset action.

**I am not authorized to perform iam:PassRole**

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS IoT SiteWise.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in AWS IoT SiteWise. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

**I want to view my access keys**

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtNfEMI/K7MDENG/bPxFfICyEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the IAM User Guide.

**I'm an administrator and want to allow others to access AWS IoT SiteWise**

To allow others to access AWS IoT SiteWise, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS IoT SiteWise.

To get started right away, see Creating your first IAM delegated user and group in the IAM User Guide.
I want to allow people outside of my AWS account to access my AWS IoT SiteWise 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 SiteWise supports these features, see How AWS IoT SiteWise works with IAM (p. 329).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the IAM User Guide.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the IAM User Guide.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.

Compliance validation for AWS IoT SiteWise

AWS IoT SiteWise is not in scope of any AWS compliance programs.

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 SiteWise 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 – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS IoT SiteWise

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency,
Infrastructure security in AWS IoT SiteWise

As a managed service, AWS IoT SiteWise 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 SiteWise 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 by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

AWS IoT SiteWise gateways, which run on AWS IoT Greengrass, use X.509 certificates and cryptographic keys to connect and authenticate to the AWS Cloud. For more information, see Device authentication and authorization for AWS IoT Greengrass in the AWS IoT Greengrass Version 1 Developer Guide.

Configuration and vulnerability analysis in AWS IoT SiteWise

IoT fleets can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make fleet setup complex and error-prone. Because devices are often constrained in computational power, memory, and storage capabilities, this limits the use of encryption and other forms of security on the devices themselves. Also, devices often use software with known vulnerabilities. These factors make IoT fleets an attractive target for hackers and make it difficult to secure your device fleet on an ongoing basis.

AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. You can use AWS IoT Device Defender to analyze, audit, and monitor connected devices to detect abnormal behavior, and mitigate security risks. AWS IoT Device Defender can audit device fleets to ensure they adhere to security best practices and detect abnormal behavior on
devices. This makes it possible to enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised. For more information, see AWS IoT Device Defender in the AWS IoT Developer Guide.

If you use AWS IoT SiteWise gateways to ingest data to the service, it’s your responsibility to configure and maintain your gateway’s environment. This responsibility includes upgrading to the latest versions of the gateway’s system software, AWS IoT Greengrass software, and the AWS IoT SiteWise connector. For more information, see Configure the AWS IoT Greengrass core in the AWS IoT Greengrass Version 1 Developer Guide and Upgrading a connector (p. 129).

AWS IoT SiteWise and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your virtual private cloud (VPC) and AWS IoT SiteWise by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that you can use to privately access AWS IoT SiteWise API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS IoT SiteWise API operations. Traffic between your VPC and AWS IoT SiteWise doesn't leave the AWS network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

Considerations for AWS IoT SiteWise VPC endpoints

Before you set up an interface VPC endpoint for AWS IoT SiteWise, review the Interface endpoint properties and limitations in the Amazon VPC User Guide.

AWS IoT SiteWise supports making calls to the following AWS IoT SiteWise API operations from your VPC:

- For all the data plane API operations, use the following endpoint. Replace region with your AWS Region.

  `data.iotsitewise.region.amazonaws.com`

  The data plane API operations include BatchPutAssetPropertyValue, GetAssetPropertyAggregates, GetAssetPropertyValue, GetAssetPropertyValueHistory and GetInterpolatedAssetPropertyValues.

- For the control plane API operations that you use to manage asset models, assets, gateways, tags, and account configurations, use the following endpoint. Replace region with your AWS Region.

  `api.iotsitewise.region.amazonaws.com`

Creating an interface VPC endpoint for AWS IoT SiteWise

You can create a VPC endpoint for the AWS IoT SiteWise service. You can use either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the Amazon VPC User Guide.

Create a VPC endpoint for AWS IoT SiteWise by using one of the following service names:

- For the data plane API operations, use the following service name:
  
  `com.amazonaws.region.iotsitewise.data`

- For the control plane API operations, use the following service name:
  
  `com.amazonaws.region.iotsitewise.api`

Accessing AWS IoT SiteWise through an interface VPC endpoint

When you create an interface endpoint, we generate endpoint-specific DNS hostnames that you can use to communicate with AWS IoT SiteWise. The private DNS option is enabled by default. For more information, see Using private hosted zones in the Amazon VPC User Guide.

If you enable private DNS for the endpoint, you can make API requests to AWS IoT SiteWise through one of the following VPC endpoints:

- For the data plane API operations, use the following endpoint. Replace `region` with your AWS Region.
  
  `data.iotsitewise.region.amazonaws.com`

- For the control plane API operations, use the following endpoint. Replace `region` with your AWS Region.
  
  `api.iotsitewise.region.amazonaws.com`

If you disable private DNS for the endpoint, you must do the following to access AWS IoT SiteWise through the endpoint:

- Specify the VPC endpoint url in API requests.
- For the data plane API operations, use the following endpoint url. Replace `vpc-endpoint-id` and `region` with your VPC endpoint ID and Region.
Creating a VPC endpoint policy for AWS IoT SiteWise

You can attach an endpoint policy to your VPC endpoint that controls access to AWS IoT SiteWise. The policy specifies the following information:

- The principal that can perform operations.
- The operations that can be performed.
- The resources on which operations can be performed.

For more information, see Controlling access to services with VPC endpoints in the Amazon VPC User Guide.

Example: VPC endpoint policy for AWS IoT SiteWise actions

The following is an example of an endpoint policy for AWS IoT SiteWise. When attached to an endpoint, this policy grants access to the listed AWS IoT SiteWise actions for the IAM user iotsitewiseadmin in AWS account 123456789012 on the specified asset.

```json
{
  // Your policy content goes here
}
```
Security best practices for AWS IoT SiteWise

This topic contains security best practices for AWS IoT SiteWise.

Use authentication credentials on your OPC-UA servers

Require authentication credentials to connect to your OPC-UA servers. Consult the documentation for your servers to do so. Then, to allow your gateway to connect to your OPC-UA servers, add server authentication secrets to your gateway. For more information, see Configuring source authentication (p. 114).

Use encrypted communication modes for your OPC-UA servers

Choose a non-deprecated, encrypted message security mode when you configure your OPC-UA sources for your gateway. This helps secure your industrial data as it moves from your OPC-UA servers to the gateway. For more information, see Data in transit over the local network (p. 323) and Configuring data sources (p. 102).

Keep your components up to date

If you use AWS IoT SiteWise gateways to ingest data to the service, it's your responsibility to configure and maintain your gateway's environment. This responsibility includes upgrading to the latest versions of the gateway's system software, AWS IoT Greengrass software, and connectors.

Note
The AWS IoT SiteWise Edge connector stores secrets on your file system. These secrets control who can view the data cached within your gateway. It's strongly recommended that you turn on disk or file-system encryption for the system running your gateway.

Encrypt your gateway's file system

Encrypt and secure your gateway, so your industrial data is secure as it moves through the gateway. If your gateway has a hardware security module, you can configure AWS IoT Greengrass to secure your


Secure access to your edge configuration

Don't share your edge console application password or your SiteWise Monitor application password. Don't put this password in places where anyone can see them. Implement a healthy password rotation policy by configuring an appropriate expiration for your password.

Grant SiteWise Monitor users minimum possible permissions

Follow the principle of least privilege by using the minimum set of access policy permissions for your portal users.

- When you create a portal, define a role that allows the minimum set of assets needed for that portal. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 341).
- When you and your portal administrators create and share projects, use the minimum set of assets needed for that project.
- When an identity no longer needs access to a portal or project, remove them from that resource. If that identity is no longer applicable to your organization, delete that identity from your identity store.

The least principle best practice also applies to IAM roles. For more information, see Policy best practices (p. 336).

Don't expose sensitive information

You should prevent the logging of credentials and other sensitive information, such as personally identifiable information (PII). We recommend that you implement the following safeguards even though access to local logs on a gateway requires root privileges and access to CloudWatch Logs requires IAM permissions.

- Don't use sensitive information in names, descriptions, or properties of your assets or models.
- Don't use sensitive information in gateway or source names.
- Don't use sensitive information in names or descriptions of your portals, projects, or dashboards.

Follow AWS IoT Greengrass security best practices

Follow AWS IoT Greengrass security best practices for your gateway. For more information, see Security best practices in the AWS IoT Greengrass Version 1 Developer Guide.

See also

- Security best practices in the AWS IoT Developer Guide
- Ten security golden rules for IoT solutions
Monitoring and monitoring in AWS IoT SiteWise

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT SiteWise and your other AWS solutions. AWS IoT SiteWise supports the following monitoring tools to watch the service, report when something is wrong, and take automatic actions when appropriate:

- **Amazon CloudWatch** monitors your AWS resources and the applications that 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](#).

- **Amazon CloudWatch Logs** enables you to monitor, store, and access your log files from AWS IoT SiteWise gateways, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).

- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).

Topics

- Monitoring AWS IoT SiteWise with Amazon CloudWatch Logs (p. 360)
- Monitoring gateway logs (p. 362)
- Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics (p. 365)
- Logging AWS IoT SiteWise API calls with AWS CloudTrail (p. 366)

Monitoring AWS IoT SiteWise with Amazon CloudWatch Logs

You can configure AWS IoT SiteWise to log information to CloudWatch Logs to monitor and troubleshoot the service.

When you use the AWS IoT SiteWise console, AWS IoT SiteWise creates a service-linked role that allows the service to log information on your behalf. If you don't use the AWS IoT SiteWise console, you must create a service-linked role manually to receive logs. For more information, see [Creating a service-linked role for AWS IoT SiteWise](p. 340).

By default, AWS IoT SiteWise doesn't log information to CloudWatch Logs. To enable logging, choose a logging level other than **Disabled** (**OFF**). AWS IoT SiteWise supports the following logging levels:

- **OFF** – Logging is disabled.
- **ERROR** – Errors are logged.
- **INFO** – Errors and informational messages are logged.

You can also configure gateways to log information to CloudWatch Logs through AWS IoT Greengrass. For more information, see [Monitoring gateway logs](p. 362).
Managing logging in AWS IoT SiteWise (console)

You can use the AWS IoT SiteWise console for the following logging configuration tasks.

Topics
- Finding your logging level (console) (p. 361)
- Changing your logging level (console) (p. 361)

Finding your logging level (console)

Use the following procedure to find your current logging level in the AWS IoT SiteWise console.

To find your current AWS IoT SiteWise logging level

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Logging options**.
   
   The current logging status appears under **Logging status**. If logging is enabled, the current logging level appears under **Level of verbosity**.

Changing your logging level (console)

Use the following procedure to change your logging level in the AWS IoT SiteWise console.

To change your AWS IoT SiteWise logging level

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Logging options**.
3. Choose **Edit**.
4. Choose the **Level of verbosity** to enable.
5. Choose **Save**.

Managing logging in AWS IoT SiteWise (CLI)

You can use the AWS Command Line Interface (AWS CLI) for the following logging configuration tasks.
Finding your logging level (CLI)

Run the following command to find your current AWS IoT SiteWise logging level with the AWS CLI.

```
aws iotsitewise describe-logging-options
```

The operation returns a response that contains your logging level in the following format.

```
{
   "loggingOptions": {
      "level": "String"
   }
}
```

Changing your logging level (CLI)

Run the following AWS CLI command to change your AWS IoT SiteWise logging level. Replace `logging-level` with the logging level you want.

```
aws iotsitewise put-logging-options --logging-options level=logging-level
```

Example: AWS IoT SiteWise log file entries

Each AWS IoT SiteWise log entry includes event information and relevant resources for that event, so you can easily understand and analyze log data.

The following example shows a CloudWatch Logs entry that AWS IoT SiteWise logs when you successfully create an asset model.

```
{
   "eventTime": "2020-05-05T00:10:22.902Z",
   "logLevel": "INFO",
   "eventType": "AssetModelCreationSuccess",
   "message": "Successfully created asset model.",
   "resources": {
      "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"
   }
}
```

Monitoring gateway logs

You can configure your gateway to log information to Amazon CloudWatch Logs or the local file system.

Topics
- Using Amazon CloudWatch Logs (p. 363)
- Using local file system logs (p. 364)
Using Amazon CloudWatch Logs

You can configure your gateway to send logs to CloudWatch Logs. You might follow this procedure if you want to use the AWS Management Console to view your gateway's log files.

To configure and access CloudWatch Logs

1. To configure CloudWatch Logs for your gateway, see Configure logging in the AWS IoT Greengrass Version 1 Developer Guide.
2. Navigate to the CloudWatch console.
3. In the navigation pane, choose Log groups.
4. You can find the AWS IoT SiteWise connector logs in the following log groups:
   - /aws/greengrass/Lambda/region/aws/swCollector – The logs for the gateway's Lambda function that collects data from the gateway's OPC-UA sources.
   - /aws/greengrass/Lambda/region/aws/swPublisher – The logs for the gateway's Lambda function that publishes OPC-UA data streams to AWS IoT SiteWise.

   Choose the log group for the function to debug.

5. Choose a log stream that has a name that ends with the name of your AWS IoT Greengrass group. By default, CloudWatch displays the most recent log stream first.

6. To show logs from the last 5 minutes, do the following:
   a. Choose custom in the upper-right corner.
   b. Choose Relative.
   c. Choose 5 minutes.
   d. Choose Apply.
Using local file system logs

You can configure your gateway to store logs locally. You might follow this procedure if you want to use the command line or local software to view your gateway’s log files.

To configure and access local file system logs

1. To configure local file system logs on your gateway, see Configure logging in the AWS IoT Greengrass Version 1 Developer Guide.

   Note
   You must have root permissions to read AWS IoT Greengrass logs on the file system.

2. On your gateway, run the following command to list all AWS IoT SiteWise connector log files and their file details. Replace `greengrass-root` with the root of your AWS IoT Greengrass installation, and replace `region` with the Region of the gateway. The default `greengrass-root` is `/greengrass`.

   ```bash
   sudo ls -l /greengrass-root/ggc/var/log/user/region/aws
   ```

You can find the most recent AWS IoT SiteWise connector logs in the following files:

- `swCollector.log` – The most recent logs for the Lambda function that collects data from the gateway’s OPC-UA sources.
- `swPublisher.log` – The most recent logs for the Lambda function that publishes OPC-UA data streams to AWS IoT SiteWise.

You can view earlier logs in the files named `swCollector.log-timestamp` and `swPublisher.log-timestamp`.

3. Run the following command to view the publisher log file contents, for example.
Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics

You can monitor AWS IoT SiteWise 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.

AWS IoT SiteWise publishes the metrics and dimensions listed in the sections below to the AWS/IoTSiteWise namespace.

**Tip**
AWS IoT SiteWise publishes metrics on a one minute interval. When you view these metrics in graphs in the CloudWatch console, we recommend that you choose a Period of 1 minute. This lets you see the highest available resolution of your metric data.

**Contents**
- Gateway metrics (p. 365)

**Gateway metrics**

AWS IoT SiteWise publishes the following gateway metrics. All gateway metrics are published on a one minute interval.

**Important**
To receive gateway metrics, you must use at least version 6 of the AWS IoT SiteWise connector on your gateway. For more information, see AWS IoT SiteWise connector in the AWS IoT Greengrass Version 1 Developer Guide.

**Gateway metrics**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway.Heartbeat</td>
<td>Generated every minute for each gateway (gatewayId) connected.</td>
</tr>
<tr>
<td>Gateway.PublishSuccessCount</td>
<td>The number of data points that a gateway (gatewayId) successfully published.</td>
</tr>
<tr>
<td>Gateway.PublishFailureCount</td>
<td>The number of data points that a gateway (gatewayId) failed to publish.</td>
</tr>
</tbody>
</table>

This metric counts errors that result from the gateway's calls to the BatchPutAssetPropertyValue operation. For more information about troubleshooting gateways, see Troubleshooting an AWS IoT SiteWise gateway (p. 372).
Logging AWS IoT SiteWise API calls with AWS CloudTrail

AWS IoT SiteWise is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS IoT SiteWise. CloudTrail captures API calls for AWS IoT SiteWise as events. The calls captured include calls from the AWS IoT SiteWise console and code calls to the AWS IoT SiteWise API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS IoT SiteWise. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT SiteWise, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

AWS IoT SiteWise information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in AWS IoT SiteWise, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for AWS IoT SiteWise, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway.ProcessFailureCount</td>
<td>The number of data points that a gateway (gatewayId) failed to process. This metric count errors that occur between the gateway and the gateway's sources, including errors reported by sources. For more information about troubleshooting gateways, see Troubleshooting an AWS IoT SiteWise gateway (p. 372).</td>
</tr>
</tbody>
</table>

### Gateway source metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCUACollector.Heartbeat</td>
<td>Generated every minute for each OPC-UA source (sourceName) connected to a gateway (gatewayId).</td>
</tr>
<tr>
<td>OPCUACollector.ActiveDataStreamCount</td>
<td>The number of data streams that a gateway (gatewayId) subscribed to for an OPC-UA source (sourceName).</td>
</tr>
<tr>
<td>OPCUACollector.IncomingValuesCount</td>
<td>The number of data points that a gateway (gatewayId) received for an OPC-UA source (sourceName).</td>
</tr>
</tbody>
</table>
configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

Most AWS IoT SiteWise operations are logged by CloudTrail and are documented in the AWS IoT SiteWise API Reference.

The following data plane operations aren't logged by CloudTrail:

- BatchPutAssetPropertyValue
- GetAssetPropertyValue
- GetAssetPropertyValueHistory
- GetAssetPropertyAggregates

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

**Example: AWS IoT SiteWise log file entries**

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the CreateAsset operation.

```json
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKGEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Administrator",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Administrator",
    "sessionContext": {
      "sessionIssuer": {},
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-03-11T17:26:40Z"
      }
    }
  }
}
```
Example: AWS IoT SiteWise log file entries

```
{
    "invokedBy": "signin.amazonaws.com",
    "eventTime": "2020-03-11T18:01:22Z",
    "eventSource": "iotsitewise.amazonaws.com",
    "eventName": "CreateAsset",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
        "assetName": "Wind Turbine 1",
        "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
        "clientToken": "a1b2c3d4-5678-90ab-cdef-00000EXAMPLE"
    },
    "responseElements": {
        "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
        "assetArn": "arn:aws:iotsitewise:us-east-1:123456789012:asset/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
        "assetStatus": {
            "state": "CREATING"
        }
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "eventID": "a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```
Tagging your AWS IoT SiteWise resources

With tags, you can organize and manage your resources in AWS IoT SiteWise. You can use tags to assign metadata to your resources, and you can use tags in IAM policies to define conditional access to your resources.

Using tags in AWS IoT SiteWise

You can use tags to categorize your AWS IoT SiteWise resources by purpose, owner, environment, or any other classification for your use case. When you have many resources of the same type, you can quickly identify a specific resource based on its tags.

Each tag consists of a key and an optional value, both of which you define. For example, you could define a set of tags for your asset models that helps you track them by the industrial processes to which assets of each model contribute. We recommend that you create a set of tag keys that meets your needs for each kind of resource. By using a consistent set of tag keys, you can more easily manage your resources.

Tagging with the AWS Management Console

The Tag Editor in the AWS Management Console provides a central, unified way for you to create and manage your tags for resources from all AWS services. For more information, see Tag Editor in the AWS Resource Groups User Guide.

Tagging with the AWS IoT SiteWise API

You can also work with tags by using the AWS IoT SiteWise API. Before you create tags, be aware of tagging restrictions. For more information, see Tag naming and usage conventions in the AWS General Reference.

- To add tags when you create a resource, define them in the tags property of the resource.
- To add tags to an existing resource, or to update tag values, use the TagResource operation.
- To remove tags from a resource, use the UntagResource operation.
- To retrieve the tags that are associated with a resource, use the ListTagsForResource operation, or describe the resource and inspect its tags property.

The following table lists resources you can tag using the AWS IoT SiteWise API and their corresponding Create and Describe operations.

Taggable AWS IoT SiteWise resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Create operation</th>
<th>Describe operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset model</td>
<td>CreateAssetModel</td>
<td>DescribeAssetModel</td>
</tr>
<tr>
<td>Asset</td>
<td>CreateAsset</td>
<td>DescribeAsset</td>
</tr>
</tbody>
</table>
Use the following operations to view and manage tags for resources that support tagging:

- **TagResource** – Adds tags to a resource, or updates an existing tag's value.
- **ListTagsForResource** – Lists the tags for a resource.
- **UntagResource** – Removes tags from a resource.

You can add or remove tags for a resource at any time. To change the value of a tag key, add a tag to the resource that defines the same key and the new value. The new value replaces the old value. You can set a value to an empty string, but you can't set a value to null.

When you delete a resource, tags that are associated with that resource are also deleted.

### Using tags with IAM policies

In your IAM policies, you can use resource tags to control user access and permissions. For example, policies can allow users to create only those resources that have a specific tag. Policies can also restrict users from creating or modifying resources that have certain tags.

**Note**

If you use tags to allow or deny users' access to resources, you should deny users the ability to add or remove those tags for the same resources. Otherwise, a user could circumvent your restrictions and gain access to a resource by modifying its tags.

You can use the following condition context keys and values in the `Condition` element (also called the `Condition` block) of a policy statement.

- `iotsitewise:ResourceTag/tag-key: tag-value`
- `aws:RequestTag/tag-key: tag-value`
- `aws:TagKeys: [tag-key, ...]`

**Note**

The condition context keys and values in an IAM policy apply only to actions that have a taggable resource as a required parameter. For example, you can set tag-based conditional access for `ListAssets`. You can't set tag-based conditional access on `PutLoggingOptions` because no taggable resource is referenced in the request.
For more information, see Controlling access to AWS resources using resource tags and IAM JSON policy reference in the IAM User Guide.

Example IAM policies using tags

- Viewing AWS IoT SiteWise assets based on tags (p. 338)
Troubleshooting AWS IoT SiteWise

Use the information in these sections to troubleshoot and resolve issues with AWS IoT SiteWise.

Topics
- Troubleshooting an AWS IoT SiteWise gateway (p. 372)
- Troubleshooting an AWS IoT SiteWise rule action (p. 374)

Troubleshooting an AWS IoT SiteWise gateway

AWS IoT SiteWise gateways run an AWS IoT Greengrass connector. You can configure your gateway to log connector events to CloudWatch and to your gateway's local file system. Then, you can view the log files associated with the connector to troubleshoot your gateway.

You can also view CloudWatch metrics reported by your gateways to troubleshoot issues with connectivity or data streams. For more information, see Gateway metrics (p. 365).

Topics
- Configuring and accessing AWS IoT SiteWise gateway logs (p. 372)
- Troubleshooting gateway issues (p. 372)
- Troubleshooting AWS IoT Greengrass issues (p. 374)

Configuring and accessing AWS IoT SiteWise gateway logs

Before you can view gateway logs, you must configure your gateway to send logs to Amazon CloudWatch Logs or store logs on the local file system.

- Use CloudWatch Logs if you want to use the AWS Management Console to view your gateway's log files. For more information, see Using Amazon CloudWatch Logs (p. 363).
- Use local file system logs if you want to use the command line or local software to view your gateway's log files. For more information, see Using local file system logs (p. 364).

Troubleshooting gateway issues

Use the following information to troubleshoot gateway issues.

Issues
- Modbus TCP sources are out of sync (p. 373)
- Unable to connect to stream manager (p. 373)
- Unable to connect to an OPC-UA source (p. 373)
- AWS IoT SiteWise doesn't receive data from OPC-UA servers (p. 374)
Modbus TCP sources are out of sync

Your Modbus TCP source might be out of sync if your source data type is ASCII, UTF8, or ISO8859 and you’re running an old version of the Modbus-TCP Protocol Adapter connector. To upgrade the connector to the latest version, do the following:

1. Sign in to the AWS IoT Greengrass V1 console.
2. In the navigation pane, choose Groups.
3. Under Greengrass groups, choose the target group.
4. In the navigation pane, choose Connectors.
5. In the Upgrade column, choose Available.
6. On the Upgrade connector page, choose the latest version, and then choose Upgrade.

For more information, see Modbus-TCP Protocol Adapter connector in the AWS IoT Greengrass Version 1 Developer Guide.

Unable to connect to stream manager

You might see the following swPublisher error log message if stream manager isn’t enabled on your gateway’s AWS IoT Greengrass group.

```
com.amazonaws.greengrass.streammanager.client.StreamManagerClientImpl: Connect failed
```

As of version 6, the AWS IoT SiteWise connector requires stream manager. For more information about how to enable stream manager, see step 5 of Configuring an AWS IoT Greengrass group (p. 96).

Unable to connect to an OPC-UA source

You might see the following OPCUACollector error log message if the version of the installed OpenJDK isn’t supported.

```
java.security.KeyStoreException: Key protection algorithm not found: PBEWithSHA1AndDESede
java.security.UnrecoverableKeyException: Encrypt Private Key failed: unrecognized algorithm name: PBEWithSHA1AndDESede
Failed to start OPC-UA Connection for Source 'Server 1': Failed to add key to store
```

To downgrade to the supported OpenJDK version, follow the steps in this section. These steps assume that you use a device with Ubuntu. If you use a different Linux distribution, consult the relevant documentation for your device.

To downgrade to the support Amazon Corretto 8

1. To uninstall the current OpenJDK, run one of the following commands.

   ```
   sudo apt purge -y openjdk-8-jre-headless
   ```

   ```
   sudo apt-get purge -y java-1.8.0-amazon-corretto-jdk
   ```

2. To download and install the supported Amazon Corretto 8, run the following command.

   ```
   curl -s https://corretto.aws/downloads/resources/8.282.08.1/java-1.8.0-amazon-corretto-jdk_8.282.08-1_amd64.deb --output /tmp/java-1.8.0-amazon-corretto-jdk_8.282.08-1_amd64.deb
   sudo apt-get update && sudo apt-get install java-common
   ```
AWS IoT SiteWise User Guide
Troubleshooting AWS IoT Greengrass issues

sudo dpkg --install /tmp/java-1.8.0-amazon-corretto-jdk_8.282.08-1_amd64.deb

3. To restart the AWS IoT Greengrass V1 Core software, run the following command.

sudo /greengrass/ggc/core/greengrassd restart

AWS IoT SiteWise doesn't receive data from OPC-UA servers

If your AWS IoT SiteWise assets aren't receiving data sent by your OPC-UA servers, you can search your gateway's logs to troubleshoot issues. Look for info-level swPublisher logs that contain the following message.

Emitting diagnostic name=PublishError.

SomeException

Based on the type of SomeException in the log, use the following exception types and corresponding issues to troubleshoot your gateway:

- **ResourceNotFoundException** – Your OPC-UA servers are sending data that doesn't match a property alias for any asset. This exception can occur in two cases:
  - Your property aliases don't exactly match your OPC-UA variables, including any source prefix you defined. Check that your property aliases and source prefixes are correct.
  - You haven't mapped your OPC-UA variables to asset properties. For more information, see Mapping industrial data streams to asset properties (p. 202).

  If you already mapped all of the OPC-UA variables that you want in AWS IoT SiteWise, you can filter which OPC-UA variables the gateway sends. For more information, see Using OPC-UA node filters (p. 120).

- **AccessDeniedException** – Your gateway's AWS IoT Greengrass group doesn't have sufficient permissions to use the BatchPutAssetPropertyValue operation to send data to asset properties. For more information, see the AWS IoT SiteWise connector requirements.

- **InvalidRequestException** – Your OPC-UA variables data types don't match your asset property data types. For example, if an OPC-UA variable has an integer data type, your corresponding asset property must be integer data type. A double-type asset property can't receive OPC-UA integer values. To fix this issue, define new properties with the correct data types.

- **TimestampOutOfRangeException** – Your gateway is sending data that is outside the range that AWS IoT SiteWise accepts. AWS IoT SiteWise rejects any data points with timestamps earlier than 7 days in the past or newer than 5 minutes in the future. If your gateway lost power or connection to the AWS Cloud, you might need to clear your gateway's cache.

- **ThrottlingException** or **LimitExceededException** – Your request exceeded an AWS IoT SiteWise service quota, such as rate of data points ingested or request rate for asset property data API operations. Check that your configuration doesn't exceed the AWS IoT SiteWise quotas (p. 382).

Troubleshooting AWS IoT Greengrass issues

To find solutions to many issues configuring or deploying your gateway on AWS IoT Greengrass, see Troubleshooting AWS IoT Greengrass in the AWS IoT Greengrass Developer Guide.

Troubleshooting an AWS IoT SiteWise rule action

To troubleshoot your AWS IoT SiteWise rule action in AWS IoT Core, you can do one of the following procedures:
• Configure CloudWatch Logs
• Configure a republish error action for your rule

Then, compare the error messages with the errors in this topic to troubleshoot your issue.

Topics
• Configuring AWS IoT Core logs (p. 375)
• Configuring a republish error action (p. 375)
• Troubleshooting issues (p. 377)

Configuring AWS IoT Core logs

You can configure AWS IoT to log various levels of information to CloudWatch Logs.

To configure and access CloudWatch Logs

1. To configure logging for AWS IoT Core, see Monitoring with CloudWatch Logs in the AWS IoT Developer Guide.
2. Navigate to the CloudWatch console.
3. In the navigation pane, choose Log groups.
4. Choose the AWSIotLogs group.
5. Choose a recent log stream. By default, CloudWatch displays the most recent log stream first.
6. Choose a log entry to expand the log message. Your log entry might look like the following screenshot.

7. Compare the error messages with the errors in this topic to troubleshoot your issue.

Configuring a republish error action

You can configure an error action on your rule to handle error messages. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client.

Note
The republish error action outputs only the equivalent of ERROR level logs. If you want more verbose logs, you must configure CloudWatch Logs (p. 375).

To add a republish error action to a rule

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose Act and then choose Rules.
3. Choose your rule.
4. Under Error action, choose Add action.
5. Choose Republish a message to an AWS IoT topic.
6. Choose Configure action at the bottom of the page.
7. In Topic, enter a unique topic (for example, sitewise/windfarm/rule/error). AWS IoT Core will republish error messages to this topic.
8. Choose Select to grant AWS IoT Core access to perform the error action.
9. Choose Select next to the role that you created for the rule.
10. Choose Update Role to add the additional permissions to the role.
11. Choose Add action.

Your rule's error action should look similar to the following screenshot.

```plaintext
Error action

Send a message to an Amazon Kinesis Stream

Republish a message to an AWS IoT topic

Store a message in an Amazon S3 bucket

Remove Edit
```

6. Choose Configure action at the bottom of the page.
7. In Topic, enter a unique topic (for example, sitewise/windfarm/rule/error). AWS IoT Core will republish error messages to this topic.
8. Choose Select to grant AWS IoT Core access to perform the error action.
9. Choose Select next to the role that you created for the rule.
10. Choose Update Role to add the additional permissions to the role.
11. Choose Add action.

Your rule's error action should look similar to the following screenshot.

```plaintext
Error action

Optionally set an action that will be executed when something goes wrong with processing your rule.

Republish a message to an AWS IoT topic

sitewise/windfarm/rule/error

Remove Edit
```

12. Choose the back arrow in the upper left of the console to return to the AWS IoT console home.

After you set up the republish error action, you can view the error messages in the MQTT test client in AWS IoT Core.

In the following procedure, you subscribe to the error topic in the MQTT test client. In the MQTT test client, you can receive your rule's error messages to troubleshoot the issue.

**To subscribe to the error action topic**
1. Navigate to the AWS IoT console.
2. In the left navigation page, choose Test to open the MQTT test client.
3. In the Subscription topic field, enter the error topic that you configured earlier (for example, sitewise/windfarm/rule/error) and choose Subscribe to topic.
4. Watch for error messages to appear and then expand the `failures` array in any error message. Next, compare the error messages with the errors in this topic to troubleshoot your issue.

## Troubleshooting issues

Use the following information to troubleshoot rule issues.

### Issues

- **Error: Member must be within 604800 seconds before and 300 seconds after the current timestamp** (p. 377)
- **Error: Property value does not match data type `<type>`** (p. 377)
- **Error: User: `<role-arn>` is not authorized to perform: iotsitewise:BatchPutAssetPropertyValue on resource** (p. 378)
- **Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: `<role-arn>`** (p. 378)
- **Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates.** (p. 378)

### Error: Member must be within 604800 seconds before and 300 seconds after the current timestamp

Your timestamp is older than 7 days or newer than 5 minutes, compared to current Unix epoch time. Try the following:

- Check that your timestamp is in Unix epoch (UTC) time. If you provide a timestamp with a different timezone, you receive this error.
- Check that your timestamp is in seconds. AWS IoT SiteWise expects timestamps split into time in seconds (in Unix epoch time) and offset in nanoseconds.
- Check that you're uploading data that is timestamped no later than 7 days in the past.

### Error: Property value does not match data type `<type>`

An entry in your rule action has a different data type than the target asset property. For example, your target asset property is a `DOUBLE` and your selected data type is `Integer` or you passed the value in `integerValue`. Try the following:

- If you configure the rule from the AWS IoT console, check that you chose the correct **Data type** for each entry.
• If you configure the rule from the API or AWS Command Line Interface (AWS CLI), check that your value object uses the correct type field (for example, \texttt{doubleValue} for a DOUBLE property).

**Error: User: <role-arn> is not authorized to perform: iotsitewise:BatchPutAssetPropertyValue on resource**

Your rule isn't authorized to access the target asset property, or the target asset property doesn't exist. Try the following:

• Check that your property alias is correct and that you have an asset property with the given property alias. For more information, see Mapping industrial data streams to asset properties (p. 202).

• Check that your rule has a role and that the role allows \texttt{iotsitewise:BatchPutAssetPropertyValue} permission to the targeted asset property, such as through the target asset's hierarchy. For more information, see Granting AWS IoT the required access (p. 69).

**Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: <role-arn>**

Your IAM user isn't authorized to assume the role on your rule.

Check that your IAM user is allowed \texttt{iam:PassRole} permission to the role on your rule. For more information, see Pass role permissions in the AWS IoT Developer Guide.

**Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates.**

**Note**

This message is an INFO level log.

Your request must have at least one entry with all of the required parameters.

Check that your rule's parameters, including substitution templates, result in non-empty values. Substitution templates can't access values defined in \texttt{AS} clauses in your rule query statement. For more information, see Substitution templates in the AWS IoT Developer Guide.
Integrating with Grafana

Grafana is a data visualization platform that you can use to visualize and monitor data in dashboards. In Grafana version 7.3.0 and later, you can use the AWS IoT SiteWise plugin to visualize your AWS IoT SiteWise asset data in Grafana dashboards. You can visualize data from multiple AWS sources (such as AWS IoT SiteWise, Amazon Timestream, and Amazon CloudWatch) and other data sources with a single Grafana dashboard.

You have two options to use the AWS IoT SiteWise plugin:

- **Local Grafana servers**
  
  You can set up the AWS IoT SiteWise plugin on a Grafana server that you manage. For more information about how to add and use the plugin, see the AWS IoT SiteWise Datasource README file on the GitHub website.

- **AWS Managed Service for Grafana**
  
  You can use the AWS IoT SiteWise plugin in the AWS Managed Service for Grafana (AMG). AMG manages Grafana servers for you so that you can visualize your data without having to build, package, or deploy any hardware or any other Grafana infrastructure. For more information, see the following topics in the AWS Managed Service for Grafana User Guide:
  
  - What is Amazon Managed Service for Grafana (AMG)?
  - Using the AWS IoT SiteWise data source

**Example Example Grafana dashboard**

The following Grafana dashboard visualizes the demo wind farm (p. 10). You can access this demo dashboard on the Grafana Play website.
AWS IoT SiteWise endpoints and quotas

The following sections describe the endpoints and quotas for AWS IoT SiteWise.

Contents

- AWS IoT SiteWise endpoints (p. 381)
- AWS IoT SiteWise quotas (p. 382)

AWS IoT SiteWise endpoints

To connect programmatically to AWS IoT SiteWise, you use an endpoint. The AWS SDKs and the AWS Command Line Interface (AWS CLI) automatically use the default endpoint in an AWS Region. For more information about Regions where AWS IoT SiteWise is available, see AWS IoT SiteWise endpoints and quotas in the AWS General Reference.

AWS IoT SiteWise supports the following endpoints.

data.iotsitewise.region.amazonaws.com

Use this endpoint to access the following data plane API operations: BatchPutAssetPropertyValue, GetAssetPropertyAggregates, GetAssetPropertyValue, GetAssetPropertyValueHistory and GetInterpolatedAssetPropertyValues. Replace region with your AWS Region.

api.iotsitewise.region.amazonaws.com

AWS IoT SiteWise offers this consolidated endpoint for the control plane API operations that you use to manage asset models, assets, gateways, tags, and account configurations. Replace region with your AWS Region.

Note

- By default, AWS IoT SiteWise uses the consolidated endpoint when you make calls to the supported control plane API operations.
- We recommend that you use the consolidated endpoint for the supported control plane API operations.
- You can't use the consolidated endpoint to access the SiteWise Monitor API operations.


The interface VPC endpoint for the control plane API operations only supports the consolidated endpoint. For more information, see AWS IoT SiteWise and interface VPC endpoints (AWS PrivateLink) (p. 355).
iotsitewise.region.amazonaws.com

You can use this endpoint to access the following API operations: DescribeStorageConfiguration, PutStorageConfiguration, DescribeDefaultEncryptionConfiguration, ListTagsForResource, PutDefaultEncryptionConfiguration, TagResource, and UntagResource. Replace region with your AWS Region.

model.iotsitewise.region.amazonaws.com

You can use this endpoint to access the following API operations: AssociateAssets, CreateAsset, CreateAssetModel, DeleteAsset, DeleteAssetModel, DeleteDashboard, DescribeAsset, DescribeAssetModel, DescribeAssetProperty, DescribeDashboard, DescribeLoggingOptions, DisassociateAssets, ListAssetModels, ListAssetRelationships, ListAssets, ListAssociatedAssets, PutLoggingOptions, UpdateAsset, UpdateAssetModel, and UpdateAssetProperty. Replace region with your AWS Region.

dge.iotsitewise.region.amazonaws.com

You can use this endpoint to access the following API operations: CreateGateway, DeleteGateway, DescribeGateway, DescribeGatewayCapabilityConfiguration, ListGateways, UpdateGateway, and UpdateGatewayCapabilityConfiguration. Replace region with your AWS Region.

monitor.iotsitewise.region.amazonaws.com

Use this endpoint to access the following API operations: BatchAssociateProjectAssets, BatchDisassociateProjectAssets, CreateAccessPolicy, CreateDashboard, CreatePortal, CreateProject, DeleteAccessPolicy, DeletePortal, DeleteProject, DescribeAccessPolicy, DescribePortal, DescribeProject, ListAccessPolicies, ListDashboards, ListPortals, ListProjectAssets, ListProjects, UpdateAccessPolicy, UpdateDashboard, UpdatePortal, and UpdateProject. Replace region with your AWS Region.

AWS IoT SiteWise quotas

The following tables describe quotas in AWS IoT SiteWise. For more information about quotas and how to request quota increases, see AWS service quotas in the AWS General Reference.

### Quotas for assets and asset models

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of asset models per Region per AWS account</td>
<td>100</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of assets per asset model</td>
<td>10,000</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of child assets per parent asset</td>
<td>100</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Depth of asset hierarchy tree</td>
<td>10</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of asset hierarchy definitions per asset model</td>
<td>10</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
### Quotas

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of properties per asset model</td>
<td>200</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of property variables per property formula expression</td>
<td>10</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of functions per property formula expression</td>
<td>10</td>
<td>Yes</td>
<td>For example, a model with a transform property C that consumes a transform property B that consumes a measurement property A has a depth of 3.</td>
</tr>
<tr>
<td>Depth of property tree per asset model</td>
<td>10</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of asset models per hierarchy tree</td>
<td>20</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of directly dependent properties per asset model</td>
<td>20</td>
<td>Yes</td>
<td>This quota limits how many properties can directly depend on a single property, as defined in property formula expressions.</td>
</tr>
<tr>
<td>Number of dependent properties per asset model</td>
<td>30</td>
<td>Yes</td>
<td>This quota limits how many properties can directly or indirectly depend on a single property, as defined in property formula expressions.</td>
</tr>
<tr>
<td>Request rate for model API operations and logging options</td>
<td>10 requests per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to API operations such as CreateAssetModel and logging options.</td>
</tr>
<tr>
<td>Request rate for asset API operations</td>
<td>30 requests per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to API operations such as CreateAsset and AssociateAssets.</td>
</tr>
</tbody>
</table>

### Quotas for asset property data

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request rate for asset property data API operations</td>
<td>1,000 requests per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to API operations such as GetAssetPropertyValue.</td>
</tr>
<tr>
<td>Resource</td>
<td>Quota</td>
<td>Adjustable</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of data points per second per data quality per asset property</td>
<td>10 data points</td>
<td>No</td>
<td>This quota applies to the maximum number of timestamp-quality-value (TQV) data points with the same timestamp in seconds per data quality for each asset property. You can store up to this number of good-quality, uncertain-quality, and bad-quality data points for any given second for each asset property.</td>
</tr>
<tr>
<td>Rate of data entries ingested per asset property</td>
<td>10 entries per asset property</td>
<td>Yes</td>
<td>This quota applies to BatchPutAssetPropertyValue entries from all sources, including gateways, AWS IoT Core rules, and API calls.</td>
</tr>
<tr>
<td>Rate of data points ingested</td>
<td>1,000 data points per second per Region per AWS account</td>
<td>Yes</td>
<td>Timestamp-quality-value (TQV) data points.</td>
</tr>
<tr>
<td>Rate of data points computed</td>
<td>10,000 data points per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to the number of timestamp-quality-value (TQV) data points output by transform and metric computations.</td>
</tr>
<tr>
<td>Number of data points processed per metric computation</td>
<td>200,000</td>
<td>No</td>
<td>Timestamp-quality-value (TQV) data points.</td>
</tr>
<tr>
<td>Rate of GetInterpolatedAssetPropertyValue requests</td>
<td>500</td>
<td>Yes</td>
<td>The maximum number of GetInterpolatedAssetPropertyValue requests per second that you can perform in this account in the current Region.</td>
</tr>
<tr>
<td>Number of results per GetInterpolatedAssetPropertyValue request</td>
<td>10</td>
<td>Yes</td>
<td>The maximum number of results to return per paginated GetInterpolatedAssetPropertyValue request.</td>
</tr>
</tbody>
</table>
### Quotas

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of days between the start date in the past and today for GetInterpolatedAssetPropertyValues</td>
<td>28</td>
<td>Yes</td>
<td>The maximum number of days between the start date and today. This quota applies to the startTimeInSeconds parameter when you specify a start date in the past for GetInterpolatedAssetPropertyValues requests.</td>
</tr>
</tbody>
</table>

#### Quotas for AWS IoT SiteWise gateways

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of gateways per Region per AWS account</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of OPC-UA sources per gateway</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of Modbus TCP sources per gateway</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of Ethernet/IP sources per gateway</td>
<td>100</td>
<td>No</td>
</tr>
</tbody>
</table>

#### Quotas for AWS IoT SiteWise Monitor

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of portals per Region per AWS account</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of projects per portal</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of dashboards per project</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of root assets per project</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Number of visualizations per dashboard</td>
<td>10</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of metrics per dashboard visualization</td>
<td>8</td>
<td>No</td>
</tr>
</tbody>
</table>
Document history for the AWS IoT SiteWise User Guide

The following table describes the documentation for this release of AWS IoT SiteWise.

- **API version:** 2019-12-02

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated functions</td>
<td>Added the following features</td>
<td>August 10, 2021</td>
</tr>
<tr>
<td></td>
<td>• In metrics, you can use nested expressions in aggregation functions and temporal functions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• In transforms, you can use the pretrigger() function to retrieve the value of a variable prior to the property update that triggered the current transform calculation.</td>
<td></td>
</tr>
</tbody>
</table>

| Custom metric time interval  | Added support for custom time intervals and offsets in metrics.                           | August 3, 2021      |
| Using AWS IoT SiteWise at the edge | The edge processing feature is now generally available.                                    | July 29, 2021       |
| Exporting data to Amazon S3  | AWS IoT SiteWise now can export data to Amazon S3.                                         | July 27, 2021       |
| VPC endpoints (AWS PrivateLink) | The interface VPC endpoint for the control plane API operations is now generally available. | July 15, 2021       |
| Transforms                   | Transforms now can input multiple asset property variables.                                | July 8, 2021        |
| Updated the timestamp() function | In transforms, you can now provide a variable as an argument to the timestamp() function.   | June 16, 2021       |
| Alarms general availability  | The alarms feature is now generally available.                                             | May 27, 2021        |
| Modbus-TCP Protocol Adapter version 2 released | Version 2 of the Modbus-TCP Protocol Adapter connector is available. This release added support for ASCII, UTF8, and ISO8859 encoded source strings. | May 24, 2021        |
| Updated service quotas | Added the following quotas for the `GetInterpolatedAssetPropertyValues` API: rate of `GetInterpolatedAssetPropertyValues` requests, number of results per `GetInterpolatedAssetPropertyValues` request, and number of days between the start date in the past and today for `GetInterpolatedAssetPropertyValues`. | April 29, 2021 |
| Updated formula expressions | Added the following operators and functions:  
  • Added the following operators: `<`, `>`, `<=`, `>=`, `==`, `!=`, `!`, `and`, `or`, and `not`.  
  • Added the following comparison function: `neq(x, y)`.  
  • Added the following string functions: `join()`, `format()`, and `f`"`. | April 22, 2021 |
<p>| VPC endpoints (AWS PrivateLink) | Added information about how to establish a private connection between your virtual private cloud (VPC) and the AWS IoT SiteWise control plane APIs by creating an interface VPC endpoint. | March 16, 2021 |
| IAM federation | Your SiteWise Monitor portal administrators and users can now log in to their assigned portals with their IAM credentials. | March 16, 2021 |
| Region launch | Launched AWS IoT SiteWise in China (Beijing). | February 3, 2021 |
| IoT SiteWise connector version 10 released | Version 10 of the IoT SiteWise connector is available. This release configures <code>StreamManager</code> to improve handling when the source connection is lost and re-established. This version also accepts OPC-UA values with a <code>ServerTimestamp</code> when no <code>SourceTimestamp</code> is available. | January 22, 2021 |
| Date and time functions | AWS IoT SiteWise now supports date and time functions. | January 21, 2021 |</p>
<table>
<thead>
<tr>
<th><strong>AWS IoT SiteWise User Guide</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function syntax</strong></td>
</tr>
<tr>
<td><strong>Integrating with Grafana</strong></td>
</tr>
<tr>
<td><strong>AWS IoT SiteWise feature release</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>AWS IoT SiteWise now supports customer managed CMKs.</strong></td>
</tr>
<tr>
<td><strong>IoT SiteWise connector version 8 released</strong></td>
</tr>
<tr>
<td><strong>Using strings and conditionals in formula expressions</strong></td>
</tr>
<tr>
<td>Feature</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Ingesting data using AWS IoT Greengrass stream manager</td>
</tr>
<tr>
<td>VPC endpoints (AWS PrivateLink)</td>
</tr>
<tr>
<td>IoT SiteWise connector version 7 released</td>
</tr>
<tr>
<td>Creating AWS SSO users from the AWS IoT SiteWise console</td>
</tr>
<tr>
<td>Improved gateway troubleshooting</td>
</tr>
<tr>
<td>Console task documentation (p. 386)</td>
</tr>
<tr>
<td>Analyzing exported data tutorial</td>
</tr>
<tr>
<td>Improved using formula expressions</td>
</tr>
<tr>
<td>IoT SiteWise connector version 6 released</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Version 6 of the IoT SiteWise connector is available. This release adds support for CloudWatch metrics and automatic discovery of new OPC-UA tags. This means you don't need to restart your gateway when tags change for your OPC-UA sources. This version of the connector requires stream manager and AWS IoT Greengrass Core software v1.10.0 or higher.</td>
</tr>
<tr>
<td>AWS IoT SiteWise feature release</td>
</tr>
<tr>
<td>---------------------------------</td>
</tr>
<tr>
<td>• Added the <strong>Exporting data to Amazon S3</strong> (p. 309) section with an AWS CloudFormation template that you can use to export new data values to an S3 bucket.</td>
</tr>
<tr>
<td>• Added the <strong>Configuring data sources</strong> (p. 102) section that improves gateway source documentation and includes the new gateway APIs.</td>
</tr>
<tr>
<td>• Added the <strong>gateway metrics</strong> (p. 365) section that describes the CloudWatch metrics that gateways publish.</td>
</tr>
<tr>
<td>• Added the <strong>Configuring a gateway on Amazon EC2</strong> (p. 101) section with an AWS CloudFormation template that you can use to quickly configure gateway dependencies on an Amazon EC2 instance.</td>
</tr>
<tr>
<td>• Added the <strong>portal service roles</strong> (p. 341) section that describes the new permissions feature of SiteWise Monitor portals.</td>
</tr>
<tr>
<td>• Updated <strong>portal documentation</strong> (p. 262) for portal service roles and portal logos.</td>
</tr>
<tr>
<td>• Added the <strong>Tagging your AWS IoT SiteWise resources</strong> (p. 369) section.</td>
</tr>
<tr>
<td>• Updated the <strong>Creating dashboards (CLI)</strong> (p. 280) section for the new dashboard definition structure.</td>
</tr>
<tr>
<td>• Added the <strong>Security</strong> (p. 320) section.</td>
</tr>
</tbody>
</table>

<p>| Ingesting data from AWS IoT Events | Added information about how to ingest data from AWS IoT Events when an event occurs. | April 20, 2020 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualizing and sharing wind farm data in SiteWise Monitor tutorial</td>
<td>Added a tutorial that you can follow to learn how to use AWS IoT SiteWise Monitor to visualize and share asset data.</td>
</tr>
<tr>
<td>AWS IoT SiteWise concepts</td>
<td>Added a glossary of AWS IoT SiteWise concepts that you can use to learn about the service and its common terms.</td>
</tr>
<tr>
<td>Removed AWS IoT Greengrass installation instructions (p. 386)</td>
<td>Removed the AWS IoT Greengrass Core software installation instructions from the AWS IoT SiteWise User Guide. The <a href="https://docs.aws.amazon.com/iot-greengrass/latest/developerguide/">AWS IoT Greengrass Developer Guide</a> offers a device setup script and instructions to set up AWS IoT Greengrass on other platforms such as Amazon EC2 and Docker.</td>
</tr>
<tr>
<td>Improved ingesting data using AWS IoT Core rules</td>
<td>Added detailed information about how to use and how to troubleshoot the AWS IoT SiteWise rule action, which you can use to ingest data from MQTT messages through AWS IoT Core.</td>
</tr>
<tr>
<td>IoT SiteWise connector version 5 released</td>
<td>Version 5 of the IoT SiteWise connector is available. This release fixes a compatibility issue with AWS IoT Greengrass Core software v1.9.4.</td>
</tr>
<tr>
<td>IoT SiteWise connector version 4 released</td>
<td>Version 4 of the IoT SiteWise connector is available. This release fixes an issue with OPC-UA server reconnection.</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Restructured modeling industrial assets</td>
<td>Restructured the Updating Assets and Models section into multiple topics within Modeling Industrial Assets.</td>
</tr>
<tr>
<td></td>
<td>- Asset and model states (p. 140)</td>
</tr>
<tr>
<td></td>
<td>- Mapping industrial data streams to asset properties (p. 202)</td>
</tr>
<tr>
<td></td>
<td>- Updating attribute values (p. 204)</td>
</tr>
<tr>
<td></td>
<td>- Associating and disassociating assets (p. 207)</td>
</tr>
<tr>
<td></td>
<td>- Updating assets and models (p. 210)</td>
</tr>
<tr>
<td></td>
<td>- Deleting assets and models (p. 215)</td>
</tr>
<tr>
<td>Ingesting data from AWS IoT things tutorial</td>
<td>Added a tutorial that you can follow to learn how to configure an AWS IoT SiteWise rule action to ingest data from a new or existing fleet of AWS IoT things.</td>
</tr>
<tr>
<td>Restructured retrieving data from AWS IoT SiteWise (p. 386)</td>
<td>Restructured the Retrieving Data section into two top-level sections: Querying asset property values and aggregates and Interacting with other AWS services.</td>
</tr>
<tr>
<td>Publishing property value updates to Amazon DynamoDB tutorial</td>
<td>Added a tutorial that you can follow to learn how to use property value notifications to store asset data in DynamoDB.</td>
</tr>
<tr>
<td>Using formula expressions</td>
<td>Added the formula expression reference to organize the constants and functions available for use in transform and metric properties. Restructured Asset properties into separate topics for each property type.</td>
</tr>
<tr>
<td>Using OPC-UA node filters</td>
<td>Added information about how to use OPC-UA node filters to improve gateway performance when adding gateway sources.</td>
</tr>
<tr>
<td>Upgrading a connector</td>
<td>Added information about how to upgrade a gateway when a new connector version is released.</td>
</tr>
<tr>
<td>Release Date</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>December 17, 2019</td>
<td>IoT SiteWise connector version 3 released. Version 3 of the IoT SiteWise connector is available. This release removes the iot:* permissions requirement.</td>
</tr>
<tr>
<td>December 10, 2019</td>
<td>IoT SiteWise connector version 2 released. Version 2 of the IoT SiteWise connector is available. This release adds support for multiple OPC-UA secret resources.</td>
</tr>
<tr>
<td>December 6, 2019</td>
<td>Creating dashboards (AWS CLI) Added information about how to create a dashboard in AWS IoT SiteWise Monitor using the AWS CLI.</td>
</tr>
<tr>
<td>December 2, 2019</td>
<td>AWS IoT SiteWise version 2 released. Released preview for version 2 of AWS IoT SiteWise. You can now ingest data over OPC-UA, MQTT, and HTTP, model your data in asset hierarchies, and visualize your data with SiteWise Monitor.</td>
</tr>
<tr>
<td></td>
<td>- Rewrote the asset modeling (p. 139) section for changes to assets, asset models, and asset hierarchies.</td>
</tr>
<tr>
<td></td>
<td>- Updated the data ingestion (p. 69) section to include AWS IoT Greengrass connector steps and non-gateway data ingestion sections.</td>
</tr>
<tr>
<td></td>
<td>- Added the AWS IoT SiteWise Monitor (p. 257) section and a separate application guide that shows how to use the SiteWise Monitor web application.</td>
</tr>
<tr>
<td></td>
<td>- Added Querying asset property values and aggregates (p. 285) and Interacting with other AWS services (p. 291) sections.</td>
</tr>
<tr>
<td></td>
<td>- Rewrote the getting started (p. 9) section to match the updated demo experience.</td>
</tr>
<tr>
<td>February 25, 2019</td>
<td>AWS IoT SiteWise version 1 released. Released initial preview for version 1 of AWS IoT SiteWise.</td>
</tr>
</tbody>
</table>
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.