User Guide

AWS IoT SiteWise

AWS IoT SiteWise: User Guide

Table of Contents

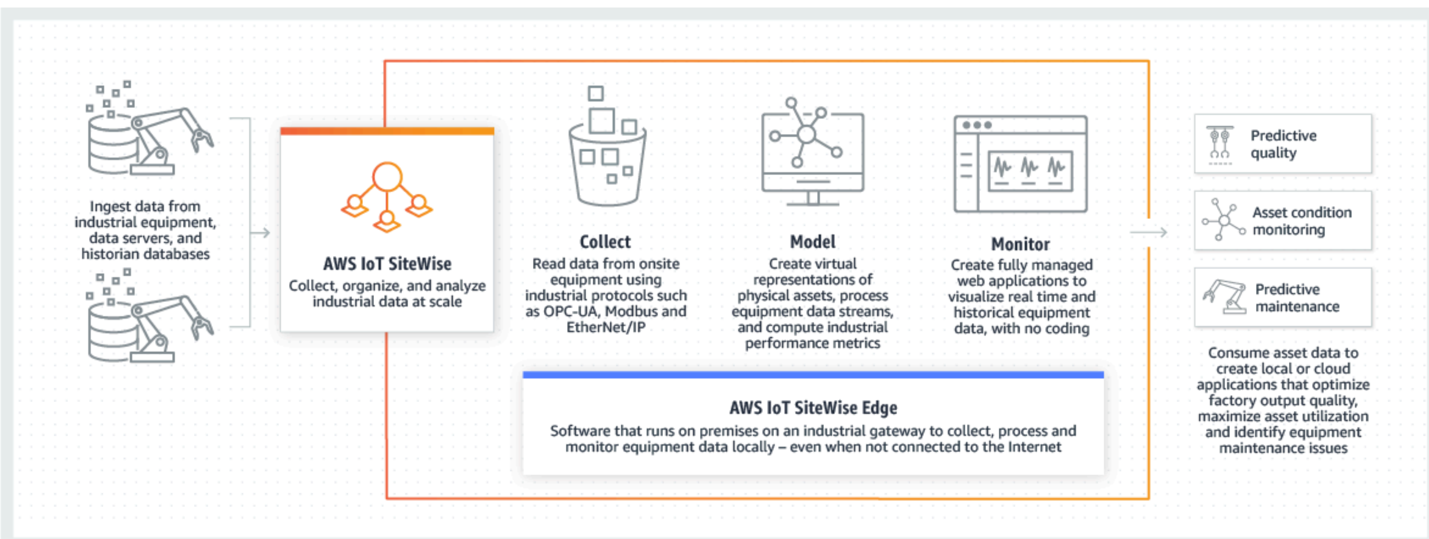
What is AWS IoT SiteWise?

AWS IoT SiteWise is a managed service with which you can collect, store, organize and monitor data from industrial equipment at scale to help you make better, data-driven decisions. You can use AWS IoT SiteWise to monitor operations across facilities, quickly compute common industrial performance metrics, and create applications that analyze industrial equipment data to prevent costly equipment issues and reduce gaps in production.

With AWS IoT SiteWise Monitor, your operational users can create web applications 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)*.

AWS IoT SiteWise Edge is a component of AWS IoT SiteWise that allows collection, storage and processing of data on local devices. This is useful if you have limited access to the internet or need to keep your data private.

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



Topics

- [How AWS IoT SiteWise works]
- [Use cases for AWS IoT SiteWise]
- [Using this service with an AWS SDK]
- [AWS IoT SiteWise concepts]

How AWS IoT SiteWise works

AWS IoT SiteWise offers a resource modeling framework that you can use to create representations of your industrial devices, processes, and facilities. The representations of your equipment and processes are called asset models in AWS IoT SiteWise. With asset models, you define the raw data to consume and how to process it into useful metrics. Build and visualize assets and models for your industrial operation in the AWS IoT SiteWise console. You can also configure asset models to collect and process data at the edge or in the AWS Cloud.

Topics

Ingest industrial data

Begin to use AWS IoT SiteWise by ingesting industrial data. Ingesting your data is done in one of several ways:

- **Direct ingestion from on-site servers:** Utilize protocols like OPC UA to read data directly from on-site devices. Deploy the SiteWise Edge gateway software, compatible with AWS IoT Greengrass V2, on a wide range of platforms such as common industrial gateways or virtual servers. You can connect up to 100 OPC UA servers to a single AWS IoT SiteWise gateway. For more information, see AWS IoT SiteWise Edge self-hosted gateway requirements.

  Note that protocols like Modbus TCP and Ethernet/IP (EIP) are supported through our partnership with Domatica in the context of AWS IoT Greengrass V2.
- **Edge data processing with packs:** Enhance your SiteWise Edge gateway by adding packs to enable comprehensive edge capabilities. With SiteWise Edge, available on AWS IoT Greengrass V2, data processing is executed directly on-site before being securely transmitted to the AWS Cloud using an AWS IoT Greengrass stream. For more information, see Set up an OPC UA source in SiteWise Edge.

- **Adaptive ingestion via Amazon S3 with bulk operations:** When working with large numbers of assets or asset models, use bulk operations to bulk import and export resources from Amazon S3 buckets. For more information, see Bulk operations with assets and models.
- **MQTT messages with AWS IoT Core Rules:** For devices connected to AWS IoT Core sending MQTT messages, employ the AWS IoT Core rules engine to direct those messages to AWS IoT SiteWise.If you have devices connected to AWS IoT Core sending MQTT messages, use the AWS IoT Core rules engine to route those messages to AWS IoT SiteWise. For more information, see Ingest data to AWS IoT SiteWise using AWS IoT Core rules.
- **Event-triggered data ingestion:** Use *AWS IoT Events actions* to configure the IoT SiteWise action in AWS IoT Events to send data to AWS IoT SiteWise when events occur. For more information, see Ingest data to AWS IoT SiteWise from AWS IoT Events.
- **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 Ingest data with AWS IoT SiteWise APIs.

Model assets to contextualize gathered data

After ingesting data, you can use the data to create virtual representations of your assets, processes, and facilities by building models of your physical operations. An asset, representing a device or process, transmits data streams to the AWS Cloud. Assets can also signify logical device groupings. Hierarchies are formed by associating assets to mirror complex operations. These hierarchies allow assets to access data from associated child assets. Assets are created from asset models. Asset models are declarative structures that standardize asset formats. Reuse components of assets for organization and maintainability of your models. For more information, see Model industrial assets.

With AWS IoT SiteWise, you can configure your assets to transform the incoming data into contextual metrics and transforms.

- Transforms work when receiving equipment data.
- Metrics are calculated at intervals you define.

Metrics and transforms are applicable to both individual assets or multiple assets.AWS IoT SiteWise automatically computes commonly used statistical aggregates like average, sum, and count, across various time frames relevant to your equipment data, metrics, and transforms.

Assets can be synchronized using AWS IoT TwinMaker. For more information, see Integrating AWS IoT SiteWise and AWS IoT TwinMaker.

Analyze using queries, alarms, and predictions

Analyze the date gathered with AWS IoT SiteWise by running queries and setting up alarms. You can also use Amazon Lookout to automatically detect anomalies within metrics and identify their root causes.

- Set specific alarms to alert your team when equipment or processes deviate from optimal performance, ensuring quick issue identification and resolution. For more information, see Monitor data with alarms in AWS IoT SiteWise.
- Use the AWS IoT SiteWise API operations to query your asset properties' current values, historical values, and aggregates over specific time intervals. For more information, see Query data from AWS IoT SiteWise.
- Use anomaly detection with Amazon Lookout for Equipment to identify and visualize changes in equipment or operating conditions. With anomaly detection, you can determine preventative maintenance measures for your operations. This integration allows customers to sync data between AWS IoT SiteWise and Amazon Lookout for Equipment. For more information, see Detect anomalies with Lookout for Equipment.

Visualize operations

Set up SiteWise Monitor to create web applications for your operational employees. The web applications help employees to visualize your operations. Handle varied levels of access for your employees using IAM Identity Center or IAM. 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.

For more information on visualizing your operations, see Monitor data with AWS IoT SiteWise Monitor.

Store data

You can integrate time series storage with your industrial data lake. AWS IoT SiteWise has three storage tiers for industrial data:

- A hot storage tier that is optimized for real-time applications.

- A warm storage tier optimized for analytical workloads.
- A customer-managed cold storage tier using Amazon S3 for operational data applications with high latency tolerance.

AWS IoT SiteWise helps you manage storage cost by keeping recent data in the hot storage tier. Then, you define data retention policies to move historical data to warm or cold tier storage. For more information, see Manage data storage in AWS IoT SiteWise.

You can also import and export asset metadata. For more information see Asset metadata.

Integrate with other services

AWS IoT SiteWise integrates with several AWS services to develop a complete AWS IoT solution in the AWS Cloud. For more information, see Interact with other AWS services.

Use cases for AWS IoT SiteWise

AWS IoT SiteWise is used across a variety of industries for many industrial data collection and analysis applications.

Collect data consistently from all your sources to help resolve issues quickly. AWS IoT SiteWise offers remote monitoring to collect the data directly on-site or gather it from multiple sources across many facilities. AWS IoT SiteWise provides the necessary flexibility for industrial IoT data solutions.

Manufacturing

AWS IoT SiteWise can simplify the process of collecting and utilizing data from your equipment to pinpoint and minimize inefficiencies, enhancing industrial operations. AWS IoT SiteWise helps you collect data from manufacturing lines and equipment. With AWS IoT SiteWise, you can transfer the data to the AWS Cloud and build performance metrics for your specific equipment and processes. You can use the metrics produced 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. Food processing plants often span multiple locations with plant and equipment operators in a centralized location to monitor processes and equipment. For example, refrigeration units assess ingredient handling and expiration. They monitor 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 facilities so your process engineers can better understand and make improvements across facilities.

Energy and utilities

With AWS IoT SiteWise, you can resolve equipment issues easier and more efficiently. You can monitor asset performance remotely and in real time. Access historical equipment data from anywhere to pinpoint potential problems, dispatch accurate resources, and both prevent and fix issues faster.

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples

SDK documentation	Code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

AWS IoT SiteWise concepts

The following are the core concepts of AWS IoT SiteWise:

Aggregate

Aggregates are fundamental metrics, or measurements, that AWS IoT SiteWise automatically calculates for all time series data. For more information, see Query asset property aggregates in AWS IoT SiteWise.

Asset

When you input, or ingest, data into AWS IoT SiteWise from your industrial equipment, your devices, equipment, and processes are each shown as assets. Each asset has associated data. For example, a piece of equipment might have a serial number, a location, a make and model, and an installation date. It might also have time series values for availability, performance, quality, temperature, pressure, and more. Group assets into hierarchies, allowing assets to access data stored in their child assets. For more information, see Model industrial assets.

Asset hierarchy

Set up asset hierarchies to create logical representations of your industrial operations. To do this, define a hierarchy in an asset model and associate assets created from that model with the specified hierarchy. Metrics in parent assets can combine data from the properties of child assets, allowing you to calculate metrics that offer insights into your overall operation or a specific part of it. For more information, see Define asset model hierarchies.

Asset model

Every asset is made using an asset model. Asset models are structures that define and standardize the format of your assets. They ensure consistent information across multiple assets of the same type, allowing you to handle data in assets that represent groups of devices. In each asset model, you can define attributes, time series inputs (measurements), time series transformations (transforms), time series aggregations (metrics), and asset hierarchies. For more information, see Model industrial assets.

Decide where your asset model's properties are processed by configuring your asset model for the edge. Utilize this feature to handle and monitor asset data on your local devices.

Asset property

Asset properties are the structures within each asset that hold industrial data. Each property has a data type and can also have a unit. A property can be an attribute, a measurement, a transform, or a metric. For more information, see Define data properties.

Configure asset properties to compute at the edge. For more information about processing data at the edge, see Set up an OPC UA source in SiteWise Edge.

Attribute

Attributes are properties of an asset that typically stay constant, like the device manufacturer or device location. Attributes can have preset values. Every asset created from an asset model includes the default values of the attributes defined in that model. For more information, see Define static data (attributes).

Computation model

A `ComputationModel` is an abstraction for certain types of compute that can enact on your data. It defines the blueprint for a suite of computations, where users can describe input, output and configuration for a specific computation engine. `ComputationModel` is a new resource with ARN and is stateful and versioned. For more information, see Create a computation model (AWS CLI).

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 customize time range to look at specific data.

Dataset

Datasets are collections of data that represents time-series data, non-time-series data, and non-equipment data such as shift schedules, maintenance records, and employee databases. They support external data and use AWS IoT SiteWise analytic capabilities. It includes dataset sources, dataset schema and dataset parameters. The AWS IoT SiteWise Assistant uses datasets that consume Amazon Kendra indexes.

Data stream

Input, or ingest, industrial data into AWS IoT SiteWise even before creating asset models and assets. AWS IoT SiteWise automatically generates data streams to collect raw data streams from your equipment.

Data stream alias

Data stream aliases help you easily identify a data stream. For example, the alias `server1-windfarm/3/turbine/7/temperature` indicates temperature values coming from turbine #7 in wind farm #3. The term `server1` is the data source name that helps identify the OPC UA server, and `server1-` is a prefix attached to all data streams reported from this OPC UA server.

Data stream association

After you create asset models and assets, associate data streams with asset properties defined in your assets to structure your data. AWS IoT SiteWise can then use asset models and assets to handle incoming data from your data streams. You can also disassociate data streams from asset properties. For more information, see [Manage data streams for AWS IoT SiteWise](#).

Destinations

Destinations in SiteWise Edge represent the endpoints where you want to send your telemetry or processed data. SiteWise Edge supports the AWS IoT SiteWise hot tier, buffered ingestion, or an Amazon S3 bucket as destinations. You can configure destinations to subscribe to specific MQTT topics using path filters. For more information, see Understand AWS IoT SiteWise Edge destinations.

Formula

Each transform and metric property comes with a formula that outlines how the property transforms or aggregates data. These formulas include property inputs, operators, and functions offered by AWS IoT SiteWise. For more information, see Use formula expressions.

Interface

An interface is a type of model that defines a standard structure that can be applied to different asset models. For more information, see Asset model interfaces.

Measurement

Measurements are properties of an asset that depict the raw sensor time series data streams from a device or equipment. For more information, see Define data streams from equipment (measurements).

Metric

Metrics are properties of an asset that represent aggregated time series data. Each metric is accompanied by a mathematical expression (formula) that outlines how to aggregate data points and a time interval for computing that aggregation. Metrics generate a single data point for each specified time interval. For more information, see Aggregate data from properties and other assets (metrics).

MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol for sensors and devices.

Packs

SiteWise Edge gateways use packs to determine how to collect, process, and route data. For more information about the available packs for your SiteWise Edge gateway, see the section called “Use packs”.

Data collection pack

Use the data collection pack so that your SiteWise Edge gateway can collect your industrial data and route it to the AWS destination of your choice.

Data processing pack

Use the data processing pack to process, store, and retrieve your data at the edge for up to 30 days. Exchange edge-processed data to and from local applications through SiteWise Edge APIs.

OPC UA

OPC UA (Open Platform Communications Unified Architecture) is a communication protocol for industrial automation.

Path filters

Use path filters within a gateway to subscribe to MQTT topics and publish to AWS IoT SiteWise supported destinations. MQTT-based sources, data processing pipelines, and destinations all exchange data using MQTT topics on a self-hosted MQTT-enabled, V3 gateway. You can define topic filters to specify the data you want to ingest or route to different destinations.

Portal

An AWS IoT 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.

Property alias

You have the option to create aliases on asset properties, such as an OPC UA server data stream path (for example, /company/windfarm/3/turbine/7/temperature), simplifying the identification of an asset property during the ingestion or retrieval of asset data. When you use a SiteWise Edge gateway to ingest data from servers, your property aliases must match the paths of your raw data streams. For more information, see Manage data streams for AWS IoT SiteWise.

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 includes details about the update to that property value. Use property value notifications to create solutions that connect your industrial data in AWS IoT SiteWise with other AWS services. For more information, see Interact with other AWS services.

SiteWise Edge gateway

A SiteWise Edge gateway is installed on the customer's premises to gather, handle, and direct data. A SiteWise Edge gateway connects to your industrial data sources through various protocols to gather and process data, sending it to the AWS cloud. SiteWise Edge gateways can also connect to partner data sources. For more information, see Use AWS IoT SiteWise Edge gateways.

Transform

Transforms are properties of an asset that represent transformed time series data. Every transform is accompanied by a mathematical expression (formula) that specifies how to convert 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 Transform data (transforms).

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.

Get started with AWS IoT SiteWise

With AWS IoT SiteWise, you can collect, organize, analyze, and visualize 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 Use the AWS IoT SiteWise demo.

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

- Ingest data to AWS IoT SiteWise from AWS IoT things
- Visualize and share wind farm data in SiteWise Monitor
- Publish property value updates to Amazon DynamoDB

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

- Ingest data to AWS IoT SiteWise
- Model industrial assets
- Configure edge capabilities on AWS IoT SiteWise Edge
- Monitor data with AWS IoT SiteWise Monitor
- Query data from AWS IoT SiteWise
- Interact with other AWS services

Topics

Requirements

You must have an AWS account to get started with AWS IoT SiteWise. If you don't have one, see the following section for instructions on how to set up an account.

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.

Set up an AWS account

Topics

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

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

For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Use 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 asset models, assets, and a SiteWise Monitor portal, and generate sample data for up to a week.

> **Important**
>
> Once you create the demo, you will start being charged for the resources that this demo creates and consumes.

Topics

- Create the AWS IoT SiteWise demo
- Delete the AWS IoT SiteWise demo

Create the AWS IoT SiteWise demo

You can create the AWS IoT SiteWise demo from the AWS IoT SiteWise console.

> **Note**
>
> The demo creates Lambda functions, one CloudWatch Events rule, and the AWS Identity and Access Management (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. (Optional) To create a SiteWise Monitor portal to monitor sample data, do the following.

Note

You will be charged for the SiteWise Monitor resources that this demo creates and consumes. For more information, see SiteWise Monitor in the *AWS IoT SiteWise Pricing*.

a. Choose **Monitor Resources**.

b. Choose **Permission**.

c. Choose an existing IAM role that grants your federated IAM users access to the portal.

Important

Your IAM role must have the following permissions.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:Describe*",
                "iotsitewise:List*",
                "iotsitewise:Get*",
                "cloudformation:DescribeStacks",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedRolePolicies",
                "sso:DescribeRegisteredRegions",
                "organizations:DescribeOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about how to work with SiteWise Monitor, see What is AWS IoT SiteWise Monitor? in the *AWS IoT SiteWise Monitor Application Guide*.

4. 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.

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

Delete 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

Welcome to the AWS IoT SiteWise tutorials page. This growing collection of tutorials empowers you with the knowledge and skills needed to navigate the intricacies of AWS IoT SiteWise. These tutorials offer a diverse range of basic topics to cater to your needs. As you delve into the tutorials, uncover invaluable insights into various aspects of AWS IoT SiteWise.

Each tutorial uses a specific equipment example. These tutorials are intended for test environments, and they use fictitious company names, models, assets, properties, and so on. Their purpose is to provide general guidance. The tutorials are not intended for direct use in a production environment without careful review and adaptation to meet the unique needs of your organization.

Topics

Calculate 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 Ingest data to AWS IoT SiteWise. 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 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 (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 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 = Successes / (Successes + Failures)` – The machine's percentage of successfully filled packages over the specified time intervals.
- `Performance = ((Successes + Failures) / Run_Time) / 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.

> **Note**
>
> If OEE is defined as a transform, output values are computed for each of the input values. There is a potential to generate unexpected values as the transform evaluation considers the latest available values for all the contributing properties in the formula. For property updates with the same timestamp, output values may be overwritten by updates from other incoming properties. For example when Availability, Quality, and Performance are computed, the OEE is computed with the last available data points for the other two properties. These contributing values share timestamps, and cause incorrect output values of the OEE. The order is not guaranteed for transforms computation.

Ingest data to AWS IoT SiteWise from AWS IoT things

Learn how to ingest data to AWS IoT SiteWise from a fleet of AWS IoT things by using device shadows in this tutorial. *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 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 Ingest data to AWS IoT SiteWise using AWS IoT Core rules.

Topics

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Set up an AWS account.
- 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 AWS Identity and Access Management (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.

Step 1: Create an AWS IoT policy

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

Console

Use the following procedure to create an AWS IoT policy using the AWS IoT Core console:

To create an AWS IoT policy

1. Sign in to the AWS Management Console.
2. Review the AWS Regions where AWS IoT SiteWise is supported. Switch to one of these supported Regions, if necessary.
3. Navigate to the AWS IoT console. If a **Connect device** button appears, choose it.
4. In the left navigation pane, choose **Security** and then choose **Policies**.
5. Choose **Create**.
6. Enter a name for the AWS IoT policy (for example, **SiteWiseTutorialDevicePolicy**).
7. Under **Policy document**, choose **JSON** 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",
      "Resource": "arn:aws:iot:us-
east-1:123456789012:client/SiteWiseTutorialDevice*"
    },
    {
      "Effect": "Allow",
```

```
      "Action": "iot:Publish",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/delete",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/get"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/delete/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/get/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/delete/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/delete/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/get/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/delete/rejected"
      ]
    },
    {
      "Effect": "Allow",
```

```
        "Action": [
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:DeleteThingShadow"
        ],
        "Resource": "arn:aws:iot:us-
east-1:123456789012:thing/SiteWiseTutorialDevice*"

      }
    ]
  }
```

8. Choose **Create**.

AWS CLI

> **Important**
>
> This policy uses wildcards to stay within AWS IoT SiteWise CLI size limits. For more restrictive permissions with explicit topic paths, create the policy through the AWS IoT SiteWise console instead. For more information, see the IoT policy example provided on the tab.

Use the following AWS CLI command to create an IoT policy:

```
aws iot create-policy \
  --policy-name "SiteWiseTutorialDevicePolicy" \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": "iot:Connect",
        "Resource": "arn:aws:iot:region:account-id:client/SiteWiseTutorialDevice*"
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Publish", "iot:Receive"],
        "Resource": [
          "arn:aws:iot:region:account-id:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
```

```
        ]
      },
      {
        "Effect": "Allow",
        "Action": "iot:Subscribe",
        "Resource": [
          "arn:aws:iot:region:account-id:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:DeleteThingShadow"
        ],
        "Resource": "arn:aws:iot:region:account-id:thing/SiteWiseTutorialDevice*"
      }
    ]
  }'
```

To verify that your policy was created successfully, use the following command:

```
aws iot get-policy --policy-name "SiteWiseTutorialDevicePolicy"
```

This policy enables your AWS IoT devices to establish connections and communicate with device shadows using MQTT messages. For more information about MQTT messages, see [What is MQTT?](). To interact with device shadows, your AWS IoT things publish and receive MQTT messages on topics that start with `$aws/things/thing-name/shadow/`. This policy incorporates a thing policy variable known as `${iot:Connection.Thing.ThingName}`. This variable substitutes the connected thing's name in each topic. The `iot:Connect` statement sets limitations on which devices can establish connections, ensuring that the thing policy variable can only substitute names starting 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.

Step 2: Create and configure an AWS IoT thing

In this procedure, you create and configure an AWS IoT thing. You can designate your development computer as an AWS IoT thing. As you progress, remember that the principles you're learning here can be applied to actual projects. You have the flexibility to make and set up AWS IoT things on any device capable of running an AWS IoT SDK, including AWS IoT Greengrass and FreeRTOS. For more information, see AWS IoT SDKs in the *AWS IoT Developer Guide*.

Console

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.

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

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

```
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 **All devices** in the **Manage** section. Then choose **Things**.
5. If a **You don't have any things yet** dialog box appears, choose **Create a thing**. Otherwise, choose **Create things**.
6. On the **Creating things** page, choose **Create a single thing** and then choose **Next**.

7. On the **Specify thing properties** 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 in *Step 1: Creating an AWS IoT policy*. Otherwise, your device can't connect to AWS IoT.

8. On the **Configure device certificate - *optional*** page, choose **Auto-generate a new certificate (recommended)**, then choose **Next**. Certificates enable AWS IoT to securely identify your devices.

9. On the **Attach policies to certificate - *optional*** page, select the policy you created in *Step 1: Creating an AWS IoT policy* and choose **Create thing**.

10. On the **Download certificates and keys** dialog box, do the following:

    a. 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.

    b. Choose the **Download** link to 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 **Done**.

AWS CLI

Follow these steps to create and configure an AWS IoT thing using the AWS CLI:

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

```
mkdir iot-sitewise-rule-tutorial
```

2. Navigate to the tutorial directory:

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

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

```
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.

4. Create an AWS IoT thing:

```
aws iot create-thing --thing-name "SiteWiseTutorialDevice1"
```

> **Important**
>
> The thing name must match the name pattern used in the policy that you created in Step 1. Otherwise, your device can't connect to AWS IoT.

5. Create a certificate and save the files. Note the certificate ARN from the output - you'll need it in the next steps:

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "device1/device.pem.crt" \
    --public-key-outfile "device1/public.pem.key" \
    --private-key-outfile "device1/private.pem.key"
```

6. Attach the policy you created in Step 1 to the certificate:

```
aws iot attach-policy \
    --policy-name "SiteWiseTutorialDevicePolicy" \
    --target "certificate-arn"
```

7. Attach the certificate to the thing:

```
aws iot attach-thing-principal \
```

```
    --thing-name "SiteWiseTutorialDevice1" \
    --principal "certificate-arn"
```

8. Download the Amazon root CA certificate:

```
curl https://www.amazontrust.com/repository/AmazonRootCA1.pem >
 AmazonRootCA1.pem
```

This certificate is required for your device to successfully connect to AWS IoT.

> **Important**
>
> Store your certificates and keys securely. You cannot download these credentials again after creating them.

You have now registered an AWS IoT thing on your computer. Take one of the following next steps:

- Continue to *Step 3: Creating a device asset model* 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.

Step 3: Create 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. To process data in assets that represent groups of devices, asset models enforce consistent information across multiple assets of the same type. For more information, see Model industrial assets.

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 asset model**.
4. Under **Model details**, enter a name for your model. 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 [Define data streams from equipment (measurements)](#).
6. Choose **Add new 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 [Aggregate data from properties and other assets (metrics)](#).
9. Choose **Add new 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 Use formula expressions. In *Step 4: Creating a device fleet asset model*, you create a parent asset that can calculate metrics using data from your entire fleet of devices.
12. Choose **Create model**.

Step 4: Create a device fleet asset model

In this procedure, you craft an asset model in AWS IoT SiteWise to symbolize your collection of devices. Within this asset model, you establish a structure that allows you to link numerous device assets to one overarching fleet asset. Following that, you outline metrics in the fleet asset model to consolidate data from all connected device assets. This approach provides you with comprehensive insights into the collective performance of your entire fleet.

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 asset model**.
4. Under **Model details**, enter a name for your model. For example, **`SiteWise Tutorial Device Fleet Model`**.
5. Under **Hierarchy definitions**, do the following:

   a. In **Hierarchy name**, enter **`Device`**.

   b. 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 [Asset hierarchies represent equipment relationships](#).

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

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

   b. In **Formula**, enter **`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 new 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 **`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**.

Step 5: Create and configure a device asset

In this procedure, you generate a device asset that's based on your device asset model. Then, you define property aliases for each measurement property. A *property alias* is a unique string that identifies an asset property. Later, you can identify a property for data upload by using the aliases instead of the asset ID and property ID. For more information, see [Manage data streams for AWS IoT SiteWise](#).

To create a device asset and define property aliases

1. Navigate to the [AWS IoT SiteWise console](#).
2. In the left navigation pane, choose **Assets**.
3. Choose **Create asset**.
4. Under **Model information**, choose your device asset model, **`SiteWise Tutorial Device Model`**.
5. Under **Asset information**, enter a name for your asset. For example, **`SiteWise Tutorial Device 1`**.
6. Choose **Create asset**.
7. For your new device asset, choose **Edit**.
8. Under **Measurements**:

    a. 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.

    b. Under **Memory Usage**, enter **`/tutorial/device/SiteWiseTutorialDevice1/memory`** as the property alias.
9. Choose **Save**.

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`**.

Step 6: Create and configure a device fleet asset

In this procedure, you form a device fleet asset derived from your device fleet asset model. Then, you link your individual device assets to the fleet asset. This association enables the metric properties of the fleet asset to compile and analyze data from multiple devices. This data provides you with a consolidated view of the collective performance of the entire fleet.

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. Under **Model information**, choose your device fleet asset model, **`SiteWise Tutorial Device Fleet Model`**.
5. Under **Asset information**, enter a name for your asset. For example, **`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** and do the following:

   a. 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.

   b. Under **Asset**, choose your device asset, **SiteWise Tutorial Device 1**.
9. (Optional) If you created multiple device assets earlier, repeat steps 8 through 10 for each device asset that you created.
10. Choose **Save**.

    You should now see your device assets organized as a hierarchy.

Step 7: Create a rule in AWS IoT Core to send data to device assets

In this procedure, you establish a rule in AWS IoT Core. The rule is designed to interpret notification messages from device shadows and transmit the 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, the aim is to handle the update message, extract the property values, and transmit 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 **Message routing** and then choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for your rule and the choose **Next**.

5. Enter the following SQL statement and the choose **Next**.

```
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 **Rule actions**, choose **Send message data to asset properties in AWS IoT SiteWise** and do the following:

   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 - *optional***, 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 - *optional***, enter **`${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 **`${floor(timestamp() / 1E3)}`**.
   > - In **Offset in nanos**, enter **`${(timestamp() % 1E3) * 1E6}`**.

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

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

g. In **Value**, enter **`${state.reported.cpu}`**. In substitution templates, you use the `.` operator to retrieve a value from within a JSON structure.

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

   i. Choose **By property alias**.

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

iii. In **Entry ID - *optional***, enter **`${concat(topic(3), "-memory-", floor(state.reported.timestamp))}`**.

iv. In **Time in seconds**, enter **`${floor(state.reported.timestamp)}`**.

v. In **Offset in nanos - *optional***, enter **`${floor((state.reported.timestamp % 1) * 1E9)}`**.

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

vii. In **Value**, enter **`${state.reported.memory}`**.

i. Under **IAM Role**, choose **Create new 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.

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

7. (Optional) Configure an error action that you can use to troubleshoot your rule. For more information, see Troubleshoot a rule (AWS IoT SiteWise).
8. Choose **Next**.
9. Review the settings and choose **Create** to create the rule.

Step 8: Run the device client script

For this tutorial, you aren't using an actual device to report data. Instead, 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.

   ```
   import AWSIoTPythonSDK.MQTTLib as AWSIoTPyMQTT

   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",
    )
    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 {}...
\n".format(self.thingName))
        while True:
            performance = self.readPerformance()
            print("[{}]".format(self.thingName))
            print("CPU:\t{}%".format(performance["cpu"]))
            print("Memory:\t{}%\n".format(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 = AWSIoTPyMQTT.AWSIoTMQTTShadowClient(self.thingName)
        mqttClient.configureEndpoint(self.host, self.port)
        mqttClient.configureCredentials(
            self.rootCAPath, self.privateKeyPath, self.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 {} {}\n".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](https://console.aws.amazon.com/iot/).
    b. In the left navigation pane, choose **All devices** and then choose **Things**.
    c. Choose your thing, **SiteWiseTutorialDevice**.
    d. Choose the **Device Shadows** tab, choose **Classic Shadow**, and verify that the **Shadow state** looks like the following example.

```
{
  "reported": {
    "cpu": 24.6,
    "memory": 85.2,
    "timestamp": 1579567542.2835066
  }
}
```

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 is configured according to this tutorial.

11. Follow these steps to verify that the rule action is sending data to AWS IoT SiteWise:

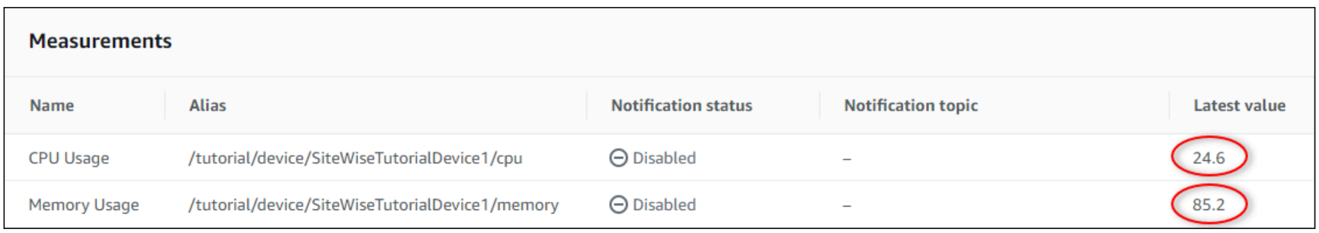
    a. Navigate to the AWS IoT SiteWise console.

    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.



Measurements

Name	Alias	Notification status	Notification topic	Latest value
CPU Usage	/tutorial/device/SiteWiseTutorialDevice1/cpu	Disabled	–	24.6
Memory Usage	/tutorial/device/SiteWiseTutorialDevice1/memory	Disabled	–	85.2

    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 Troubleshoot a rule (AWS IoT SiteWise).

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 Monitor data with AWS IoT SiteWise Monitor. 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.

Step 9: Clean up resources after the tutorial

> **Note**
>
> The resources created in this tutorial are required for the Integrate data into SiteWise Edge tutorial. Do not clean up the resources in this step if you plan on completing it.

After you complete the tutorial about ingesting data from AWS IoT things, 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**.

      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**.
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]. 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 deleting hierarchical asset models, start by deleting the parent asset model first.
5. Choose **Delete**.
6. In the confirmation dialog, enter `Delete` and then choose **Delete**.

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 **Message routing** and then choose **Rules**.
3. Select your rule and choose **Delete**.
4. In the confirmation dialog, enter the name of the rule and then choose **Delete**.

Integrate data into SiteWise Edge using an MQTT-enabled, V3 gateway

This tutorial guides you through integrating third-party devices and sensors that use MQTT messaging protocol with the AWS IoT SiteWise MQTT-enabled, V3 gateway. You will learn how to set up an AWS IoT SiteWise edge gateway to collect and monitor data from your MQTT-enabled devices. AWS IoT SiteWise enables you to collect, process, and monitor industrial equipment data. Use SiteWise Edge capabilities to optimize industrial IoT operations, and transform raw data into actionable insights.

In this tutorial, we use data from a wind farm demonstration to illustrate key concepts. After you become familiar with the process, you can repeat the tutorial with your own data.

After you complete this tutorial, you can do the following items:

- Set up and configure an MQTT-enabled, V3 gateway to receive data from industrial devices
- Process and validate incoming MQTT messages from your equipment at the edge
- View device data in AWS IoT SiteWise using a third-party visualization platform
- Send processed data from your edge gateway to the AWS Cloud to enable centralized storage and further analysis

Additionally, you can leverage your edge gateway capabilities by connecting to other AWS IoT services to perform the following tasks:

- Configure AWS IoT rules to route data to services like Amazon S3, Amazon Timestream, and AWS Lambda.

- Use AWS IoT Device Defender to remotely manage and update your gateway configurations.
- Implement secure device authentication and authorization using AWS IoT security features. For more information, see AWS IoT security in the *AWS IoT Developer Guide*.
- Create automated alerts and notifications based on equipment data. For more information, see Rules for AWS IoT in the *AWS IoT Developer Guide*.

> **Note**
>
> This tutorial references third-party services, tools, and documentation. AWS isn't a vendor or supplier for any third-party products or services, and can't guarantee the accuracy of information from external providers. Evaluate and validate all third-party tools before deployment.

Topics

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Set up an AWS account.
- An AWS Identity and Access Management (IAM) user with administrator permissions. For more information, see Identity and access management for AWS IoT SiteWise.
- The latest version of Python installed on your device.

> **Important**
>
> This tutorial requires the use of resources created in the Ingest data tutorial. You must complete it before proceeding with this tutorial.

Step 1: Create an AWS IoT policy

This tutorial uses the AWS IoT policy you created in the Ingest data tutorial. This policy sets the security rules for your devices and creates a digital representation of your external devices and sensors in AWS IoT. The policy allows your third-party devices to send data to AWS IoT Core using MQTT (Message Queuing Telemetry Transport). For more information about MQTT messages, see What is MQTT?.

Console

Ensure completion of an AWS IoT policy. For detailed instructions, see Step 1 in the Ingest data tutorial.

To verify you have an active AWS IoT policy

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose **Securities**, then **Policies**.
3. Choose the policy you created. For example, **`SiteWiseTutorialDevicePolicy`**.
4. Confirm that the policy's status is listed as Active.

AWS CLI

Ensure completion of an AWS IoT policy. For detailed instruction, see Step 1 in the Ingest data tutorial.

Use the following AWS CLI get-policy command in the *AWS CLI Command Reference* to verify you have an active AWS IoT policy:

```
aws iot get-policy --policy-name "SiteWiseTutorialDevicePolicy"
```

This policy enables your AWS IoT devices to establish connections and to communicate with device shadows using MQTT messages. To interact with device shadows, your AWS IoT things publish and receive MQTT messages on topics that start with `$aws/things/thing-name/shadow/`. This policy incorporates a thing policy variable known as `${iot:Connection.Thing.ThingName}`. This variable substitutes the connected thing's name in each topic. The `iot:Connect` statement sets limitations on which devices can establish connections, ensuring that the thing policy variable can only substitute names starting 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.

Step 2: Create and configure an AWS IoT thing

In this step, register your edge device as an AWS IoT thing and generate your thing's certificates and keys needed for secure communication with AWS IoT SiteWise Edge. This process establishes the foundation for your device to send third-party data through your MQTT-enabled, V3 gateway.

Console

Ensure completion of the creation and configuration steps for an AWS IoT thing. For detailed instructions, see Step 2 in the Ingest data tutorial.

To verify you have an active AWS IoT thing

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose **All devices**, then **Things**.
3. Choose the thing you created. For example, **`SiteWiseTutorialDevice1`**.
4. Under **Certificates**, confirm that the status is listed as active.

AWS CLI

Ensure completion of the creation and configuration steps for an AWS IoT thing. For detailed instructions, see Step 2 in the Ingest data tutorial.

Use the following AWS CLI command to verify you have an active AWS IoT policy:

```
aws iot describe-thing --thing-name "SiteWiseTutorialDevice1"
```

After completing these steps, you can securely connect your device to AWS IoT SiteWise Edge. You created a local directory to store your certificates and keys you generated for MQTT authentication. Your device is registered as an AWS IoT thing in the AWS IoT console, and your device is prepared to integrate data with SiteWise Edge. You can connect your industrial equipment or other devices to the AWS IoT platform and start ingesting data into SiteWise Edge.

Step 3: Configure your SiteWise Edge MQTT-enabled, V3 gateway

In this step, create your AWS IoT SiteWise Edge MQTT-enabled, V3 gateway and configure it to receive data from the EMQX broker. The gateway acts as a bridge between your devices and AWS IoT. This allows you to process data locally at the edge before sending it to the AWS Cloud. This configuration reduces bandwidth and decreases cloud processing delays.

Console

To create your AWS IoT SiteWise MQTT-enabled, V3 gateway

1. Sign in to the AWS Management Console and open the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Edge gateways**, then choose **Create gateway**.
3. Under **Deployment target**, choose **Self-hosted gateway**.
4. Under Self-hosted gateway options, choose **MQTT-enabled, V3 gateway - *recommended***.
5. Under **Gateway configuration**:
   a. In **Gateway name**, enter a name for your gateway. For example, **`SiteWise Tutorial Device Gateway`**.
   b. In **Greengrass device OS**, select the appropriate option for your device.
6. Under **Advanced configuration**:
   a. Choose **Default setup**.

   b. Enter a name for the Greengrass core device or use the name generated by AWS IoT SiteWise.

7. Choose **Create gateway**.
8. In the confirmation dialog, choose **Generate and download** to generate an installer for your SiteWise Edge gateway. For more information, see [Create a self-hosted SiteWise Edge gateway](#).

> **Warning**
>
> Store the installer file in a secure location. This file can't be regenerated, and is needed to complete the gateway setup in later steps.

AWS CLI

Use AWS CLI to create a self-hosted gateway. You need to provide a name for the gateway, specify the platform and gateway version. For more information, see [CreateGateway](#) in the *AWS IoT SiteWise API Reference*.

To use this example, replace the user input placeholders with your own information.

```
aws iotsitewise create-gateway \
    --gateway-name SiteWise Tutorial Device Gateway \
    --gateway-platform greengrassV2={coreDeviceThingName=your-core-device-thing-name, coreDeviceOperatingSystem=LINUX_AMD64} \
    --gateway-version 3 \
    [--cli-input-json your-configuration]
```

- `gateway-name` – A unique name for the gateway, for example, `SiteWise Tutorial Device Gateway`.
- `gateway-platform` – Enter `greengrassV2`. For more information, see [CreateGateway](#) in the *AWS IoT SiteWise API Reference*.
  - `coreDeviceThingName` – The name of the AWS IoT thing for your AWS IoT Greengrass V2 core device. For example, `SiteWiseTutorialDevice1`.
  - `coreDeviceOperatingSystem` – The operating system of the core device in AWS IoT Greengrass V2. Specifying the operating system is required for gateway-version 3. Options include: `LINUX_AARCH64`, `LINUX_AMD64`, and `WINDOWS_AMD64`.

- `gateway-version` – The version of the gateway.
  - Use 3 for the gateway version to create an MQTT-enabled, V3 gateway.
- `cli-input-json` – A JSON file containing request parameters.

Use the following AWS CLI command to verify that your gateway was created successfully:

```
aws iotsitewise describe-gateway --gateway-id your-gateway-id
```

Step 4: Install SiteWise Edge gateway software

To install the gateway software, use the installer package that you downloaded in the previous step. The installation process configures the necessary components, starts the Greengrass core service, and registers your device with AWS IoT Greengrass. After installation is complete, verify that your gateway appears in the AWS IoT SiteWise console under **Edge gateways** and that the Greengrass service is running properly on your device.

For detailed instructions, see Install the AWS IoT SiteWise Edge gateway software on your local device.

Step 5: Configure the EMQX broker to connect to external applications

> **Note**
>
> You must have deployed your SiteWise Edge MQTT-enabled, V3 gateway before proceeding. The gateway provides the necessary infrastructure and security settings required for configuring the EMQX broker. The broker configuration will fail without an active gateway deployment.

Configure the EMQX broker to enable secure communication between your IoT devices and external applications. The EMQX broker functions as a central messaging hub that routes data between your IoT devices, gateway, and applications. The EMQX broker ensures reliable message delivery on your gateway and connected applications at the edge. For more information, see Connect external applications to the EMQX broker.

To configure the EMQX broker

1. Set up the EMQX broker. For detailed configuration instructions, follow Steps 1-14 in Update the EMQX deployment configuration for authentication.
2. Set up MQTT topics for wind farm monitoring. For more information on MQTT requirements, see MQTT topic requirements.
   a. CPU Usage: `SiteWiseTutorialDevice/cpu`
   b. Memory Usage: `SiteWiseTutorialDevice/memory`
   c. Timestamp: `SiteWiseTutorialDevice/timestamp`
3. Review your configuration and complete the deployment.
   a. Choose **Confirm** to save your settings.
   b. Choose **Next** until you reach the **Review** step.
   c. On the **Review** page, choose **Deploy**.
   d. Wait for the deployment to complete successfully before proceeding.
4. Prepare messages using the payload format to send to the EMQX broker. For more information about structuring payloads, see Update the EMQX deployment configuration for authentication.
5. Implement the following security measures:
   a. Use Transport Layer Security (TLS) encryption (port 8833) to protect data in transit. For more information, see Configure TLS for secure connections to the EMQX broker on AWS IoT SiteWise Edge.
   b. Set up username and password authentication to verify device identities. This security measure helps protect your data, and ensures only authorized devices can connect to your system. For more information, see Enable username and password authentication.

EMQX allows you to create authorization rules based on identifiers such as username, IP address, or client ID. This is useful for controlling access to your data. For more information, see Set up authorization rules for AWS IoT SiteWise Edge in EMQX.

After successful deployment, your EMQX broker is configured and ready to securely connect with external applications.

Note

The payload format must follow a specific structure for AWS IoT SiteWise Edge to properly process and ingest your data. For more information about the required structure, see JSON payload structure.

Example: Add CPU, memory, and timestamp JSON payloads

CPU JSON payload

```
{
  "propertyAlias": "SiteWiseTutorialDevice/cpu",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "offsetInNanos": 0,
        "timeInSeconds": 1753206441
      },
      "value": {
        "integerValue": 45.2
      }
    }
  ]
}
```

Memory JSON payload

```
{
  "propertyAlias": "SiteWiseTutorialDevice/memory",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "offsetInNanos": 0,
        "timeInSeconds": 1753206441
      },
      "value": {
        "integerValue": 67.8
      }
    }
```

```
  ]
}
```

Timestamp JSON payload


```
{
  "propertyAlias": "SiteWiseTutorialDevice/timestamp",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "offsetInNanos": 0,
        "timeInSeconds": 1753206441
      },
      "value": {
        "integerValue": 23.5
      }
    }
  ]
}
```


Note

Each JSON payload must be published separately as an individual message. Don't combine multiple property values into a single message. Send each CPU, memory, and timestamp payload as its own distinct MQTT publication.

The payload defines the required JSON structure that your IoT devices must use to send device data through the EMQX broker to SiteWise Edge. This format ensures that AWS IoT SiteWise can identify your devices and process the sensor readings. After you implement these configurations and payload structures, your wind farm monitoring system is ready to collect and process data.

Step 6: Publish data with Mosquitto

After creating your MQTT-enabled, V3 gateway, configure Eclipse Mosquitto to send test data to SiteWise Edge. Mosquitto is an open-source MQTT message broker that uses the MQTT protocol for lightweight messaging between devices. The Mosquitto client allows you to publish messages to MQTT topics, simulating data from wind farm sensors. Using Mosquitto, simulate device data without requiring any third-party services or additional equipment. For more information, see documentation on the official Eclipse Mosquitto website. In this tutorial, local data from the Ingest data tutorial and fictitious data are being used for demonstration purposes.

Use Mosquitto CLI client to test the SiteWise Edge EMQX broker

1. Install Mosquitto on your local device. For detailed instructions, see Download Mosquitto on the official Eclipse Mosquitto website.
2. For more information about connecting external applications to transfer industrial data, see Connect external applications to the EMQX broker.

> **Important**
>
> Ensure that the MQTT connection settings you configure here match the settings used in Mosquitto publish command. The host must be the IP address or hostname of your SiteWise Edge gateway. The port is typically 1883 (or 8883 if using SSL/TLS).

Use Mosquitto to publish test data. Open a command line and run the following commands:

Example: CPU property

```
mosquitto_pub -h localhost -p 1883 -t "SiteWiseTutorialDevice/cpu" -m '{
  "propertyAlias": "SiteWiseTutorialDevice/cpu",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "timeInSeconds": 1753206441,
        "offsetInNanos": 0
      },
      "value": {
        "integerValue": 45.2
      }
    }
  ]
}'
```

Example: Memory property

```
mosquitto_pub -h localhost -p 1883 -t "SiteWiseTutorialDevice/memory" -m '{
```

```
  "propertyAlias": "SiteWiseTutorialDevice/memory",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "timeInSeconds": 1753206441,
        "offsetInNanos": 0
      },
      "value": {
        "integerValue": 72.1
      }
    }
  ]
}'
```

Example: Timestamp property

```
mosquitto_pub -h localhost -p 1883 -t "SiteWiseTutorialDevice/timestamp" -m '{
  "propertyAlias": "SiteWiseTutorialDevice/timestamp",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "timeInSeconds": 1753206441,
        "offsetInNanos": 0
      },
      "value": {
        "integerValue": 1683000000
      }
    }
  ]
}'
```

Note

The use of `localhost` as the EMQX broker address is for demonstration purposes only. In production environments or when connecting from external devices, you must use the appropriate EMQX broker address for your specific deployment configuration. For detailed connection instructions, see [Connect an application to the EMQX broker on AWS IoT SiteWise Edge](#).

Step 7: Specify destinations

In this step, specify destinations to determine where to direct your source data. Use AWS IoT SiteWise with Amazon S3 buffering as your destination. This option provides a scalable way to store and process your IoT data.

Console

To add destinations

1. Navigate to the AWS IoT SiteWise console and select **Edge gateways**.
2. Under **SiteWise Tutorial Device Gateway**, choose **Add destinations**.
3. Under **Destination details**, choose **AWS IoT SiteWise buffered using Amazon S3**. To learn more about destination types, see AWS IoT SiteWise gateway destinations.
4. Under **Destination name**, enter a name for your destination, for example, `SiteWise Tutorial S3 Destination`.
5. Under **S3 upload settings**, enter your S3 bucket location. For example, `s3://sitewise-tutorial-mqtt-data-[your-account-id]`. To learn more about Amazon S3, see Creating, configuring, and working with Amazon S3 buckets in the *Amazon Simple Storage Service User Guide*.
6. Under **Data upload frequency**, enter a value between 1 minute and 30 days. For example, `1 minute`.
7. Under **Data storage settings**:
   - Deselect **Copy data to storage**. While this setting is recommended for production environments, you don't need it for this tutorial. When you deselect this option, the **Delete data from S3** option is automatically deselected.
8. Choose **Add destination**.

> **Note**
>
> This tutorial uses a 1-minute interval for testing. After you complete the tutorial, you can adjust this interval to match your production needs or delete it to avoid additional charges.

AWS CLI

Example: Create a new AWS IoT SiteWise destination buffered using Amazon S3

Use the update-gateway-capability-configuration in the *AWS CLI Command Reference* to configure the publisher. Set the `capabilityNamespace` parameter to `iotsitewise:publisher:3`.

```
{
    "sources": [
      {
        "type": "MQTT"
      }
    ],
    "destinations": [
      {
        "type": "SITEWISE_BUFFERED",
        "name": "your-s3-destination-name",
        "config": {
          "targetBucketArn": "arn:aws:s3:::amzn-s3-demo-bucket/Optional/SomeFolder",
          "publishPolicy": {
            "publishFrequency": "1m",
            "localSizeLimitGB": 10
          },
          "siteWiseImportPolicy": {
            "enableSiteWiseStorageImport": true,
            "enableDeleteAfterImport": true,
            "bulkImportJobRoleArn": "arn:aws:iam::123456789012:role/your-role-name"
          }
        },
        "filters": [
          {
            "type": "PATH",
            "config": {
              "paths": [
                "#"
              ]
            }
          }
        ]
      }
    ]
  }
```

For more information about destinations, see Add an AWS IoT SiteWise buffered destination using Amazon S3.

Step 8: Specify path filters

In this step, configure path filters to specify which MQTT topics to monitor for your wind farm device data.

Path filters follow the MQTT topic wildcard specification, which supports two special characters:

- + – This symbol represents a single-level wildcard, which matches any string at a single level.
- # – This symbol represents a multi-level wildcard, which matches any number of levels in the topic hierarchy.

> **Note**
>
> For more information about other path filters, see Special characters in path filter names.

Console

To configure your path filters

Under **Path filters**:

1. Navigate to the AWS IoT SiteWise console and select **Edge gateways**.
2. Under **SiteWise Tutorial Device Gateway**, choose **Add destinations**.
3. Choose **Add path filters** to enter the following path filters manually:
    - `SiteWiseTutorialDevice/#`
    - `windfarm/+/turbine/+/performance/#`
    - `cpu/+/idle-time`
    - `cpu/+/interruption-count/+`
    - `+/memory/consumption`
    - `timestamp/+/measurement`

- `device/+/status/+`
- `system/+/performance-log`

4. Choose **Add destination**.

For more information about best practices for path filters, see Best practices for path filters.

AWS CLI

Use the following AWS CLI commands to configure your path filters:

Example 1: Device data using wildcard

```
{
 "destinations": [
   {
     "name": "All Device Data Destination"
   }
 ],
 "filters": [
   {
     "type": "PATH",
     "config": {
       "paths": [
         "SiteWiseTutorialDevice/#",
         "windfarm/+/turbine/+/performance/#"
       ]
     }
   }
 ]
}
```

This path filter configuration uses multi-level wildcards (#) to capture all data from the SiteWiseTutorialDevice and all performance data from any turbine in the wind farm.

Example 2: CPU and memory performance

```
{
 "destinations": [
   {
     "name": "Performance Metrics Destination"
   }
 ],
```

```
  "filters": [
    {
      "type": "PATH",
      "config": {
        "paths": [
          "cpu/+/idle-time",
          "+/memory/consumption",
          "cpu/+/interruption-count/+"
        ]
      }
    }
  ]
}
```

This example captures various CPU metrics (idle time and interruption count) and memory consumption data across devices.

Example 3: Device diagnostics

```
{
  "destinations": [
    {
      "name": "Device Diagnostics Destination"
    }
  ],
  "filters": [
    {
      "type": "PATH",
      "config": {
        "paths": [
         "device/+/status/+",
         "system/+/performance-log"
        ]
      }
    }
  ]
}
```

This configuration uses the + wildcard to capture diagnostic data from multiple devices, specifically system performance logs and device status updates.

These three path filters match the MQTT topics that you use to publish test data with Mosquitto. The filters ensure your SiteWise Edge gateway captures and processes the relevant

MQTT messages. For more information on how to add path filters, see Add path filters to AWS IoT SiteWise Edge destinations.

Step 9: Configure your AWS IoT resources

In this step, create the necessary AWS IoT SiteWise asset models and assets to represent your simulated third-party devices and enable data ingestion through your edge gateway.

Before starting this step, you should have completed steps 3 to 8 in the Ingest data tutorial. These steps establish the foundational components to integrate your third-party data through the MQTT-enabled V3 gateway. You also set up rules that define how your sensor data flows into AWS IoT SiteWise, and run a device client script that simulates industrial wind farm data.

To validate your AWS IoT resource configuration

1. Use the following AWS CLI command to verify you created and properly configured your SiteWise Tutorial Device Model and SiteWise Tutorial Device Fleet Model:

   ```
   aws iotsitewise describe-asset-model --asset-model-id your-device-model-id
   ```

   Use the following AWS CLI command to retrieve your asset models' ID:

   ```
   aws iotsitewise list-asset-models
   ```

2. Use the following AWS CLI command to verify you created and properly configured your SiteWise Tutorial Device 1 asset and SiteWise Tutorial Device Fleet 1 asset:

   ```
   aws iotsitewise describe-asset --asset-id your-asset-id
   ```

   Use the following AWS CLI command to retrieve your assets' ID:

   ```
   aws iotsitewise list-assets
   ```

Step 10: Visualize your data

Set up the open-source version of Grafana to visualize your wind farm device data. Grafana is a visualization platform that displays your real-time operational data. These dashboards help you track operational efficiency and identify maintenance needs across your infrastructure. For more information about integration, see Integrate AWS IoT SiteWise with Grafana.

To setup Grafana

1. For instructions to download and install the latest version of Grafana, see Install Grafana on the official Grafana website.
2. For detailed configuration instructions specific to your operating system, see Configure Grafana on the official Grafana website.
3. Configure the AWS IoT SiteWise data source. This allows you to set up the AWS IoT SiteWise plugin on your Grafana server. For detailed instructions about how to use the plugin, see Connect to an AWS IoT SiteWise data source in the *Amazon Managed Grafana User Guide*.

> **Important**
>
> Ensure you have the latest version of Grafana for compatibility with the AWS IoT SiteWise data source.

After completing these steps, you can build and customize Grafana dashboards to display your wind farm's operational metrics. This enables you to track and analyze your wind farm performance at the edge in real time.

> **Note**
>
> While this tutorial uses the open-source version of Grafana, AWS also offers Amazon Managed Grafana for production environments. Amazon Managed Grafana is a fully managed service that eliminates the need to set up, configure, and maintain your own Grafana servers.
> Consider upgrading to Amazon Managed Grafana when you're ready to scale your solution. For detailed instructions on how to connect your SiteWise data to Grafana, see the Visualize and share data in Grafana tutorial.

You have completed the tutorial. In this procedure, you configured AWS IoT SiteWise Edge to integrate third-party device data using an MQTT-enabled, V3 gateway. This setup allows you to collect, process, and visualize industrial equipment data at the edge, reducing latency and operational costs. By using the wind farm demo, you collected and processed operational metrics like CPU and memory usage data through your MQTT-enabled, V3 gateway.

To enhance your IoT solution, consider exploring advanced features like anomaly detection by leveraging [Detect anomalies with Lookout for Equipment], or integrating with other AWS services like [Amazon QuickSight] in the *Amazon QuickSight User Guide* for advanced analytics.

Step 11: Clean up resources after the tutorial

After you complete this tutorial about integrating data into AWS IoT SiteWise Edge, 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**.

   > **Note**
   >
   > The device assets are no longer organized as a hierarchy now.

4. Choose your device asset (**SiteWise Tutorial Device 1**).
5. Choose **Delete**.
6. In the confirmation dialog, enter **`Delete`**, and then choose **Delete**.
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. Delete your device and device fleet assets.
3. In the left navigation pane, choose **Models**.
4. Choose your device fleet asset model (**SiteWise Tutorial Device Fleet Model**). You can't delete a model if you have assets that were created from that model.

   When deleting hierarchical asset models, start by deleting the parent asset model first.
5. Choose **Delete**.
6. In the confirmation dialog, enter **`Delete`**, and then choose **Delete**.
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 **Message routing**, and then choose **Rules**.
3. Select your rule and choose **Delete**.
4. In the confirmation dialog, enter the name of the rule and then choose **Delete**.

To delete an Amazon S3 bucket

1. Navigate to the Amazon S3 console.
2. In the left navigation pane, choose **General purpose bucket**.
3. In the buckets list, select the option button next to the bucket you created, and then choose **Empty** at the top of the page.
4. In the confirmation dialog, confirm the deletion, and then choose **Empty**.
5. After the bucket is empty, choose **Delete** to delete the bucket.
6. In the confirmation dialog, enter the name of your bucket to confirm deletion.
7. Choose **Delete bucket**.

To delete a SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.

2. In the left navigation pane, choose **Edge gateways**.
3. Under Gateways, choose the gateway you created for this tutorial. For example, `SiteWise Tutorial Device Gateway`.
4. Choose **Delete**.
5. To confirm you want to delete the gateway, type **`Delete`** in the confirmation dialog, and then choose **Delete** in the window that appears.

To delete your IoT thing

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose **Manage**, then choose **Things**.
3. Select the IoT thing you created for this tutorial. For example, `SiteWiseTutorialDevice1`.
4. Choose **Delete**.
5. In the confirmation dialog, enter the name of the thing, and then choose **Delete**.

To uninstall AWS IoT Greengrass Core

Uninstall the AWS IoT Greengrass Core software from your local device. For detailed instructions, see Uninstall the AWS IoT Greengrass Core software in the *AWS IoT Greengrass Developer Guide, Version 2*.

> **Important**
>
> Uninstalling Greengrass removes all local configurations and data. Ensure you've backed up any important information before proceeding.

(Optional) To delete third-party resources

After completing this tutorial, consider shutting down any external resources you created. This helps to prevent incurring charges from third-party providers.

Additional resources

Refer to the following resources for more information:

- Interact with other AWS services
- Use AWS IoT SiteWise Edge gateways
- Troubleshooting a SiteWise Edge gateway
- Security best practices for AWS IoT SiteWise
- AWS IoT pricing
- Ingest data to AWS IoT SiteWise
- Use tags in AWS IoT SiteWise

Visualize and share data in Grafana

This tutorial guides you through configuring the AWS IoT SiteWise data source plugin with Grafana, a data visualization platform. With Grafana, you can create dashboards that visualize and monitor your industrial data. In this tutorial, a sample dataset from a wind farm demonstration is used to illustrate key concepts. After you become familiar with the process, you can repeat the tutorial with your own data.

After you complete this tutorial, you can do the following:

- Collect, query, and analyze data from industrial equipment
- Create interactive Grafana dashboards to visualize asset performance metrics
- Monitor operational data through a unified interface
- Share insights with your team using Grafana's collaboration features
- Combine AWS IoT SiteWise data with other AWS data sources such as Amazon CloudWatch or Amazon Timestream

Topics

- Additional resources

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Set up an AWS account.
- An AWS Identity and Access Management (IAM) user with administrator permissions. For detailed instructions, see the section called "How AWS IoT SiteWise works with IAM".
- A running AWS IoT SiteWise demo.

> **Note**
>
> This tutorial requires the use of resources created in the Use the AWS IoT SiteWise demo. You must complete it before proceeding with this tutorial.

The demo typically takes around 3 minutes to create. If the demo fails to create, it might indicate insufficient permissions in your AWS account. In this case, switch to an account with administrative access. For more information about required permissions, see How AWS IoT SiteWise works with IAM.

> **Important**
>
> Keep all demo resources until you complete this tutorial. Deleting any components might disrupt the demo's functionality and affect your ability to display data in Grafana.

Step 1: Configure your Amazon Managed Grafana workspace

In this procedure, create and configure an Amazon Managed Grafana workspace to visualize your wind farm data.

1. Sign in to the Amazon Managed Grafana console.
2. Choose **Create workspace**.
3. Under **Workspace details**, enter a name for your workspace, such as **`SiteWiseTutorialDemo`**.

4. Under **Grafana version**, choose the latest version. Choose this version for the most up-to-date features and capabilities. For more information about different version sets, see Differences between Grafana versions in the *Amazon Managed Grafana User Guide*.
5. Choose **Next**.
6. Under **Authentication access**, choose **AWS IAM Identity Center**.
   - If AWS IAM Identity Center in your account isn't enabled, you will be prompted to set it up first. For detailed instruction about how to set up user access, see Identity-based policy examples for Amazon Managed Grafana in the *Amazon Managed Grafana User Guide*.
7. Under **Permission type**, choose **Service managed**. Amazon Managed Grafana automatically creates and configures the necessary IAM roles and permissions for any AWS data sources you choose to use in this workspace. For organizational member accounts, the Service managed option is only available if the account is designated as a delegated administrator. For information about setting up delegated administrator accounts, see Register a delegated administrator member account in the *AWS CloudFormation User Guide*.
8. Under **Workspace configuration options**, take the following actions:
   a. Select **Turn Grafana alerting on**. With this setting, you can create and manage alerts through a centralized interface in your workspace. For more information, see Working with Grafana alerting in the *Amazon Managed Grafana User Guide*.
   b. Select **Turn plugin management on**. This allows you to install, update, and uninstall plugins in your workspace. For more information, see Extend your workspace with plugins in the *Amazon Managed Grafana User Guide*.

> **Important**
>
> Be sure to enable plugin management. If you don't select this option, you cannot add AWS IoT SiteWise as a data source in the following step.

9. Under **Network access control**, choose **Open access**. You use demo data in this tutorial, so you can make the workspace publically available.
   - **Open access** – Allows your workspace to be publicly accessible.
   - **Restricted access** – Limits access to specific IP ranges or VPC endpoints. For more information, see How VPC connectivity works in the *Amazon Managed Grafana User Guide*.
10. Choose **Next**.

11. Under the **Service managed permission settings** page, choose **Current account** to have Amazon Managed Grafana automatically create policies and permissions for accessing AWS data within your account.

12. Under **Data sources**, select **AWS IoT SiteWise**. For more information, see [Connect to an AWS IoT SiteWise data source]() in the *Amazon Managed Grafana User Guide*.

13. (Optional) Under **Notification channels**, select **Amazon SNS** to enable Grafana alerts to be sent through Amazon SNS, This creates an IAM policy that allows publishing to Amazon SNS topics with names starting with Grafana. You need to complete the notification channel setup later in your Grafana console within the workspace.

14. Confirm the workspace details, and choose **Create workspace**. This process takes a couple of minutes.

15. On the **Authentication** tab, under **AWS IAM Identity Center**. assign users or groups to your workspace by doing the following:

    - To assign the user who will manage AWS IoT SiteWise data, choose **Assign new user or group**. Then choose **Make admin** from the **Actions** dropdown list to grant them administrative privileges.

      > **Important**
      >
      > To manage the Grafana workspace, you must assign the `Admin` role to at least one user. This user will have full access to the Grafana workspace console.

You have now set up and configured your Grafana workspace. In the next step, you can add AWS IoT SiteWise as a data source and begin creating visualizations for your wind farm data. From your workspace, you can query, visualize, and analyze your industrial data in real time. For more information about Amazon Managed Grafana workspaces, see [Use your Grafana workspace]() in the *Amazon Managed Grafana User Guide*.

Step 2: Add AWS IoT SiteWise as a data source

To help you visualize your data, Amazon Managed Grafana includes the AWS Data Sources plugin, which simplifies the process of connecting to AWS services. This plugin comes pre-installed in your workspace and provides a unified interface for discovering and configuring AWS resources as data sources. For your wind farm visualization, you'll use this plugin to connect to AWS IoT SiteWise.

For more information, see [Connect to an AWS IoT SiteWise data source] in the *Amazon Managed Grafana User Guide*.

Before you can start querying your wind farm data, the AWS Data Sources plugin needs the appropriate permissions to access your AWS IoT SiteWise resources. These permissions were automatically configured when you selected AWS IoT SiteWise as a data source in the previous step. For more information about plugin permissions, see [Required permissions] in the *Amazon Managed Grafana User Guide*.

To connect AWS IoT SiteWise to your Grafana workspace

1. Open the [Amazon Managed Grafana console]. In your workspace details page, choose the URL displayed under **Grafana workspace URL**. The workspace URL opens the Grafana workspace console login page.
2. Choose **Sign in with AWS IAM Identity Center** and enter your credentials. These credentials only work if you responded to the email from Amazon Managed Grafana that prompted you to create a password for IAM Identity Center.
3. In the left navigation pane, choose **Apps**, then **AWS Data Sources**, and then select the **AWS services** tab.
4. Under **AWS IoT SiteWise**, choose **Install now** to install the latest version of the AWS IoT SiteWise plugin.
5. Navigate to the **Data sources** tab, and select **IoT SiteWise** as the service.
6. Under **Default region**, select the Region where you want to retrieve data from, for example, **US East (N. Virginia)**.
7. After specifying the parameters for the plugin, select **Add data source**.
8. Select **Go to settings**.
9. Under **Connection details**, select **Save and test** to verify that the service is working.

Step 3: Create a dashboard to explore and visualize your data

In this step, create a Grafana dashboard to visualize the demo wind farm data that you created earlier. Dashboards help you monitor your data by displaying multiple visualizations in a single view. You can use dashboards to track metrics, analyze patterns, and gain insights from your industrial data. For more information, see [Create your first dashboard] in the *Amazon Managed Grafana User Guide*.

To create your first dashboard in Grafana

1. In the left navigation pane, select **Dashboards** and then choose **Create Dashboard** to start building your first dashboard.
2. Select **Add visualization**. This opens the panel editor where you can configure data sources, queries, and visualization settings.
3. Under the **Query** tab, select the AWS IoT SiteWise data source from the dropdown menu.
4. Under **Query type**, select **Get property value aggregates** from the dropdown menu to retrieve aggregated values for asset properties over time.
5. Select **Explore** to view available assets in your hierarchy. From the **Hierarchy** tab, select **Demo Wind Farm Asset**, and then select **Demo Turbine Asset 1**.
6. Under **Property**, select **Average Power** from the available properties. Select **Run queries** to run the query so you can preview the output. The visualization will update to show the average power data for `Demo Turbine Asset 1`.
7. In the right navigation pane, give the new panel a title, such as **`Turbine Demo 1 (Average Power)`**. Choose **Apply** to save your changes.

   > **Warning**
   >
   > Whenever you make any changes to the dashboard, save the dashboard before refreshing the page or navigating away. Otherwise, you will lose your progress.

8. In the top-right corner, select **Save dashboard**. You will be prompted to enter a name for your dashboard, for example, **`SiteWise Wind Farm Demo Dashboard`**.
9. Choose **Save**.

For information about sharing dashboards, see [Sharing dashboards and panels](#) in the *Amazon Managed Grafana User Guide*.

To add another panel to visualize the wind speed

1. Select **Add visualization** to open a blank panel.
2. Under the **Query** tab, select the AWS IoT SiteWise data source from the dropdown menu.
3. Under **Query type**, select **Get property value** from the dropdown menu and under **Asset**, select `Demo Wind Farm Asset`, then `Demo Turbine Asset 1`.

4. Under **Property**, select **Wind Speed** from the available properties. Select **Run queries** to update the changes.
5. Under **Visualization**, select **Gauge**. Gauges work best for displaying single, real-time metrics like wind speed.
6. In the right navigation pane, give the new panel a title, such as **`Turbine Demo 1 (Wind Speed)`**.
7. Under **Standard options** from the Panel options, select **Unit**. Choose **Velocity**, and then choose **meters/second (m/s)**.
8. Choose **Apply** to save your changes.

   The following image displays what your Grafana dashboards might look like when you complete this step.

(optional) Step 4: Set up alerts to monitor performance

Alerts indicate state changes after they occur to identify performance issues with your industrial equipment. For more information, see [Amazon Managed Grafana alerting]() in the *Amazon Managed Grafana User Guide*.

To set up alerts in Grafana

1. Under the **Rule** tab in the `Turbine Demo 1 (Average Power)`, set **Evaluate every** to `5m` and **For** to `15m`. This configuration evaluates the average power every 5 minutes and triggers an alert if the condition persists for longer than 15 minutes.

2. Under **Conditions**, select **IS BELOW** and enter **`7,020`** watts. This setting will notify you if average turbine conditions fall below 7,020 watts for longer than 5 minutes. For more information about creating alerts, see Alert rule fields in the *Amazon Managed Grafana User Guide*.

You have completed the tutorial. In this procedure, you created a Grafana workspace and configured it to visualize wind farm data from AWS IoT SiteWise. You built an interactive dashboard with multiple widget types, including a time-series graph for average power and a gauge for wind speed. You also set up alerts to monitor turbine performance, enabling you to identify potential issues before they disrupt production. You can continue to enhance your dashboard by adding more visualizations, creating additional alerts, or connecting other AWS data sources to gain deeper insights into your industrial operations.

Step 5: Clean up resources after the tutorial

After you complete this tutorial about visualizing data with Grafana, clean up your resources to avoid incurring additional charges.

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 are 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.

(optional) 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.

For more information, see Delete the AWS IoT SiteWise demo.

If you delete an Amazon Managed Grafana, all the configuration data for that workspace is also deleted. This includes dashboards, data source configuration, alerts, and snapshots.

To delete an Amazon Managed Grafana workspace

1. Open the Amazon Managed Grafana console.
2. In the left navigation pane, choose the menu icon.
3. Choose **All workspaces**.
4. Choose the name of the workspace that you want to delete.
5. Choose **Delete**.
6. To confirm the deletion, enter the name of the workspace and choose **Delete**.

> **Note**
>
> This procedure deletes a workspace. Other resources may not be deleted. For example, IAM roles that were in use by the workspace are not deleted (but may be unlocked if they are no longer in use).

For more information, see Delete an Amazon Managed Grafana workspace in the *Amazon Managed Grafana User Guide*.

Additional resources

For more information about visualizing data, see the following resources:

- Troubleshooting Amazon Managed Grafana identity and access in the *Amazon Managed Grafana User Guide*
- Security best practices in the *Amazon Managed Grafana User Guide*
- Integrate AWS IoT SiteWise with Grafana
- Process and visualize data with SiteWise Edge and open-source tools
- Users, teams, and permissions in the *Amazon Managed Grafana User Guide*
- Amazon Managed Grafana permissions and policies for AWS data sources in the *Amazon Managed Grafana User Guide*

Visualize and share wind farm data in SiteWise Monitor

This tutorial explains how to use AWS IoT SiteWise Monitor to visualize and share industrial data through managed web applications, known as portals. Each *portal* encompasses projects, providing you with the flexibility to choose which data is accessible within each project. Then, specify people in your organization that can access each portal. Your users sign in to portals using AWS IAM Identity Center accounts, so you can use your existing identity store or a store managed by AWS.

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 to each project for every user in your company. For more information, see Monitor data with AWS IoT SiteWise Monitor.

Throughout the tutorial, you enhance the AWS IoT SiteWise demo, providing a sample dataset for a wind farm. You configure a portal in SiteWise Monitor, create a project, and dashboards to visualize the wind farm data. The tutorial also covers the creation of additional users, along with the assignment of permissions to own or view the project and its associated dashboards.

> **Note**
>
> When you use SiteWise Monitor, you're charged per user that signs in to a portal (per month). In this tutorial, you create three users, but you only need to sign in with one user. After you complete this tutorial, you incur charges for one user. For more information, see AWS IoT SiteWise Pricing.

Topics

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Set up an AWS account.
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see What is the AWS Management Console?.
- An AWS Identity and Access Management (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 Use the AWS IoT SiteWise demo.
- If you enabled IAM Identity Center 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 IAM Identity Center, 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 IAM Identity Center user in your organization. In this case, you can create the portal and dashboards, but you can't create new IAM Identity Center users to assign to projects.

Step 1: Create a portal in SiteWise Monitor

In this procedure, you create a portal in AWS IoT SiteWise Monitor. Each *portal* is a managed web application that you and your users can sign in to with AWS IAM Identity Center accounts. With IAM Identity Center, you can 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.

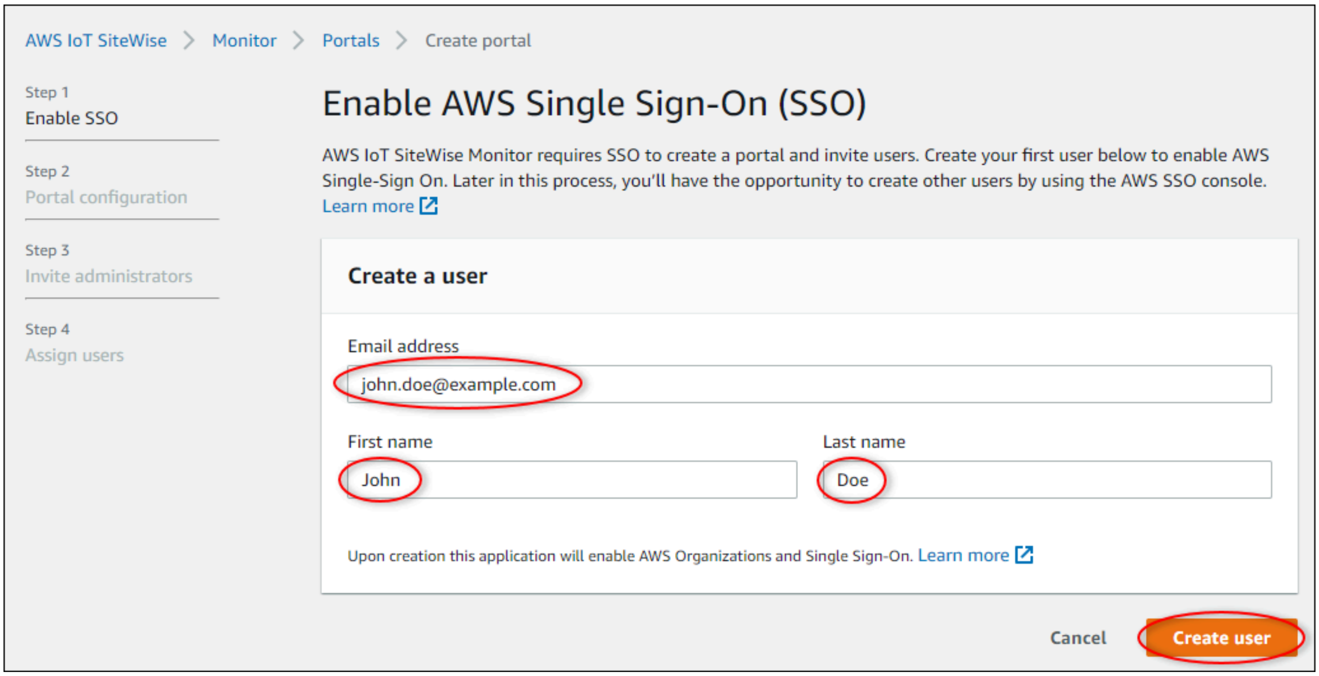
To create a portal

1. Sign in to the AWS IoT SiteWise console.
2. Review the AWS IoT SiteWise endpoints and quotas 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 IAM Identity Center, skip to step 6. Otherwise, complete the following steps to enable IAM Identity Center:

    a. On the **Enable AWS IAM Identity Center (SSO)** page, enter your **Email address**, **First name**, and **Last name** to create an IAM Identity Center 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 IAM Identity Center 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.

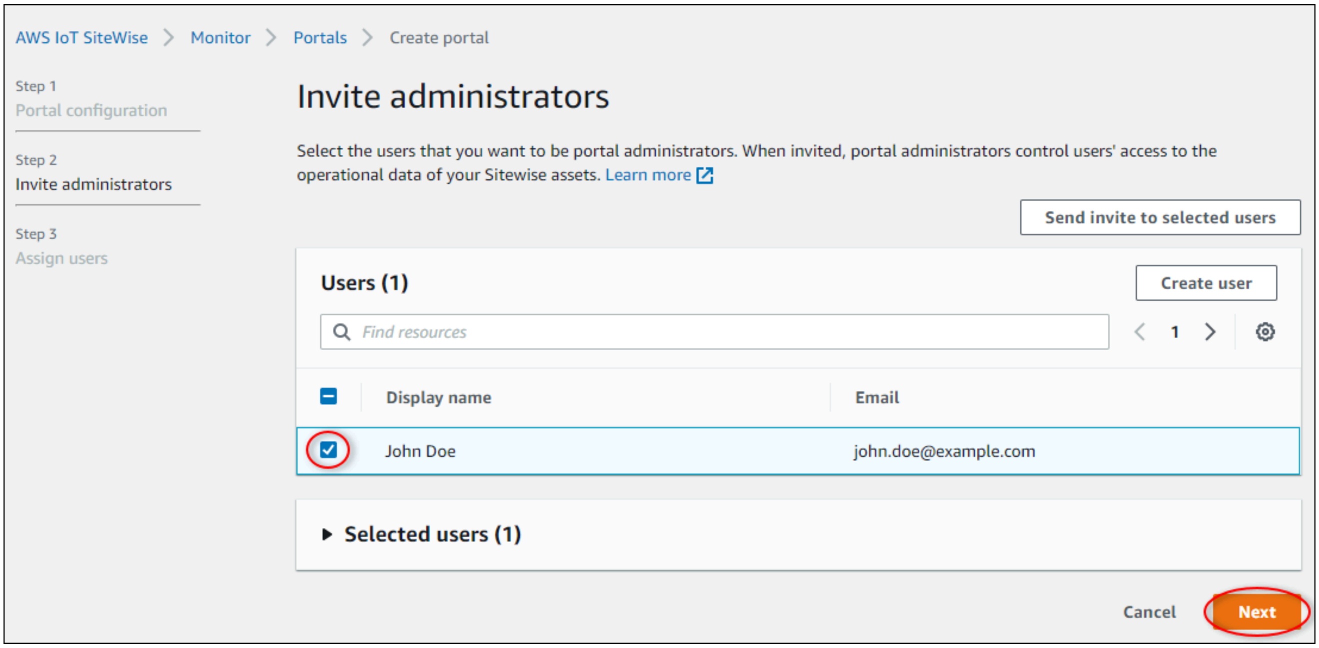
    c. (Optional) Upload an image to display in the portal.

    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 IAM Identity Center 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 IAM Identity Center 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 IAM Identity Center 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 IAM Identity Center users to add to the portal.

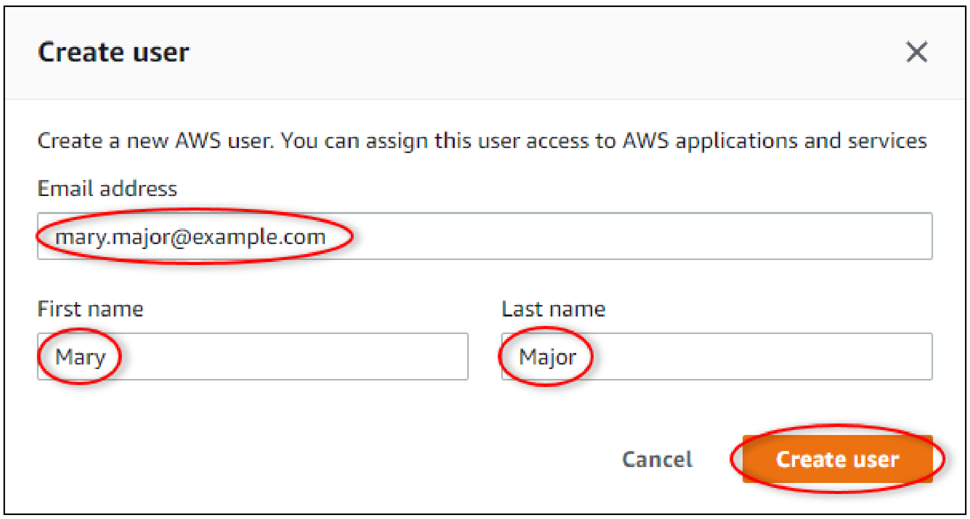
> **Note**
>
> If you aren't signed in to your AWS Organizations management account, you can't create IAM Identity Center users. Choose **Assign users** to create the portal without portal users, and then skip this step.

On this page, do the following:

a. Complete the following steps twice to create two IAM Identity Center users:

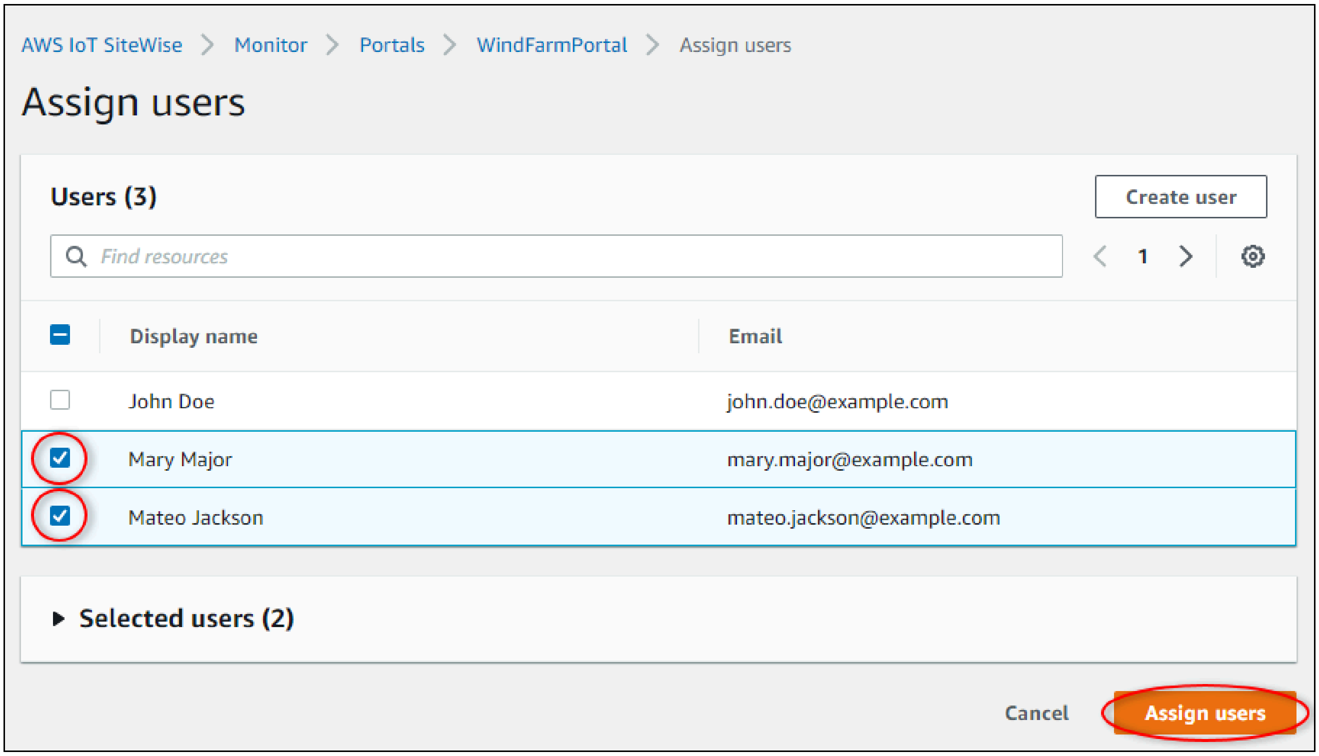
   i. Choose **Create user** to open a dialog box where you enter details for the new user.

   ii. Enter an **Email address**, **First name**, and **Last name** for the new user. IAM Identity Center 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 IAM Identity Center users that you created in the previous step.



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

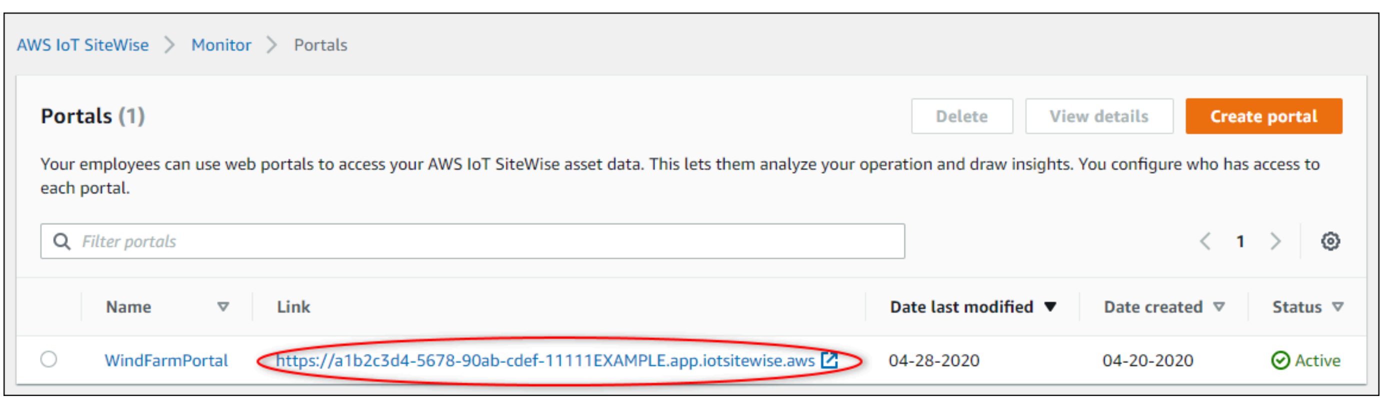
The portals page opens with your new portal listed.

Step 2: Sign in to a portal

In this procedure, you sign in to your new portal using the AWS IAM Identity Center 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 IAM Identity Center 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 IAM Identity Center**.

    b. Open that invitation email and choose **Accept invitation**.

    c. In the new window, set a password for your IAM Identity Center user.

    If you want to sign in later to the portal as the second and third IAM Identity Center 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 IAM Identity Center console. For more information, see Reset the IAM Identiy Center user password for an end user in the *AWS IAM Identity Center User Guide*.

3. Enter your IAM Identity Center **Username** and **Password**. If you created your IAM Identity Center 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 IAM Identity Center 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.

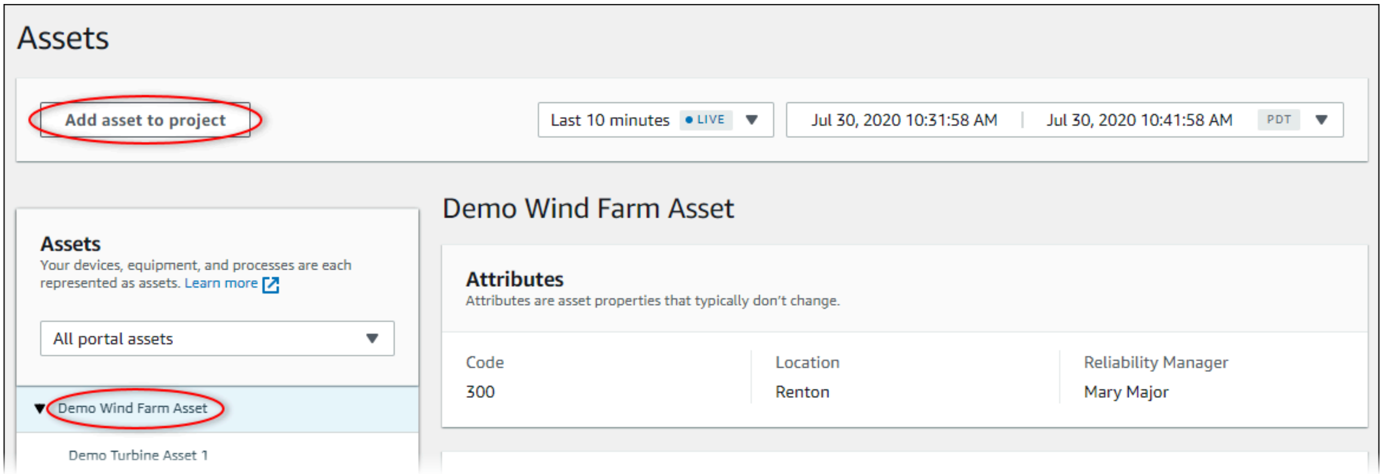
Step 3: Create a wind farm project

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.

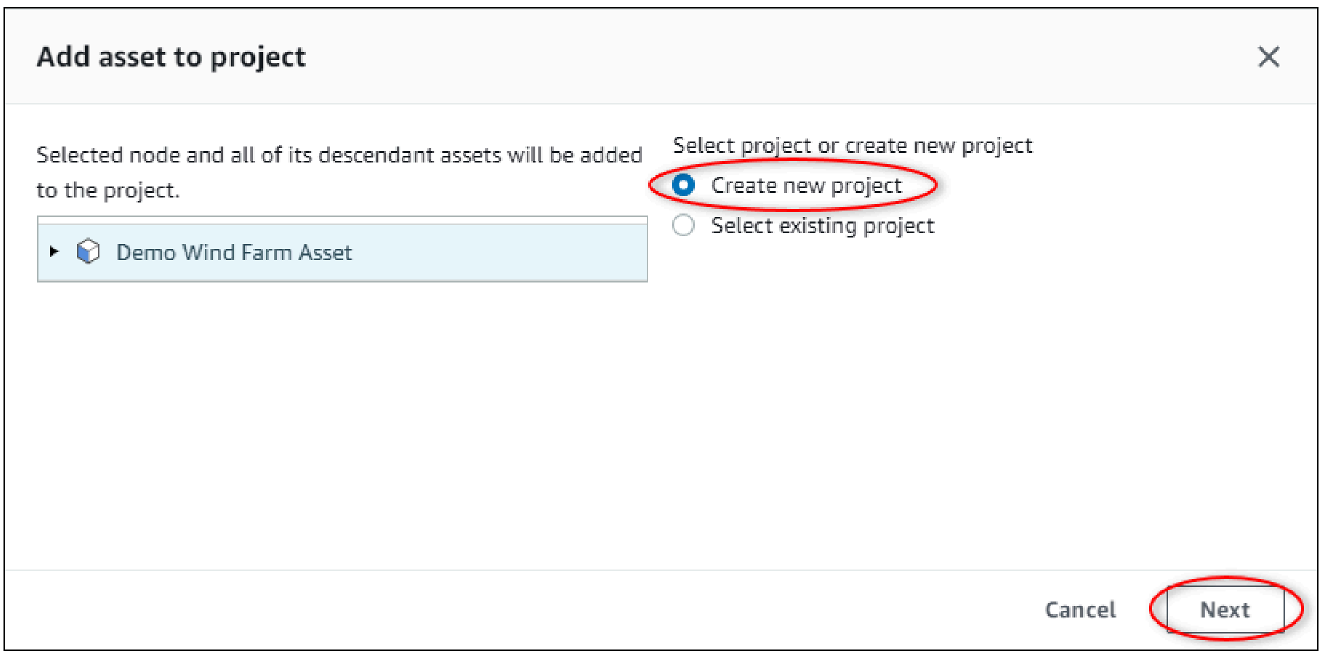
To create a wind farm project

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.

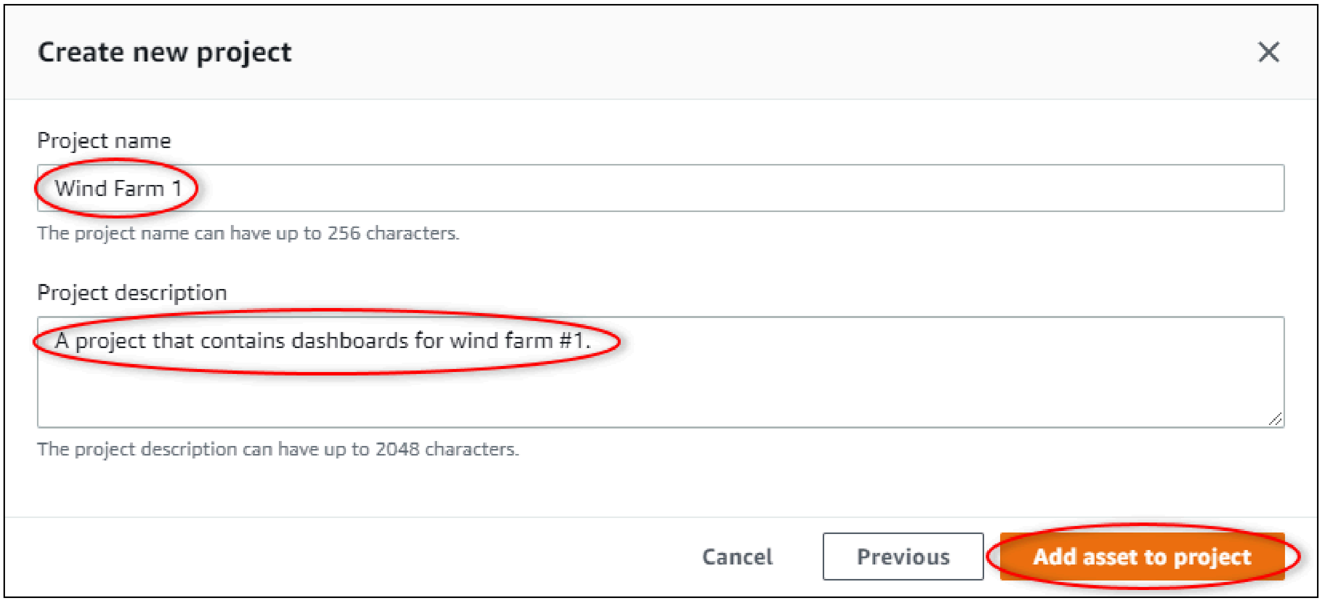
3. Choose **Add asset to project** in the upper left. Projects contain dashboards that your portal users can view to explore your data. Each project has access to a subset of your assets in AWS IoT SiteWise. When you add an asset to a project, all users with access to that project can also access data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



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**.



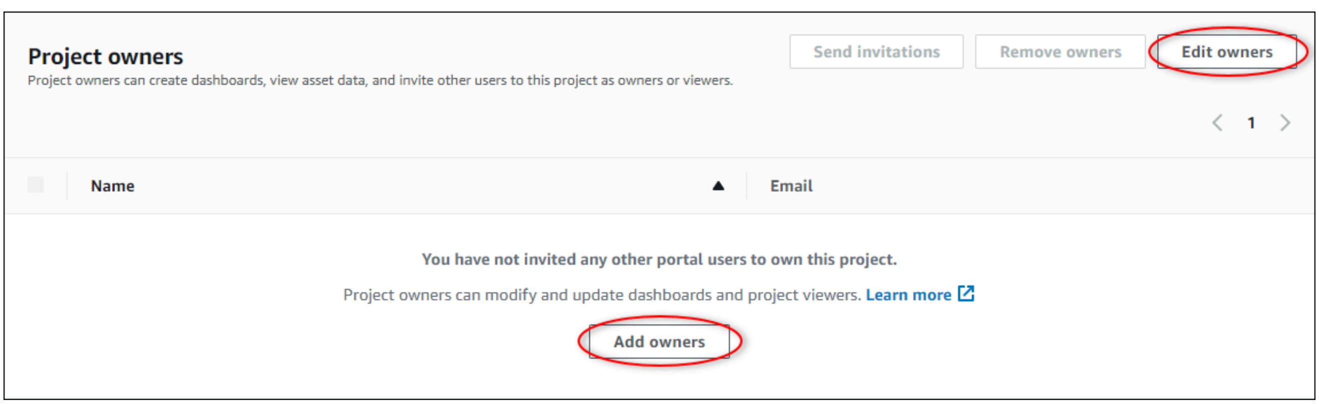
Your new project's page opens.

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

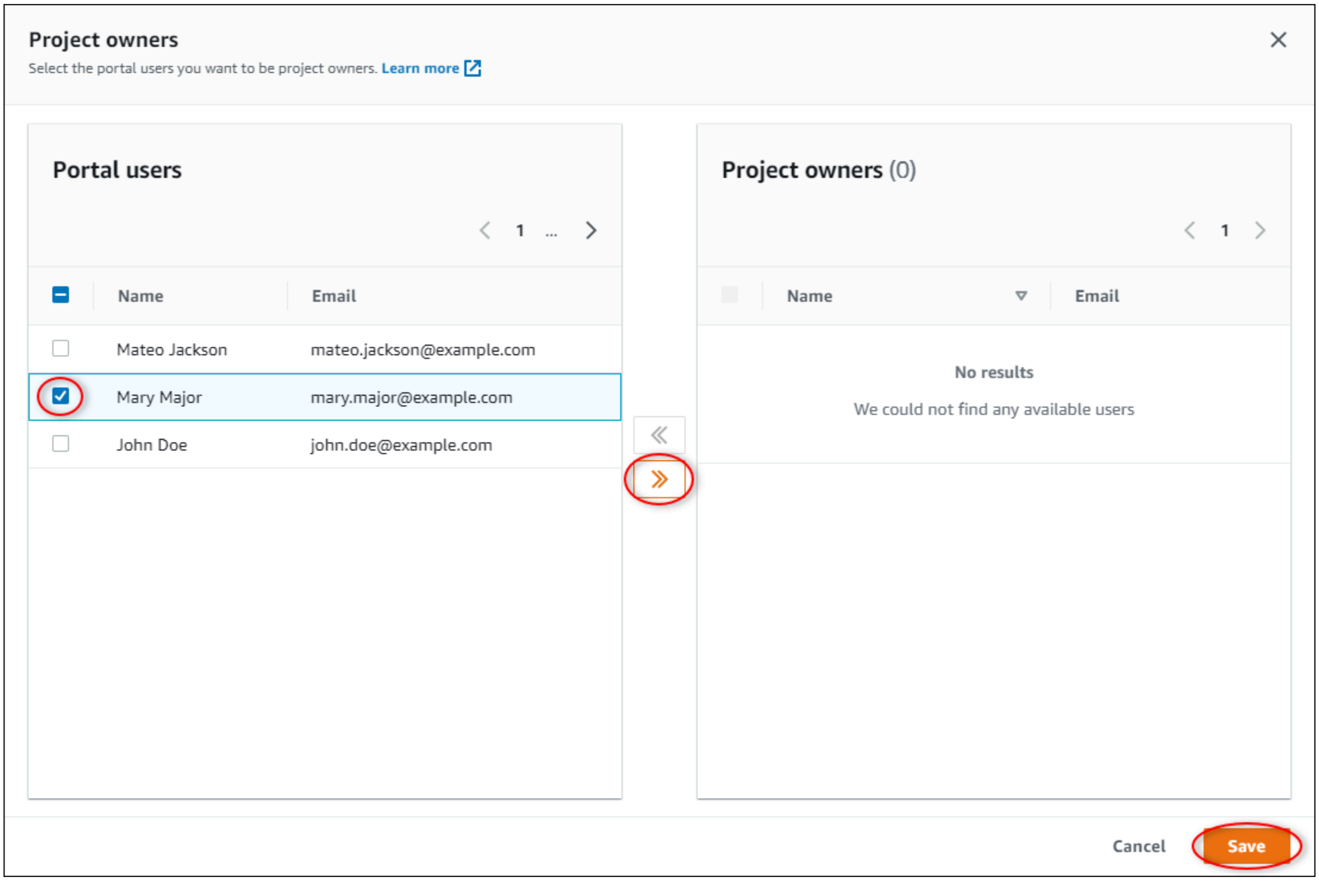
> **Note**
>
> If you aren't signed in to your AWS Organizations management account, you might not have portal users to assign to this project, so you can skip this step.

On this page, do the following:

a. Under **Project owners**, choose **Add owners** or **Edit users**.



b. Choose the user to add as a project owner (for example, **Mary Major**), and then choose the **>>** icon.



c. Choose **Save**.

Your IAM Identity Center user **Mary Major** can sign in to this portal to edit the dashboards in this project and share this project with other users in this portal.

d. Under **Project viewers**, choose **Add viewers** or **Edit users**.

e. Choose the user to add as a project viewer (for example, **Mateo Jackson**), and then choose the **>>** icon.

f. Choose **Save**.

Your IAM Identity Center user **Mateo Jackson** can sign in to this portal to view, but not edit, the dashboards in the wind farm project.

Step 4: Create a dashboard to visualize wind farm data

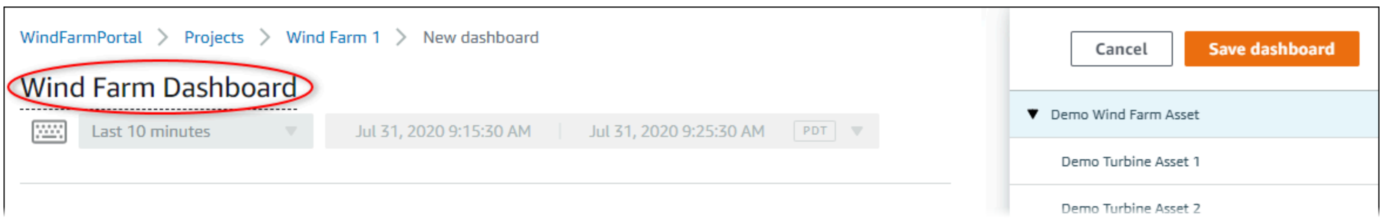
In this procedure, you create dashboards to visualize the demo wind farm data. Dashboards contain customizable visualizations of your project's asset data. Each visualization can have a different type, such as a line chart, bar chart, or key performance indicator (KPI) display. You can choose the visualization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

To create a dashboard with visualizations

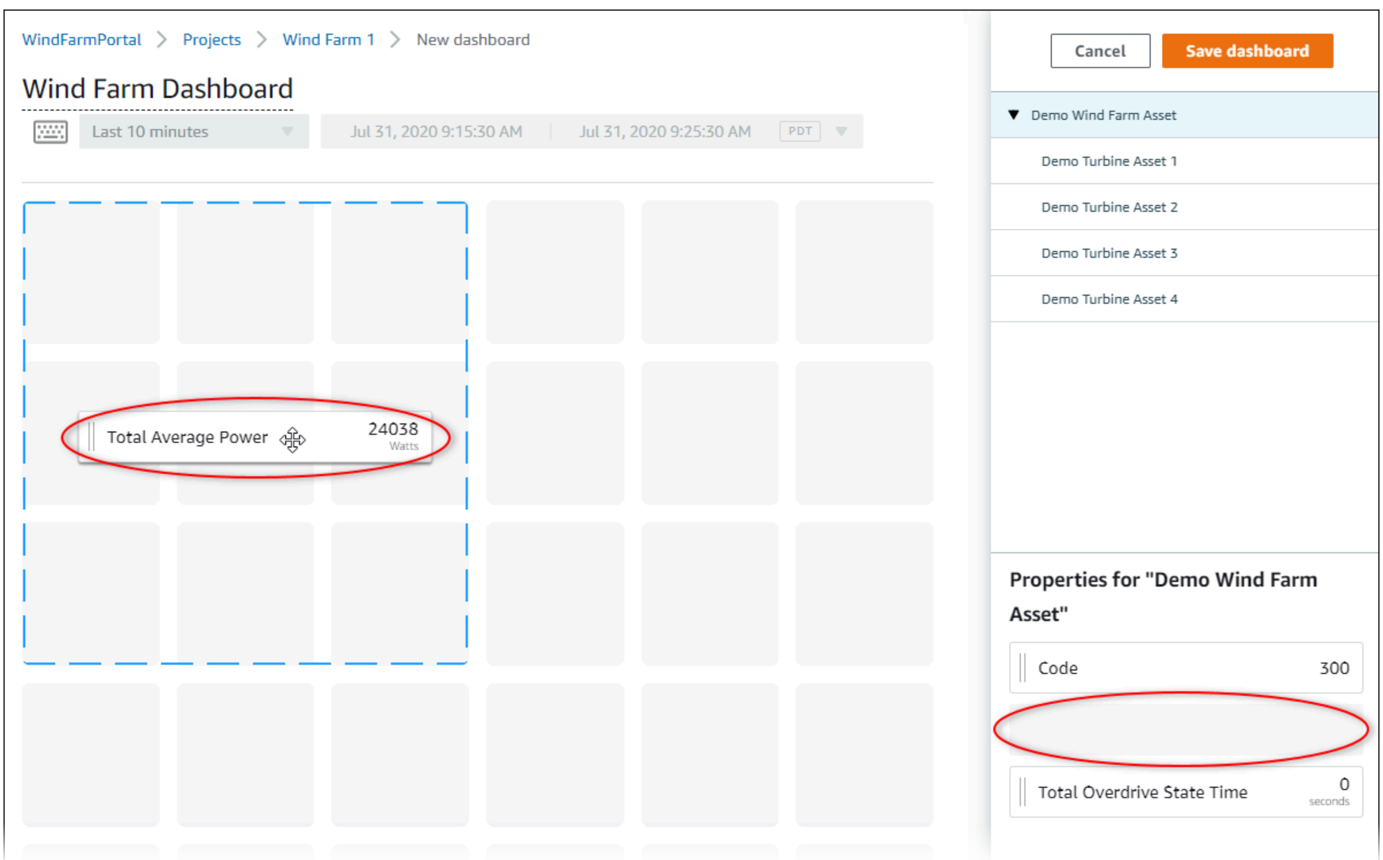
1. On your new project's page, choose **Create dashboard** to create a dashboard and open its edit page.

   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.

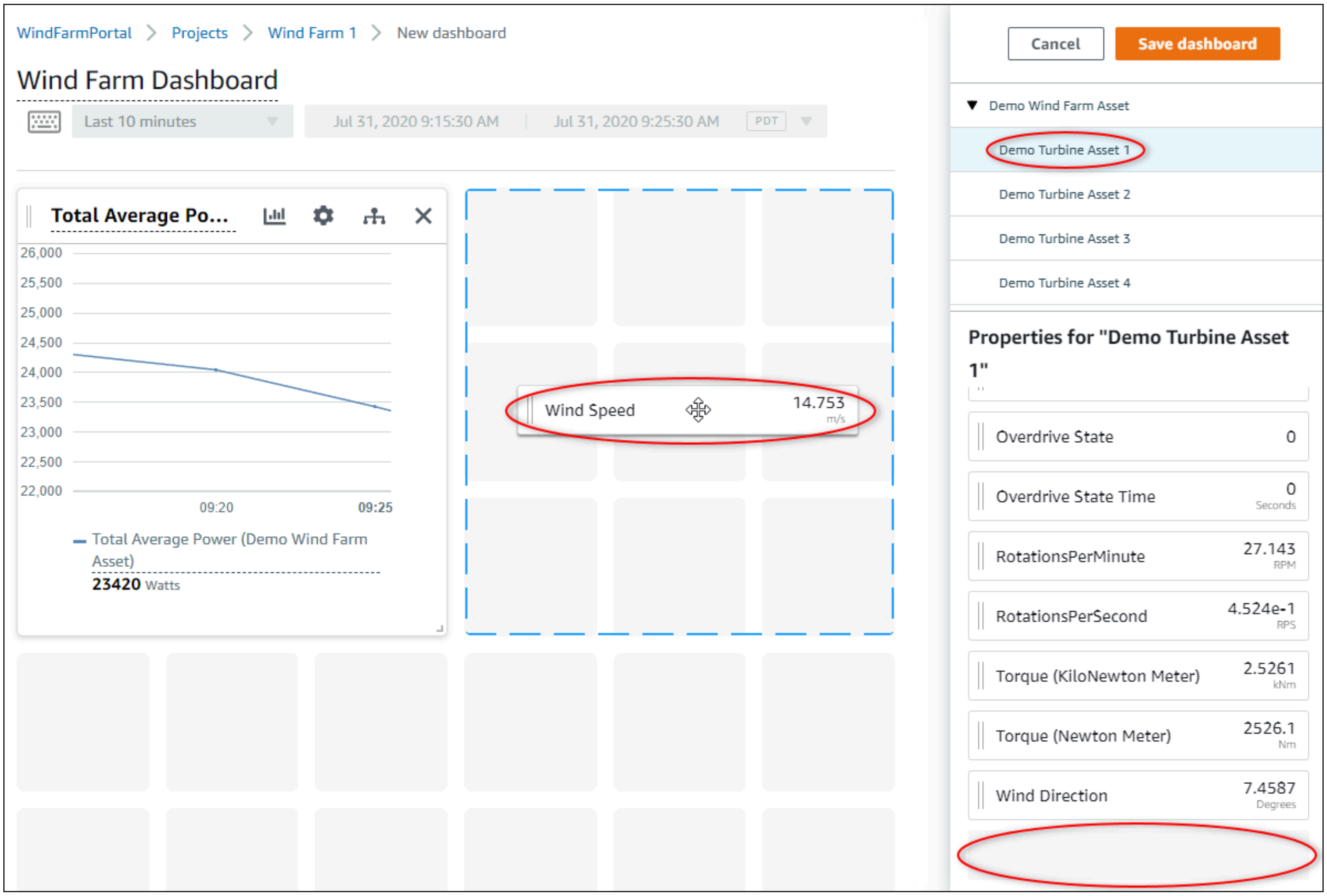
2. Enter a name your dashboard.



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

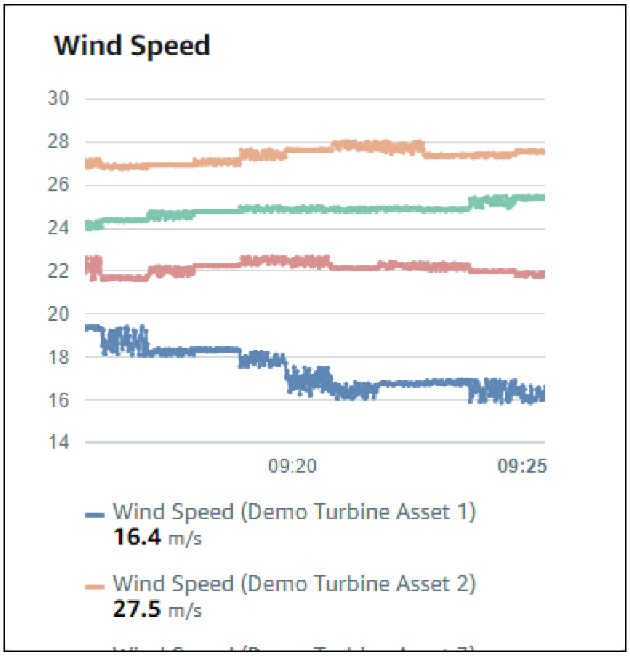


4. Choose **Demo Turbine Asset 1** to show properties for that asset, and then drag **Wind Speed** to the dashboard to create a visualization for wind speed.

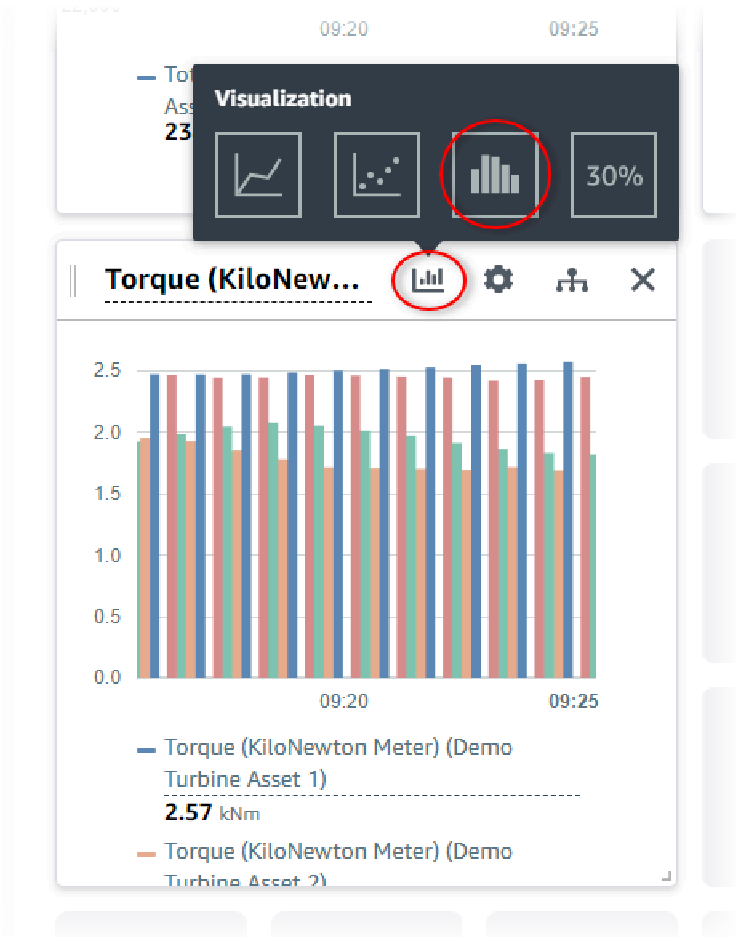


5. Add **Wind Speed** to the new wind speed visualization for each **Demo Turbine Asset 2**, **3**, and **4** (in that order).

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

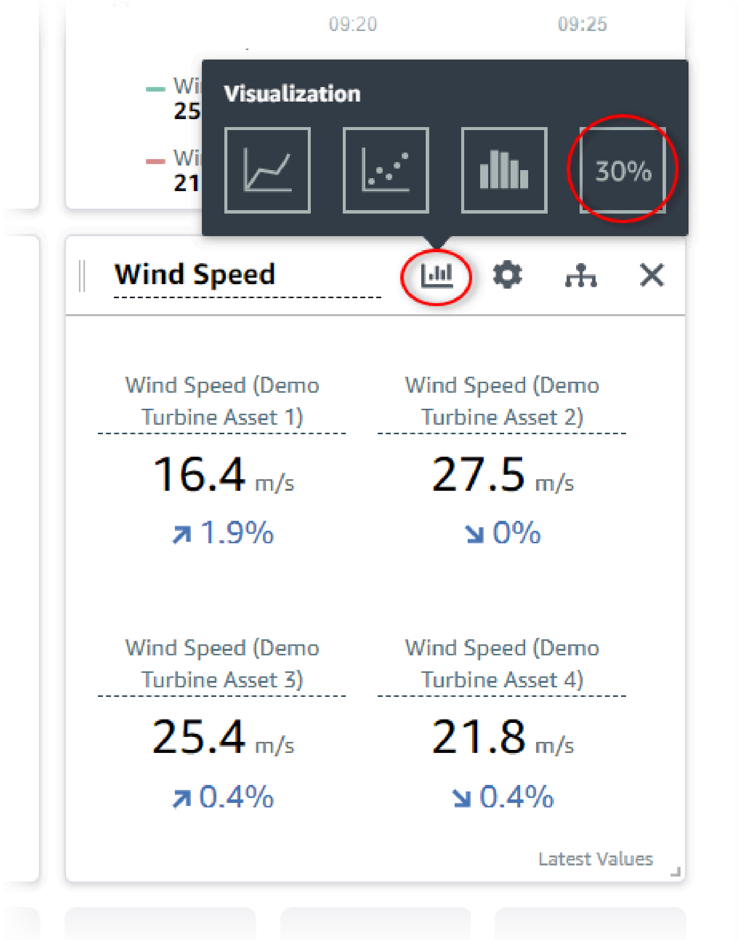


6. Repeat steps 4 and 5 for the wind turbines' **Torque (KiloNewton Meter)** properties to create a visualization for wind turbine torque.
7. Choose the visualization type icon for the **Torque (KiloNewton Meter)** visualization, and then choose the bar chart icon.



8. Repeat steps 4 and 5 for the wind turbines' **Wind Direction** properties to create a visualization for wind direction.

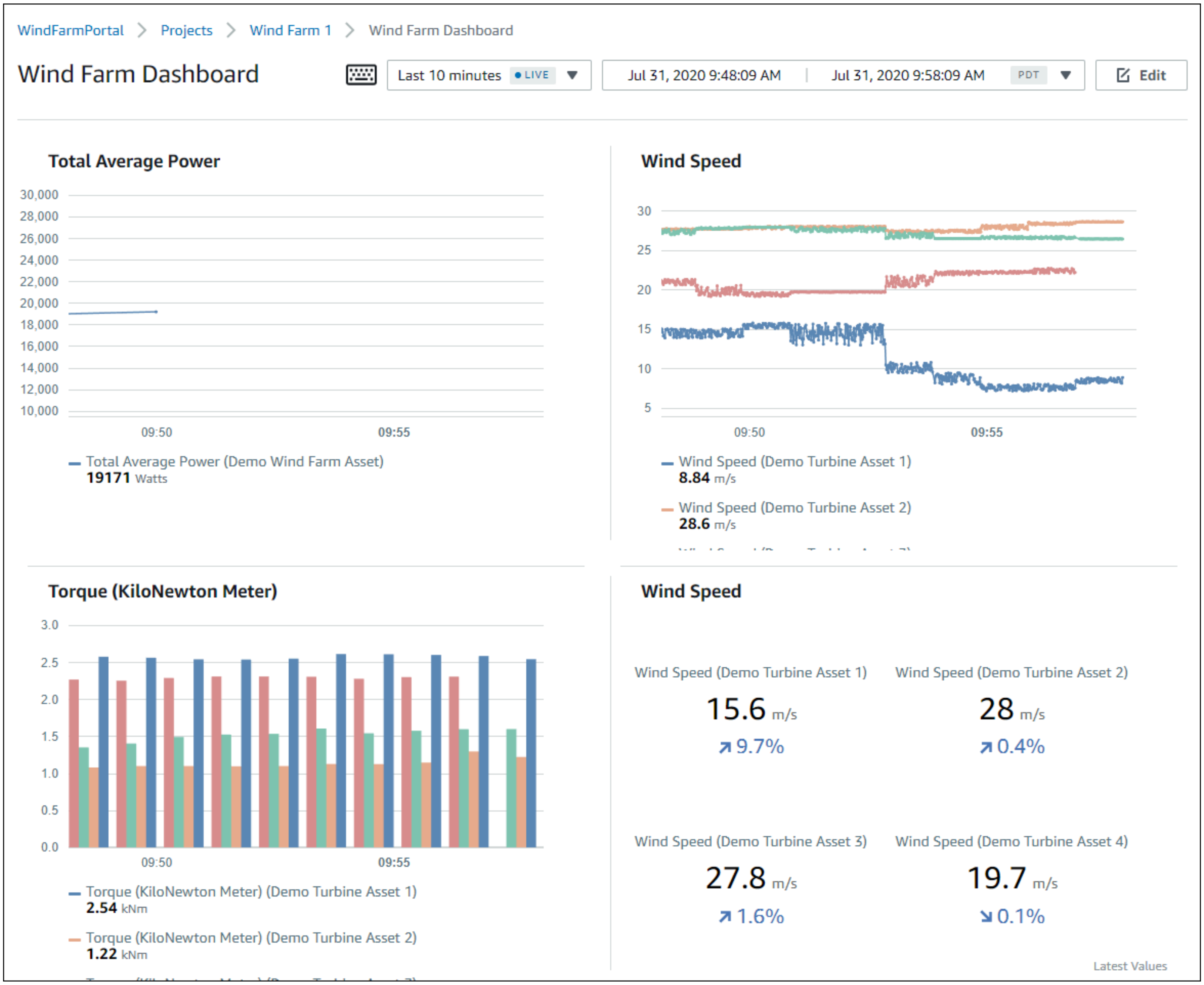
9. Choose the visualization type icon for the **Wind Direction** visualization, and then choose the KPI chart icon (**30%**).



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.

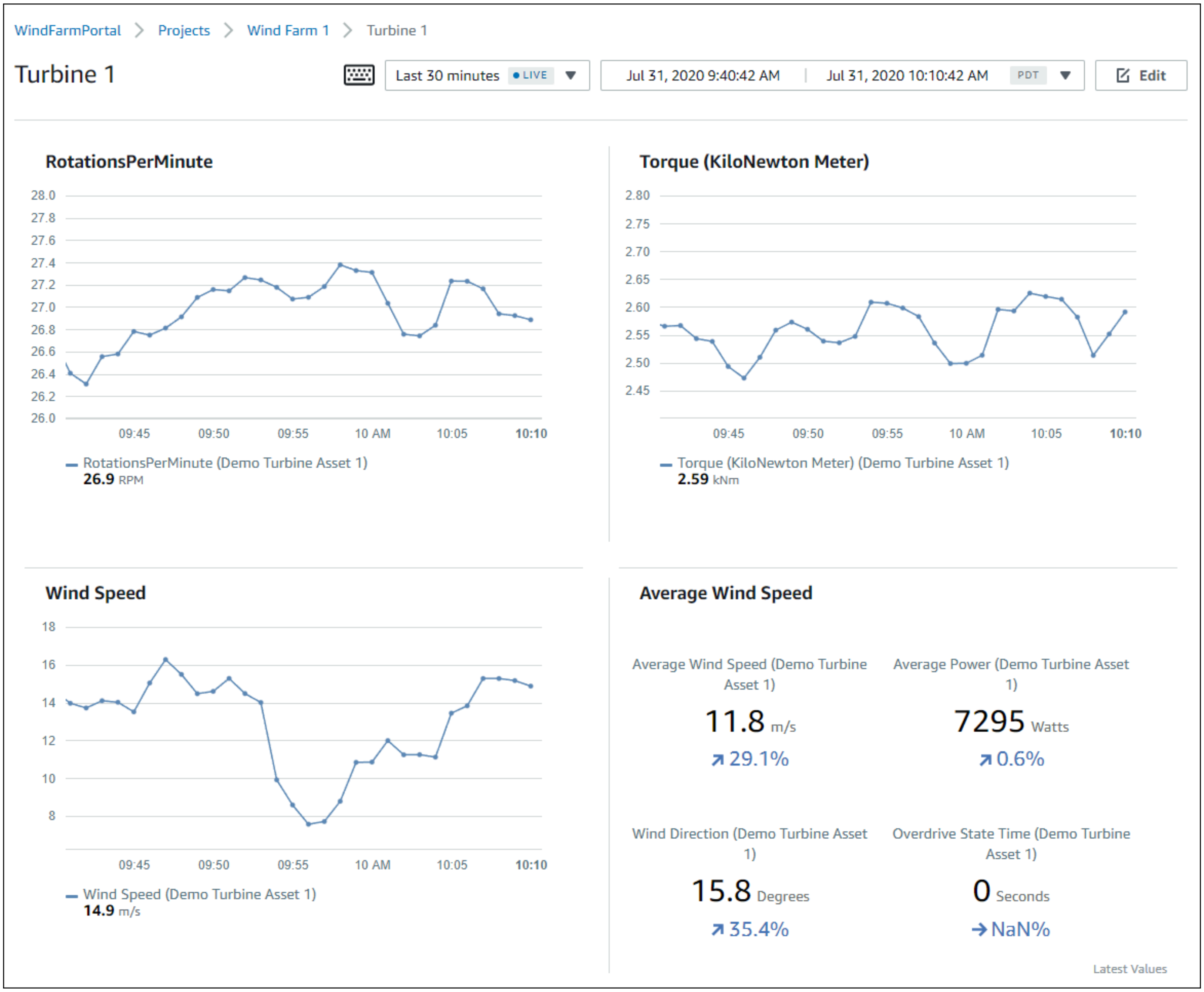
    Your dashboard should look similar to the following screenshot.



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*.

Step 5: Explore the portal

In this procedure, you can explore the portal as a user with fewer permissions than an AWS IoT SiteWise portal administrator.

To explore the portal and finish the tutorial

- (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 IAM Identity Center application portal to sign out of your IAM Identity Center user.
  c. Sign in to the portal as the IAM Identity Center user that you assigned as a project owner or project viewer. For more information, see Step 2: Sign in to a portal.

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.

Step 6: Clean up resources after the tutorial

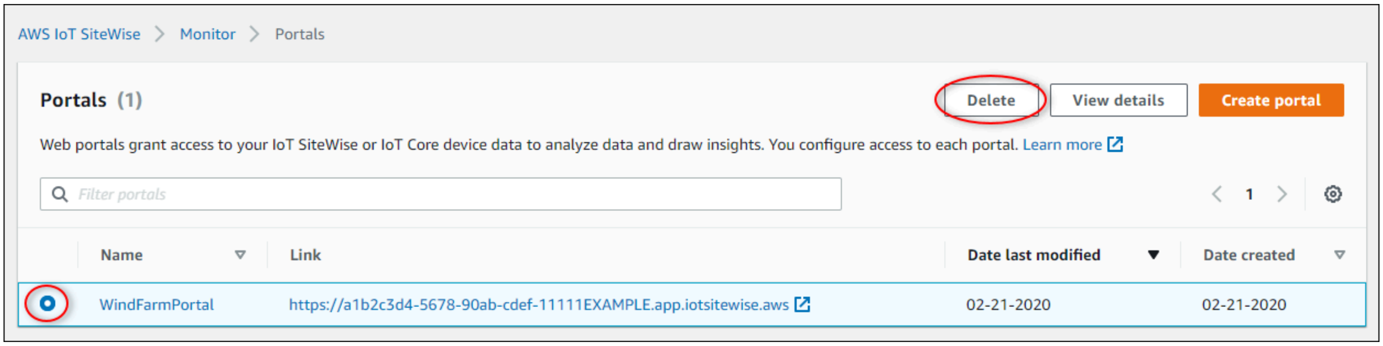
After you complete the tutorial, you can clean up your resources. You aren't charged for AWS IoT SiteWise if users don't sign in to your portal, but you can delete your portal and AWS IAM Identity Center directory users. Your demo wind farm assets are deleted at the end of the duration that you chose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

Use the following procedures to delete your portal and IAM Identity Center users.

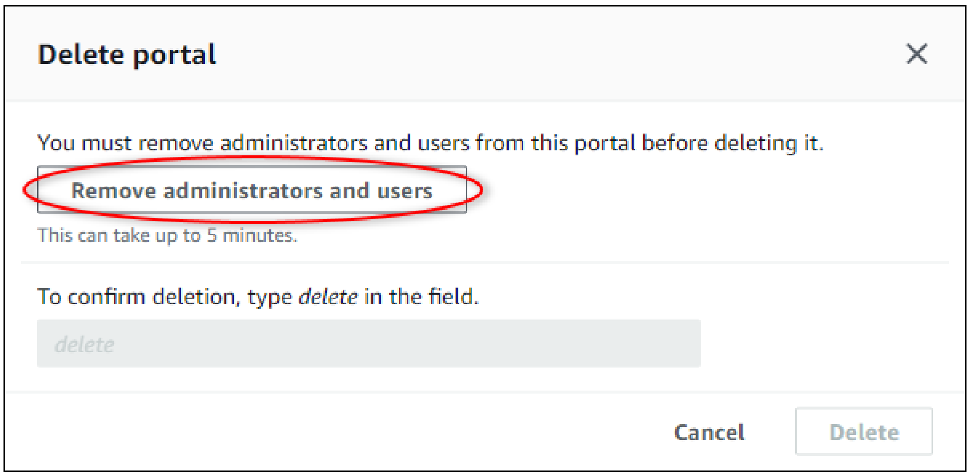
To delete a portal

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**.

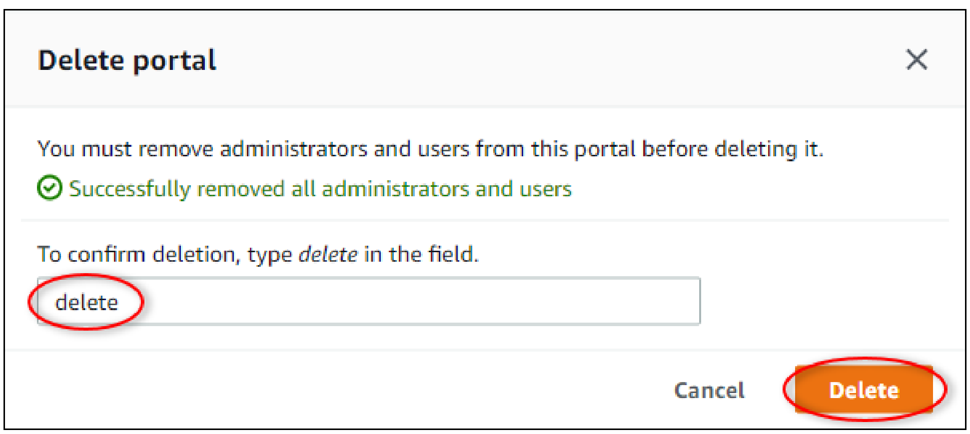
When you delete a portal or project, the assets associated to deleted projects aren't affected.



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



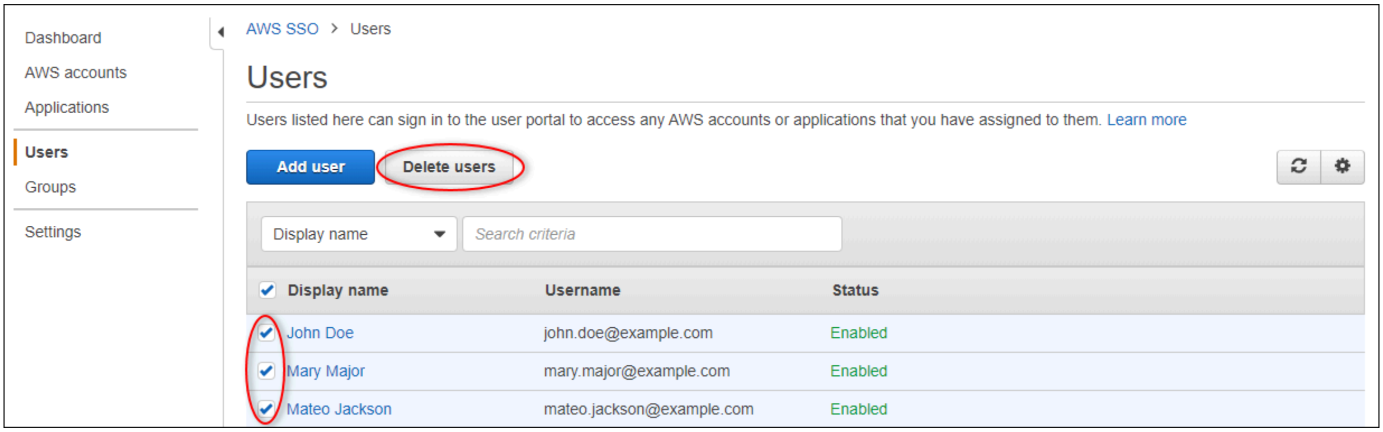
5. Enter **`delete`** to confirm deletion, and then choose **Delete**.



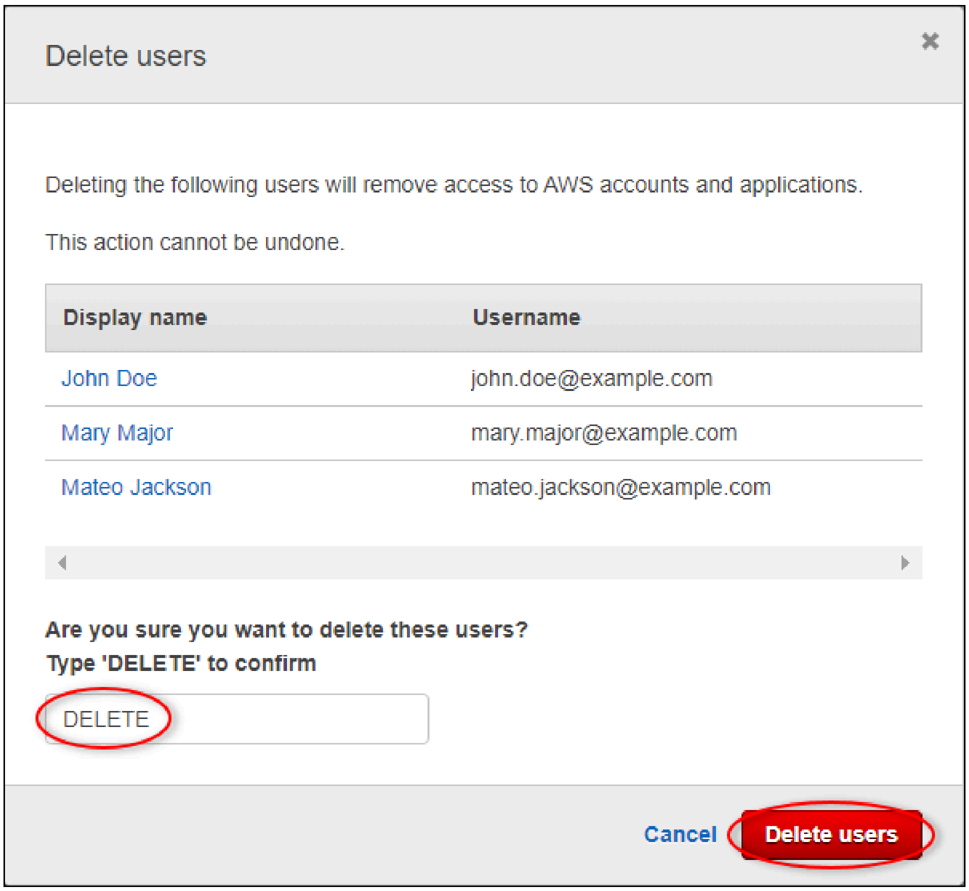
To delete IAM Identity Center users

1. Navigate to the [IAM Identity Center 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**.



Publish property value updates to Amazon DynamoDB

This tutorial introduces a convenient way to store your data by using [Amazon DynamoDB](), making it easier to access historical asset data without repeatedly querying the AWS IoT SiteWise API. After you complete this tutorial, you can 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 want to monitor and visualize your data without implementing a custom software solution, see Monitor data with AWS IoT SiteWise Monitor.

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, define AWS IoT Core rules that perform actions, such as the DynamoDB action, depending on the contents of those messages. For more information, see Interact with other AWS services.

Topics

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Set up an AWS account.
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see What is the AWS Management Console?
- An IAM user with administrator permissions. For detailed instructions, see the section called "How AWS IoT SiteWise works with IAM".
- 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 Use the AWS IoT SiteWise demo.

Step 1: Configure 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 IoT SiteWise endpoints and quotas 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. Choose **Measurements**.
7. Update the **Wind Speed** property's **MQTT Notification status** to **ACTIVE**.
8. Choose **Save** at the bottom of the page.
9. Repeat steps 5 through 7 for each demo turbine asset.
10. Choose a demo turbine (for example, **Demo Turbine Asset 1**).
11. Choose **Measurements**.
12. 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
    ```

Step 2: Create a rule in AWS IoT Core

In this step, create a rule in AWS IoT Core that parses the property value notification messages and inserts data into an Amazon DynamoDB table. AWS IoT Core rules parse MQTT messages and perform actions based on the contents and topic of each message. Then, you 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](https://console.aws.amazon.com/iot/).
2. In the left navigation pane, choose **Message routing**, and then choose **Rules**.
3. Choose **Create rule**.
4. Under **Specify rule properties**, 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 +. This selects 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 like the following example.

```
$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
```

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 **Rule actions**, navigate to **Action 1**.
8. On the **Select an action** page, choose **DynamoDBv2**. This splits the message into multiple columns of a DynamoDB table

9. Under **Table name**, choose **Create new table**. You create an Amazon DynamoDB table to receive wind speed data from the rule action.
10. Under **Table name** in the [DynamoDB console](https://console.aws.amazon.com/dynamodb/) enter a name for your table.
11. For **Partition 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**.
12. Choose **Create table**.
13. Return to the tab with the **Configure action** page.
14. On the **Attach rule action** page, refresh the **Table name** list, and choose your new DynamoDB table you created in the previous step.

Step 3: Configure the DynamoDB rule action

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

To configure the DynamoDB rule action

1. Choose **Create role** to create an IAM role that grants AWS IoT Core access to perform the rule action.
2. Enter a role name, for example, `WindSpeedDataRole`. Choose **Create role**.
3. Choose **Next**.
4. Choose **Create** at the bottom of the page to finish creating the rule.

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

Step 4: Explore data in DynamoDB

In this step, explore the demo assets' wind speed data in your new Amazon 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 **Explore table 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 Troubleshoot a rule (DynamoDB).
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 Query asset property notifications in AWS IoT SiteWise.

Now that you have completed the tutorial, disable or delete the rule and delete your DynamoDB table to avoid incurring additional charges. To clean up your resources, see Step 5: Clean up resources after the tutorial.

Step 5: Clean 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. You can also delete the demo manually.

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 are 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.

(optional) 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.

For more information, see Delete the AWS IoT SiteWise demo.

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 **INACTIVE**.
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 **Message routing** and then choose **Rules**.
3. Select your rule and choose **Delete**.
4. In the confirmation dialog, enter the name of the rule and then choose 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, for example, `WindSpeedData`.
4. Choose **Delete**.
5. In the confirmation dialog, enter **`confirm`** to delete the table.

Ingest data to AWS IoT SiteWise

AWS IoT SiteWise is designed to efficiently collect and correlate industrial data with corresponding assets, representing various aspects of industrial operations. This documentation focuses on the practical aspects of ingesting data into AWS IoT SiteWise, offering multiple methods tailored to diverse industrial use cases. For instructions to build your virtual industrial operation, see Model industrial assets.

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

- **AWS IoT SiteWise Edge**–Use SiteWise Edge gateway as an intermediary between AWS IoT SiteWise and your data servers. AWS IoT SiteWise provides AWS IoT Greengrass components that you can deploy on any platform that can run AWS IoT Greengrass to set up a SiteWise Edge gateway. This option supports linking with OPC UA server protocol.
- **AWS IoT SiteWise API**–Use the AWS IoT SiteWise API to upload data from any other source. Use our streaming BatchPutAssetPropertyValue API for ingestion within seconds, or the batch-oriented CreateBulkImportJob API to facilitate cost-effective ingestion in larger batches.
- **AWS IoT Core rules**–Use AWS IoT Core rules to upload data from MQTT messages published by an AWS IoT thing or another AWS service.
- **AWS IoT Events actions**–Use AWS IoT Events actions triggered by specific events in AWS IoT Events. This method is suitable for scenarios where data upload is tied to event occurrences.
- **AWS IoT Greengrass stream manager**–Use AWS IoT Greengrass stream manager to upload data from local data sources using an edge device. This option caters to situations where data originates from on-premises or edge locations.

These methods offer a range of solutions for managing data from different sources. Delve into the details of each option to gain a comprehensive understanding of the data ingestion capabilities AWS IoT SiteWise provides.

Manage data streams for AWS IoT SiteWise

A data stream is the resource that contains historical time series data. Each data stream is identified by a unique alias, making it easier to keep track of the origin for each piece of data. Data streams are automatically created in AWS IoT SiteWise when the first time series data is received. If the first time series data is identified with an alias, AWS IoT SiteWise creates a new data stream with that alias, provided no asset properties are already assigned that alias. Alternatively, if the first time series data is identified with an asset ID and property ID, AWS IoT SiteWise creates a new data stream and associates that data stream with the asset property.

There are two ways to assign an alias to an asset property. The method used depends on if data is sent to AWS IoT SiteWise first, or an asset is created first.

- If data is sent to AWS IoT SiteWise first, this automatically creates a data stream with the assigned alias. When the asset is created later, use the AssociateTimeSeriesToAssetProperty API to associate the data stream and its alias to the asset property.
- If an asset is created first, use the UpdateAssetProperty API to assign an alias to an asset property. When data is later sent to AWS IoT SiteWise, the data stream is automatically created and associated with the asset property.

Currently, you can only associate data streams with measurements. *Measurements* are a type of asset property that represent devices' raw sensor data streams, such as timestamped temperature values or timestamped rotations per minute (RPM) values.

When these measurements define metrics or transformations, the incoming data triggers specific calculations. It’s important to note that an asset property can only be linked to one data stream at a time.

AWS IoT SiteWise uses `TimeSeries` for the Amazon Resource Name (ARN) resource to determine your storage charges. For more information, see AWS IoT SiteWise Pricing.

The following sections show you how to use the AWS IoT SiteWise console or API to manage data streams.

Topics

Configure permissions and settings

Data streams are automatically created in AWS IoT SiteWise when the first time series data is received. If the data ingested is not associated with an asset property, AWS IoT SiteWise creates a new disassociated data stream which is configurable to be associated with an asset property. Configure the access control of the gateway sending data to AWS IoT SiteWise, using IAM policies to specify the type of data to be ingested.

The following IAM policy disables disassociated data ingestion from the gateway, while still allowing data ingestion to data streams associated with an asset property:

Example IAM user policy that disables disassociated data ingestion from the gateway

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
      "Sid": "AllowPutAssetPropertyValuesUsingAssetIdAndPropertyId",
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "arn:aws:iotsitewise:*:*:asset/*"
    },
    {
      "Sid": "AllowPutAssetPropertyValuesUsingAliasWithAssociatedAssetProperty",
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "arn:aws:iotsitewise:*:*:time-series/*",
      "Condition": {
        "StringLikeIfExists": {
          "iotsitewise:isAssociatedWithAssetProperty": "true"
        }
      }
    },
    {
      "Sid": "DenyPutAssetPropertyValuesUsingAliasWithNoAssociatedAssetProperty",
      "Effect": "Deny",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "arn:aws:iotsitewise:*:*:time-series/*",
      "Condition": {
        "StringLikeIfExists": {
```

```
                "iotsitewise:isAssociatedWithAssetProperty": "false"
            }
        }
    }
  ]
}
```

Example IAM user policy that disables all data ingestion from the gateway

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyPutAssetPropertyValues",
            "Effect": "Deny",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": [
                "arn:aws:iotsitewise:*:*:asset/*",
                "arn:aws:iotsitewise:*:*:time-series/*"
            ]
        }
    ]
}
```

Associate a data stream to an asset property

Manage your data streams using the AWS IoT SiteWise console or AWS CLI.

Console

Use the AWS IoT SiteWise console to manage your data streams.

To manage data streams (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Data streams**.

3. Choose a data stream by either filtering on data stream alias, or selecting **Disassociated data streams** in the filter drop down menu.
4. Select the data stream to update. You may select multiple data streams. Click **Manage data streams** on the upper right.
5. Select the data stream to be associated from **Update data stream associations**, and click the **Choose measurement** button.
6. In the **Choose measurement** section, find the corresponding asset measurement property. Select the measurement then click **Choose**.
7. Perform steps 4 and 5 for other data streams selected in step 3. Assign asset properties to all the data streams.
8. Choose **Update** to commit the changes. A successful confirmation banner is displayed to confirm the update.

AWS CLI

To associate a data stream (identified by its alias) to an asset property (identified by its IDs), run the following command:

```
aws iotsitewise associate-time-series-to-asset-property \
    --alias <data-stream-alias> \
    --assetId <asset-ID> \
    --propertyId <property-ID>
```

Disassociate a data stream from an asset property

Console

Use the AWS IoT SiteWise console to disassociate your data stream from an asset property.

To disassociate data streams from an asset property (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Data streams**.
3. Choose a data stream by either filtering on data stream alias, or selecting **Associated data streams** in the filter drop down menu.

4. Select the data stream to disassociate. The **Data stream alias** column must contain an alias. The **Asset name** and **Asset property name** columns must contain the values of the asset property the data stream is associated with. You can select multiple data streams.
5. Click **Manage data streams** on the upper right.
6. In the **Update data stream associations** section, click **X** in the **Measurement name** column. A `submitted` status should appear in the **Status** column.
7. Choose **Update** to commit the changes. The data stream is now disassociated from the asset property, and the alias is now used to identify the data stream.

AWS CLI

To disassociate a data stream from an asset property, (identified by its `IDs` and its alias), run the following command:

```
aws iotsitewise disassociate-time-series-from-asset-property \
    --alias <asset-property-alias> \
    --assetId <asset-ID> \
    --propertyId <property-ID>
```

The data stream is now disassociated from the asset property, and the alias is used to identify the data stream. The alias is no longer associated with the asset property, as it is now associated with the data stream.

Delete a data stream

When a property is removed from an asset model, AWS IoT SiteWise deletes the properties and their data streams from all assets that are managed by the asset model. It also deletes all properties and their data streams of an asset when the asset is deleted. If a data stream data must be preserved, it must be disassociated from the asset property before it is deleted.

> **Warning**
> When a property is deleted from an asset, the associated data stream is also deleted. To preserve the data stream, disassociate it from the asset property first, before deleting the property from the asset model, or deleting the asset.

Console

Use the AWS IoT SiteWise console to disassociate your data stream from an asset property.

To delete a data stream (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Data streams**.
3. Choose a data stream by filtering on data stream alias.
4. Select the data stream to delete. You may select multiple data streams.
5. Choose the **Delete** button to delete the data stream.

AWS CLI

Use the DeleteTimeSeries API to delete a specific data stream, by its alias.

```
aws iotsitewise delete-time-series \
    --alias <data-stream-alias>
```

Update an asset property alias

Aliases must be unique within an AWS region. This includes aliases of both asset properties and data streams. Do not assign an alias to an asset property, if another property or data stream is using that alias.

Console

Use the AWS IoT SiteWise console to update an asset property alias.

To update an asset property alias (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets** .
3. Select the asset from the table.
4. Click the **Edit** button.
5. Select the **Property type** in the **Properties** table.

6. Find the property, and type the new alias in the property alias text field.
7. Click the **Save** button to save the changes.

AWS CLI

To update an alias on an asset property, run the following command:

```
aws iotsitewise update-asset-property \
    --asset-id <asset-ID> \
    --property-id <property-ID> \
    --property-alias <asset-property-alias> \
    --property-notification-state <ENABLED|DISABLED>
```

> **Note**
>
> If property notifications are currently enabled, it must be provided again to ensure it continues to be enabled.

Common scenarios

Move a data stream

To change a data stream's association to another asset property, first disassociate the data stream from the current asset property. When disassociating a data stream from an asset property, there must be an alias assigned to that asset property.

```
aws iotsitewise disassociate-time-series-from-asset-property \
    --alias <asset-property-alias> \
    --assetId <asset-ID> \
    --propertyId <property-ID>
```

Now re-assign the data stream to the new asset property.

```
aws iotsitewise associate-time-series-from-asset-property \
    --alias <data-stream-alias> \
    --assetId <new-asset-ID> \
```

```
        --propertyId <new-property-ID>
```

Error when assigning an alias to an asset property

When using the `UpdateAssetProperty` API to assign an alias to a property, you may see the following error message:

```
Given alias <data-stream-alias> for property <property-name> with ID <property-ID>
 already in use by another property or data stream
```

This error message indicates the alias is not assigned to the property, because it is currently used by another property or a data stream.

This happens if data is being ingested to AWS IoT SiteWise with an alias. When data is sent with an alias not being used by another data stream or asset property, a new data stream is created with that alias. The below two options resolve the issue.

- Use `AssociateTimeSeriesToAssetProperty` API to associate the data stream with its alias to the asset property.
- Temporarily stop the data ingestion and delete the data stream. Use `UpdateAssetProperty` API to assign the alias to the asset property, and then turn data ingestion back on.

Error when associating a data stream to an asset property

When associating a data stream to an asset property, the following error message is seen.

```
assetProperty <property-name> with assetId <asset-ID> propertyId <property-ID> contains
 data
```

This error message indicates the asset property already is associated with a data stream containing data. That data stream must be disassociated or deleted, before associating an other data stream to that asset property.

> **Note**
>
> When disassociating a data stream from an asset property, the alias assigned to the property is given to the data stream. For that alias to remain assigned to the property, assign a new alias to that property before disassociating the data stream.

To preserve the data stored in the asset property do the following:

- Ensure no data is being ingested to the asset property, to prevent creating a new data stream.
- Use `UpdateAssetProperty` API to set a new alias that is given to the currently assigned data stream.
- Use `DisassociateTimeSeriesFromAssetProperty` API to disassociate the current data stream from the asset property.
- Use `AssociateTimeSeriesToAssetProperty` API to associate the desired data stream to the asset property.

If the data stored in the asset property must be deleted, do the following:

- Ensure no data is being ingested to the asset property, to prevent creating a new data stream.
- Use `DeleteTimeSeries` API to delete the currently assigned data stream.
- Use `AssociateTimeSeriesToAssetProperty` API to associate the desired data stream to the asset property.

Ingest data with AWS IoT SiteWise APIs

Use AWS IoT SiteWise APIs to send timestamped industrial data to your assets' attribute and measurement properties. The APIs accepts payload containing timestamp-quality-value (TQV) structures.

BatchPutAssetPropertyValue API

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

> **Important**
>
> The BatchPutAssetPropertyValue operation is subject to the following quotas:
>
> - Up to 10 entries per request.
> - Up to 10 property values (TQV data points) per entry.

- AWS IoT SiteWise rejects any data with a timestamp dated to more than 7 days in the past or more than 10 minutes in the future.

For more information about these quotas, see BatchPutAssetPropertyValue in the *AWS IoT SiteWise API Reference*.

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

- The `assetId` and `propertyId` of the asset property that data is sent 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 set property aliases, see Manage data streams for AWS IoT SiteWise.

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.

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

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

```
{
  "enablePartialEntryProcessing": true,
  "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": {
            "doubleValue": 15.09
          },
          "timestamp": {
            "timeInSeconds": 1575691200
          },
          "quality": "GOOD"
        }
      ]
    },
    {
  "entryId": "unique entry ID",
      "propertyAlias": "/company/windfarm/3/turbine/7/rpm",
      "propertyValues": [
        {
  "value": {
  "nullValue":{"valueType": "D"}
          },
          "timestamp": {
  "timeInSeconds": 1575691200
          },
          "quality": "BAD"
        }
      ]
    }
  ]
}
```

Specifying `enablePartialEntryProcessing` as `true` allows ingestion of all values that do not result in failure. The default behavior is `false`. If a value is invalid, the entire entry fails ingestion.

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`
  - `nullValue`
- `nullValue` – A structure with the following field denoting the type of the property value with value Null and quality of `BAD` or `UNCERTAIN`.
  - `valueType` – Enum of {"B", "D", "S", "I"}
- `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 10 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.

CreateBulkImportJob API

Use the `CreateBulkImportJob` API to import large amounts of data from Amazon S3. Your data must be saved in the CSV format in Amazon S3. Data files can have the following columns.

> **Note**
>
> Data older than 1 January 1970 00:00:00 UTC is not supported.
> To identify an asset property, specify one of the following.
>
> - The `ASSET_ID` and `PROPERTY_ID` of the asset property that you you're sending data to.

- The ALIAS, 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 the section called "Manage data streams".

- ALIAS – The alias that identifies the property, such as an OPC UA server data stream path (for example, /company/windfarm/3/turbine/7/temperature). For more information, see Manage data streams for AWS IoT SiteWise.
- ASSET_ID – The ID of the asset.
- PROPERTY_ID – The ID of the asset property.
- DATA_TYPE – The property's data type can be 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.
- TIMESTAMP_SECONDS – The timestamp of the data point, in Unix epoch time.
- TIMESTAMP_NANO_OFFSET – The nanosecond offset coverted from TIMESTAMP_SECONDS.
- QUALITY – (Optional) The quality of the asset property value. The value can be one of the following.
  - 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.
- VALUE – The value of the asset property.

Example data file(s) in the .csv format

```
asset_id,property_id,DOUBLE,1635201373,0,GOOD,1.0
asset_id,property_id,DOUBLE,1635201374,0,GOOD,2.0
asset_id,property_id,DOUBLE,1635201375,0,GOOD,3.0
```

```
unmodeled_alias1,DOUBLE,1635201373,0,GOOD,1.0
unmodeled_alias1,DOUBLE,1635201374,0,GOOD,2.0
unmodeled_alias1,DOUBLE,1635201375,0,GOOD,3.0
unmodeled_alias1,DOUBLE,1635201376,0,GOOD,4.0
unmodeled_alias1,DOUBLE,1635201377,0,GOOD,5.0
unmodeled_alias1,DOUBLE,1635201378,0,GOOD,6.0
unmodeled_alias1,DOUBLE,1635201379,0,GOOD,7.0
unmodeled_alias1,DOUBLE,1635201380,0,GOOD,8.0
unmodeled_alias1,DOUBLE,1635201381,0,GOOD,9.0
unmodeled_alias1,DOUBLE,1635201382,0,GOOD,10.0
```

AWS IoT SiteWise provides the following API operations to create a bulk import job and get information about an existing job.

- CreateBulkImportJob – Creates a new bulk import job.
- DescribeBulkImportJob – Retrieves information about a bulk import job.
- ListBulkImportJob – Retrieves a paginated list of summaries of all bulk import jobs.

Create an AWS IoT SiteWise bulk import job (AWS CLI)

Use the CreateBulkImportJob API operation to transfer data from Amazon S3 to AWS IoT SiteWise. The CreateBulkImportJob API enables ingestion of large volumes of historical data, and buffered ingestion of analytical data streams in small batches. It provides a cost-effective primitive for data ingestion. The following example uses the AWS CLI.

Important

Before creating a bulk import job, you must enable AWS IoT SiteWise warm tier or AWS IoT SiteWise cold tier. For more information, see Configure storage settings in AWS IoT SiteWise.
The CreateBulkImportJob API supports ingestion of historical data into AWS IoT SiteWise with the option to set the adaptive-ingestion-flag parameter.

- When set to `false`, the API ingests historical data without triggering computations or notifications.
- When set to `true`, the API ingests new data, calculating metrics and transforming the data to optimize ongoing analytics and notifications within seven days.

Run the following command. Replace *file-name* with the name of the file that contains the bulk import job configuration.

```
aws iotsitewise create-bulk-import-job --cli-input-json file://file-name.json
```

Example Bulk import job configuration

The following are examples of configuration settings:

- Replace *adaptive-ingestion-flag* with `true` or `false`.
  - If set to `false`, the bulk import job ingests historical data into AWS IoT SiteWise.
  - If set to `true`, the bulk import job does the following:
    - Ingests new data into AWS IoT SiteWise.
    - Calculates metrics and transforms, and supports notifications for data with a time stamp that's within seven days.
- Replace *delete-files-after-import-flag* with `true` to delete the data from the Amazon S3 data bucket after ingesting into AWS IoT SiteWise warm tier storage.
- Replace amzn-s3-demo-bucket*-for-errors* with the name of the Amazon S3 bucket to which errors associated with this bulk import job are sent.
- Replace amzn-s3-demo-bucket*-for-errors-prefix* with the prefix of the Amazon S3 bucket to which errors associated with this bulk import job are sent.

  Amazon S3 uses the prefix as a folder name to organize data in the bucket. Each Amazon S3 object has a key that is its unique identifier in the bucket. Each object in a bucket has exactly one key. The prefix must end with a forward slash (/). For more information, see [Organizing objects using prefixes](#) in the *Amazon Simple Storage Service User Guide*.
- Replace amzn-s3-demo-bucket*-data* with the name of the Amazon S3 bucket from which data is imported.
- Replace *data-bucket-key* with the key of the Amazon S3 object that contains your data. Each object has a key that is a unique identifier. Each object has exactly one key.
- Replace *data-bucket-version-id* with the version ID to identify a specific version of the Amazon S3 object that contains your data. This parameter is optional.
- Replace *column-name* with the column name specified in the .csv file.
- Replace *job-name* with a unique name that identifies the bulk import job.
- Replace *job-role-arn* with the IAM role that allows AWS IoT SiteWise to read Amazon S3 data.

> **Note**
>
> Make sure that your role has the permissions shown in the following example. Replace amzn-s3-demo-bucket-*data* with the name of the Amazon S3 bucket that contains your data. Also, replace *amzn-s3-demo-bucket-for-errors* with the name of the Amazon S3 bucket to which errors associated with this bulk import job are sent.
>
> JSON
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
>             "Action": [
>                 "s3:GetObject",
>                 "s3:GetBucketLocation"
>             ],
>             "Resource": [
>                 "arn:aws:s3:::amzn-s3-demo-bucket-data",
>                 "arn:aws:s3:::amzn-s3-demo-bucket-data/*"
>             ],
>             "Effect": "Allow"
>         },
>         {
>             "Action": [
>                 "s3:PutObject",
>                 "s3:GetObject",
>                 "s3:GetBucketLocation"
>             ],
>             "Resource": [
>                 "arn:aws:s3:::amzn-s3-demo-bucket-for-errors",
>                 "arn:aws:s3:::amzn-s3-demo-bucket-for-errors/*"
>             ],
>             "Effect": "Allow"
>         }
>     ]
> }
> ```

```
{
```

```
    "adaptiveIngestion": adaptive-ingestion-flag,
    "deleteFilesAfterImport": delete-files-after-import-flag,
    "errorReportLocation": {
       "bucket": "amzn-s3-demo-bucket-for-errors",
       "prefix": "amzn-s3-demo-bucket-for-errors-prefix"
    },
    "files": [
       {
          "bucket": "amzn-s3-demo-bucket-data",
          "key": "data-bucket-key",
          "versionId": "data-bucket-version-id"
       }
    ],
    "jobConfiguration": {
       "fileFormat": {
          "csv": {
             "columnNames": [ "column-name" ]
          }
       }
    },
    "jobName": "job-name",
    "jobRoleArn": "job-role-arn"
}
```

Example response

```
{
    "jobId":"f8c031d0-01d1-4b94-90b1-afe8bb93b7e5",
    "jobStatus":"PENDING",
    "jobName":"myBulkImportJob"
}
```

Describe an AWS IoT SiteWise bulk import job (AWS CLI)

Use the DescribeBulkImportJob API operation to retrieve information about a specific bulk import job in AWS IoT SiteWise. This operation returns details such as the job's status, creation time, and error information if the job failed. You can use this operation to monitor job progress and troubleshoot issues. To use `DescribeBulkImportJob`, you need the job ID from the `CreateBulkImportJob` operation. The API returns the following information:

- List of files being imported, including their Amazon S3 bucket locations and keys
- Error report location (if applicable)

- Job configuration details, such as file format and CSV column names
- Job creation and last update timestamps
- Current job status (for example, whether the job is in progress, completed, or failed)
- IAM role ARN used for the import job

For completed jobs, review the results to confirm successful data integration. If a job fails, examine the error details to diagnose and resolve issues.

Replace *job-ID* with the ID of the bulk import job that you want to retrieve.

```
aws iotsitewise describe-bulk-import-job --job-id job-ID
```

Example response


```
{
   "files":[
      {
         "bucket":"amzn-s3-demo-bucket1",
         "key":"100Tags12Hours.csv"
      },
      {
         "bucket":"amzn-s3-demo-bucket2",
         "key":"BulkImportData1MB.csv"
      },
      {
         "bucket":" amzn-s3-demo-bucket3",
         "key":"UnmodeledBulkImportData1MB.csv"
      }
   ],
   "errorReportLocation":{
      "prefix":"errors/",
      "bucket":"amzn-s3-demo-bucket-for-errors"
   },
   "jobConfiguration":{
      "fileFormat":{
         "csv":{
            "columnNames":[
               "ALIAS",
               "DATA_TYPE",
               "TIMESTAMP_SECONDS",
```

```
                "TIMESTAMP_NANO_OFFSET",
                "QUALITY",
                "VALUE"
            ]
        }
    }
   },
   "jobCreationDate":1645745176.498,
   "jobStatus":"COMPLETED",
   "jobName":"myBulkImportJob",
   "jobLastUpdateDate":1645745279.968,
   "jobRoleArn":"arn:aws:iam::123456789012:role/DemoRole",
   "jobId":"f8c031d0-01d1-4b94-90b1-afe8bb93b7e5"
}
```

List AWS IoT SiteWise bulk import jobs (AWS CLI)

Use the ListBulkImportJobs API operation to retrieve a list of summaries for bulk import jobs in AWS IoT SiteWise. This operation provides an efficient way to monitor and manage your data import processes. It returns the following key information for each job:

- Job ID. A unique identifier for each bulk import job
- Job name. The name you assigned to the job when creating it
- Current status. The job's current state (for example, COMPLETED, RUNNING, FAILED)

ListBulkImportJobs is particularly useful for getting a comprehensive overview of all your bulk import jobs. This can help you track multiple data imports, identify any jobs that require attention, and maintain an organized workflow. The operation supports pagination, allowing you to retrieve large numbers of job summaries efficiently. You can use the job IDs returned by this operation with the DescribeBulkImportJob operation to retrieve more detailed information about specific jobs. This two-step process allows you to first get a high-level view of all jobs, and then drill down into the details of jobs of interest. When using `ListBulkImportJobs`, you can apply filters to narrow down the results. For example, you can filter jobs based on their status to retrieve only completed jobs or only running jobs. This feature helps you focus on the most relevant information for your current task. The operation also returns a `nextToken` if there are more results available. You can use this token in subsequent calls to retrieve the next set of job summaries, enabling you to iterate through all your bulk import jobs even if you have a large number of them. The following example demonstrates how to use `ListBulkImportJobs` with the AWS CLI to retrieve a list of completed jobs.

```
aws iotsitewise list-bulk-import-jobs --filter COMPLETED
```

Example Response for completed jobs filter

```
{
   "jobSummaries":[
      {
         "id":"bdbbfa52-d775-4952-b816-13ba1c7cb9da",
         "name":"myBulkImportJob",
         "status":"COMPLETED"
      },
      {
         "id":"15ffc641-dbd8-40c6-9983-5cb3b0bc3e6b",
         "name":"myBulkImportJob2",
         "status":"COMPLETED"
      }
   ]
}
```

This command demonstrates how to use `ListBulkImportJobs` to retrieve a list of jobs that completed with failures. The maximum is set to 50 results and we're using a next token for paginated results.

```
aws iotsitewise list-bulk-import-jobs --filter COMPLETED_WITH_FAILURES --max-results 50
 --next-token "string"
```

Ingest data to AWS IoT SiteWise using AWS IoT Core rules

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*.

To follow a tutorial that walks through the steps required to set up a rule that ingests data through device shadows, see Ingest data to AWS IoT SiteWise from AWS IoT things.

You can also send data from AWS IoT SiteWise to other AWS services. For more information, see Interact with other AWS services.

Topics

Grant 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 the rule to perform actions on 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 a 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*.

For the AWS IoT SiteWise rule action, you must define a role that allows `iotsitewise:BatchPutAssetPropertyValue` access to the asset properties to which the rule sends data. To improve security, you can specify an AWS IoT SiteWise asset hierarchy path in the `Condition` property.

The following example trust policy allows access to a specific asset and its children.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
```

```
        }
      }
    }
  ]
}
```

Remove the `Condition` from the policy to allow access to all of your assets. The following example trust policy allows access to all of your assets in the current Region.

JSON

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

Configure 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,to 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 `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. You can find the Asset ID and Property ID using the AWS IoT SiteWise console. If you know the Asset ID, you can use the AWS CLI to call DescribeAsset to find the Property ID.
- 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 Manage data streams for AWS IoT SiteWise.

For the timestamp in each entry, 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
- Converting timestamps that are in string format

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

If your sensor or equipment doesn't report accurate time data, 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`), 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 **`${floor(time_to_epoch("2020-03-03T14:57:14.699Z", "yyyy-MM-dd'T'HH:mm:ss'Z'") / 1E3)}`** to convert the timestamp string to milliseconds, and then to seconds.
- For **Offset in nanos** (`offsetInNanos`), use **`${(time_to_epoch("2020-03-03T14:57:14.699Z", "yyyy-MM-dd'T'HH:mm:ss'Z'") % 1E3) * 1E6}`** 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, 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.

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, 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 your first Lambda function in the *AWS Lambda Developer Guide*.
> - AWS IoT rules with the AWS IoT SiteWise rule action. For more information, see Ingest data to AWS IoT SiteWise using AWS IoT Core rules.

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 3.11** (`python3.11`).
   - **Permissions –** Create a role with basic Lambda permissions (**AWSLambdaBasicExecutionRole**).
   - **Layers –** Add the **AWSSDKPandas-Python311** 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.

   ```
   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()
       time_in_seconds = math.floor(nanoseconds / 1E9)
       # 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 Troubleshoot an AWS IoT SiteWise rule action.

> **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 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 topic()) as the property alias to identify asset properties. 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.

```
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.

```
{
  "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, send the following message to a wind turbine property alias (for example, `/company/windfarm/3/turbine/7/temperature`) as a topic to ingest data.

```
{
  "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.

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

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

```
{
  "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, send the following message to the `my/asset/property/topic` to ingest data.

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

Troubleshooting the rule action

To troubleshoot your AWS IoT SiteWise rule action in AWS IoT Core, configure CloudWatch Logs or configure a republish error action for your rule. For more information, see Troubleshoot an AWS IoT SiteWise rule action.

Reduce costs with Basic Ingest in AWS IoT SiteWise

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.

```
$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 in the *AWS IoT Developer Guide*.

Ingest data to AWS IoT SiteWise from AWS IoT Events

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

> **Note**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS IoT Events. After May 20, 2026, you will no longer be able to access the AWS IoT Events console or AWS IoT Events resources. For more information, see AWS IoT Events end of support.

AWS IoT Events is designed to streamline the development of event monitoring applications for IoT devices and systems within the AWS Cloud. Using AWS IoT Events, you can:

- Detect and respond to changes, anomalies, or specific conditions across your IoT fleet.
- Enhance your operational efficiency and enable proactive management of your IoT ecosystem.

By integrating with AWS IoT SiteWise through the AWS IoT SiteWise action, AWS IoT Events extends its capabilities, allowing you to automatically update asset properties in AWS IoT SiteWise in response to specific events. This interaction can simplify data ingestion and management. It can also empower you with actionable insights.

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

Use AWS IoT Greengrass stream manager in AWS IoT SiteWise

AWS IoT Greengrass stream manager is an integration feature that facilitates the transfer of data streams from local sources to the AWS Cloud. It acts as an intermediary layer that manages data flows, enabling devices operating at the edge to gather and store data before it is sent to AWS IoT SiteWise, for further analysis and processing.

Add a data destination by configuring a local source on the AWS IoT SiteWise console. You can also use stream manager in your custom AWS IoT Greengrass solution to ingest data to AWS IoT SiteWise.

> **Note**
>
> To ingest data from OPC UA sources, configure an AWS IoT SiteWise Edge gateway that runs on AWS IoT Greengrass. For more information, see Use AWS IoT SiteWise Edge gateways.

For more information about how to **configure a destination** for local source data, see Understand AWS IoT SiteWise Edge destinations.

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 2 Developer Guide*:

- What is AWS IoT Greengrass?
- Manage data streams on the AWS IoT Greengrass core
- Exporting data to AWS IoT SiteWise asset properties

Use AWS IoT SiteWise Edge gateways

AWS IoT SiteWise Edge extends cloud capabilities to industrial edge environments, enabling local data processing, analysis, and decision-making. SiteWise Edge integrates with AWS IoT SiteWise and other AWS services to provide comprehensive industrial IoT solutions. Gateways serve as the intermediary between your industrial equipment and AWS IoT SiteWise.

SiteWise Edge gateways runs on two different deployment targets:

- AWS IoT Greengrass V2
- Siemens Industrial Edge

You can use a SiteWise Edge gateway to collect data at the edge and publish it to the cloud. For gateways running on AWS IoT Greengrass, you can also process data at the edge using asset models and assets.

The AWS IoT SiteWise Edge application on Siemens Industrial Edge supports integration between industrial equipment and AWS IoT SiteWise so that you can aggregate and process raw machine data and run analyses locally before sending refined data to the AWS Cloud.

Key concepts of SiteWise Edge gateways

SiteWise Edge has several useful features for edge computing in industrial environments.

Local data collection and processing

Supports data collection from industrial assets using protocols like OPC-UA and MQTT. Gateways run on AWS IoT Greengrass Core devices or Siemens Industrial Edge.

Offline operation

Continues collecting and processing data during internet outages, syncing with the cloud when connectivity is restored.

Edge computing with AWS IoT Greengrass components

Uses IoT SiteWise publisher to forward data to the cloud and AWS IoT SiteWise processor for local transformations and calculations. Both the publisher and processor are AWS IoT Greengrass V2 components. For more information on AWS IoT Greengrass components, see AWS-provided components.

Integration with AWS IoT SiteWise to extend cloud features

Works with the AWS IoT SiteWise cloud features, extending asset models, analytics, and dashboards to the edge.

For gateways with a data processing pack enabled, you can use AWS OpsHub for AWS IoT SiteWise to centrally manage your SiteWise Edge gateways. AWS OpsHub provides remote management and monitoring capabilities. For more information, see Manage SiteWise Edge gateways using AWS OpsHub for AWS IoT SiteWise.

Partner data source integration

Connect a partner data source to your gateway and receive data from the partner in your SiteWise Edge gateway and the AWS cloud. For more information, see Partner data sources on SiteWise Edge gateways.

Local visualization on the edge

Provides custom dashboards for real-time insights at the edge.

Monitor data locally in your facility using SiteWise Monitor portals on your local devices. For more information, see Enabling your AWS IoT SiteWise portal at the edge.

Benefits of implementing SiteWise Edge

SiteWise Edge offers numerous advantages that can significantly improve industrial operations and decision-making processes.

- Real-time operational insights without cloud processing delays
- Operational continuity in disconnected environments
- Reduced bandwidth and storage costs through edge pre-processing
- Increased reliability with the ability to make local, data-driven decisions

Self-host an AWS IoT SiteWise Edge gateway with AWS IoT Greengrass V2

Set up AWS IoT SiteWise Edge to collect, process, and visualize data from industrial equipment locally before sending it to the cloud. Self-host using AWS IoT Greengrass Version 2.

An AWS IoT SiteWise Edge gateway acts as the intermediary between your industrial equipment and AWS IoT SiteWise. Running on AWS IoT Greengrass Version 2, the SiteWise Edge gateway supports data collection and processing on premises. Monitor data locally within your facility through SiteWise Monitor portals on your local devices with the data processing pack enabled and AWS OpsHub installed.

There are two types of self-hosted gateways:

MQTT-enabled, V3 gateway

The MQTT-enabled, V3 gateway architecture provides improved data ingestion capabilities. It utilizes MQTT protocol for efficient data communication and offers configurable data destinations. These include options for buffered data ingestion using Amazon S3, as well as real-time data ingestion. You can implement path filters to subscribe to specific MQTT topics, enabling targeted data collection. Note that the MQTT-enabled, V3 gateway does not support the Data Processing Pack feature. For more information, see MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge.

Classic streams, V2 gateway

The Classic streams, V2 gateway represents the traditional AWS IoT SiteWise Edge gateway architecture. It is well-suited for existing SiteWise Edge deployments and users accustomed to the established workflow. While the Classic streams, V2 gateway supports the data processing pack, note that data generated by the data processing pack cannot be ingested through Amazon S3. Use the Classic streams, V2 gateway if you need to maintain compatibility with existing deployments or if you require the data processing pack functionality. For more information, see Classic streams, V2 gateways for AWS IoT SiteWise Edge.

Topics

- Configure proxy support and manage trust stores for AWS IoT SiteWise Edge
- Use AWS IoT SiteWise APIs on the edge

AWS IoT SiteWise Edge self-hosted gateway requirements

AWS IoT SiteWise Edge gateways run on AWS IoT Greengrass V2 as a set of AWS IoT Greengrass components that support data collection, processing, and publishing on premises. To configure a SiteWise Edge gateway that runs on AWS IoT Greengrass V2, create a gateway in the AWS Cloud and run the SiteWise Edge gateway software to set up your local device. When you use the AWS Management Console to create the SiteWise Edge gateway, an installation script is provided. Run this script on your target gateway device to set up necessary software and dependencies.

Local device requirements

Local devices must meet the following requirements to install and run the SiteWise Edge gateway software.

- Supports AWS IoT Greengrass V2 Core software version v2.3.0 or newer. 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 later

    Architecture: x86_64 (AMD64) or ARMv8 (Aarch64)
  - OS: Red Hat Enterprise Linux (RHEL) 8

    Architecture: x86_64 (AMD64) or ARMv8 (Aarch64)
  - OS: Amazon Linux 2

    Architecture: x86_64 (AMD64) or ARMv8 (Aarch64)
  - OS: Debian 11

    Architecture: x86_64 (AMD64) or ARMv8 (Aarch64)
  - OS: Windows Server 2019 and later

    Architecture: x86_64 (AMD64)

> **Note**
>
> ARM platforms support SiteWise Edge gateways with Data Collection Pack only. The data processing pack is not supported.

- Minimum 4 GB RAM.
- Minimum 10 GB disk space available for the SiteWise Edge gateway software.
- Configure your local device to make sure that the proper ports are accessible. For a full list of the required outbound service endpoints, see Required service endpoints for AWS IoT SiteWise Edge gateways.
- Java Runtime Environment (JRE) version 11 or higher. Java must be available on the PATH environment variable on the device. To use Java to develop custom components, you must install a Java Development Kit (JDK). We recommend that you use Amazon Corretto or OpenJDK.

Amazon S3 buckets to allowlist for local devices

Configure your local device to provide firewall access the following Amazon S3 bucket. Configure access based on the respective regions for your devices.

Region	Endpoint
Asia Pacific (Tokyo)	https://iot-sitewise-gateway-ap-northeast-1-785558802005.s3.ap-northeast-1.amazonaws.com
Asia Pacific (Seoul)	https://iot-sitewise-gateway-ap-northeast-2-310055672453.s3.ap-northeast-2.amazonaws.com
Asia Pacific (Mumbai)	https://iot-sitewise-gateway-ap-south-1-677656657204.s3.ap-south-1.amazonaws.com
Asia Pacific (Singapore)	https://iot-sitewise-gateway-ap-southeast-1-475191558554.s3.ap-southeast-1.amazonaws.com
Asia Pacific (Sydney)	https://iot-sitewise-gateway-ap-southeast-2-396319432685.s3.ap-southeast-2.amazonaws.com

Region	Endpoint
Canada (Central)	https://iot-sitewise-gateway-ca-central-1-842060018567 .s3.ca-central-1.amazonaws.com
China (Beijing)	https://iot-sitewise-gateway-cn-north-1-237124890262.s3.cn-north-1.amazonaws.com.cn
Europe (Frankfurt)	https://iot-sitewise-gateway-eu-central-1-748875242063 .s3.eu-central-1.amazonaws.com
Europe (Ireland)	https://iot-sitewise-gateway-eu-west-1-383414315062.s3.eu-west-1.amazonaws.com
US East (N. Virginia)	https://iot-sitewise-gateway-us-east-1-223558168232.s3.us-east-1.amazonaws.com and https://iot-sitewise-gateway-us-east-1-223558168232.s3.amazonaws.com/
US East (Ohio)	https://iot-sitewise-gateway-us-east-2-005072661813.s3.us-east-2.amazonaws.com
AWS GovCloud (US-West)	https://iot-sitewise-gateway-us-gov-west-1-59998456567 9.s3.us-gov-west-1.amazonaws.com/
US West (Oregon)	https://iot-sitewise-gateway-us-west-2-502577205460.s3.us-west-2.amazonaws.com

Data processing pack requirements

- If you plan to use the data processing pack at the edge with AWS IoT SiteWise, your local 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 32 GB for RAM if using Microsoft Windows.
  - Had at least 256 GB of free disk space.
  - The local device must allow network inbound traffic on port 443.

- The following ports are reserved for use by AWS IoT SiteWise: 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8445, 8086, 9000, 9500, 11080, and 50010. Using a reserved port for traffic can result in a terminated connection.

  > **Note**
  >
  > The AWS IoT Greengrass V2 Stream manager component has its own requirements. For more information, see Configuration in the *AWS IoT Greengrass Version 2 Developer Guide*.

- The minimum disk space and compute capacity requirements depend on a variety of factors that are unique to your implementation and use case.
  - 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 permissions to use SiteWise Edge gateways

You must have the following permissions to use SiteWise Edge gateways:

> **Note**
>
> If you use the AWS IoT SiteWise console to create your SiteWise Edge gateway, these permissions are added for you.

- The IAM role for your SiteWise Edge gateway must allow you to use an SiteWise Edge 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:

- `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": [
                "iotsitewise:BatchPutAssetPropertyValue",
                "iotsitewise:List*",
                "iotsitewise:Describe*",
                "iotsitewise:Get*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeCertificate",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:DescribeEndpoint"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

Create a self-hosted SiteWise Edge gateway

Use the AWS IoT SiteWise console or AWS CLI to create a self-hosted SiteWise Edge gateway. This procedure details how to create a self-hosted SiteWise Edge gateway that you'll install on your own hardware. For information about creating a SiteWise Edge gateway that runs on Siemens Industrial Edge, see Host a SiteWise Edge gateway on Siemens Industrial Edge.

Create a SiteWise Edge gateway

Console

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose **Create gateway**.
4. For **Choose deployment target**, choose **self-hosted gateway**.
5. Select either **MQTT-enabled, V3 gateway** or **Classic streams, V2 gateway**. For more information on each option, see Self-host an AWS IoT SiteWise Edge gateway with AWS IoT Greengrass V2. The MQTT-enabled, V3 gateway is recommend for it's future-ready features.
6. In the **Gateway configuration** section, enter a name for your SiteWise Edge gateway or use the name generated by AWS IoT SiteWise.
7. Under **Greengrass device OS**, select the operating system of the device where you'll install this SiteWise Edge gateway.

   Note

   The data processing pack is only available on x86 platforms. It is only available on the Classic streams, V2 gateway

8. (Optional) To process and organize data at the edge, under **Edge capabilities**, select **Data Processing Pack**.

> **Note**
>
> To grant user groups in your corporate directory access to this SiteWise Edge gateway, see [Set up edge capability in SiteWise Edge]

9. (Optional) Under **advanced configuration**, do the following:

   - 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*.

10. Choose **Create gateway**.

11. In the **Generate SiteWise Edge gateway installer** dialog box, choose **Generate and download**. AWS IoT SiteWise automatically generates an installer that you can use to configure your local device.

> **Important**
>
> You can't regenerate this file. Make sure that you save the installer file in a secure location because you'll use the file later.

AWS CLI

To create a self-hosted gateway by using the AWS CLI, provide a name for the gateway, specify the platform, and the gateway version. There are many other options that you can specify when creating a gateway. For more information, see [create-gateway] in the *AWS CLI Command Reference* for AWS IoT SiteWise

To use this example, replace the user input placeholders with your own information.

```
aws iotsitewise create-gateway \
    --gateway-name your-gateway-name \
    --gateway-platform greengrassV2={coreDeviceThingName=your-core-device-thing-
name, coreDeviceOperatingSystem=LINUX_AMD64} \
    --gateway-version 3 \
    [--cli-input-json your-configuration]
```

- `gateway-name` – A unique name for the gateway.
- `gateway-platform` – Specifies the gateway platform configuration. For self-hosted gateways, enter `greengrassV2`. For more information, see [Options](#) in the create-gateway section of *AWS CLI Command Reference* for AWS IoT SiteWise.
  - `coreDeviceThingName` – The name of the AWS IoT thing for your AWS IoT Greengrass V2 core device.
  - `coreDeviceOperatingSystem` – The operating system of the core device in AWS IoT Greengrass V2. Specifying the operating system is required for `gateway-version 3` and not applicable for `gateway-version 2`. Options include: `LINUX_AARCH64`, `LINUX_AMD64`, and `WINDOWS_AMD64`
- `gateway-version` – The version of the gateway.
  - To create an MQTT-enabled, V3 gateway, use 3 for the gateway version.
  - To create a Classic streams, V2 gateway, use 2 for the gateway version.
- `cli-input-json` – A JSON file containing request parameters.

Now that you've created the SiteWise Edge gateway, Install the AWS IoT SiteWise Edge gateway software on your local device.

Install the AWS IoT SiteWise Edge gateway software on your local device

After you've created an AWS IoT SiteWise Edge gateway, install the SiteWise Edge gateway software on your local device. SiteWise Edge gateway software can be installed on local devices that have Linux or Microsoft Windows server operating systems installed.

> **Important**
>
> Make sure that your local device connects to the internet.

Linux

The following procedure uses SSH to connect to your local device. Alternatively, you can use a USB flash drive or other tools to transfer the installer file to your local device. If you don't want to use SSH, skip to **Step 2: Install the SiteWise Edge gateway software** below.

SSH prerequisites

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

- Linux and macOS - Download and install OpenSSH. For more information, see https://www.openssh.com.

Step 1: Copy the installer to your SiteWise Edge gateway device

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

1. To connect to your device, run the following command in a terminal window on your computer, replacing *username* and *IP* with a username that has elevated privileges and IP address.

```
ssh username@IP
```

2. To transfer the installer file that AWS IoT SiteWise generated to your SiteWise Edge 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 local device.
> - Replace *directory-to-receive-installer* with the path on your local 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: Install the SiteWise Edge gateway software

In the following procedures, run the commands in a terminal window on your SiteWise Edge 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
   ```

Windows Server

Prerequisites

You must have the following prerequisites to install the SiteWise Edge gateway software:

- Microsoft Windows Server 2019 or later installed
- Administrator privileges
- PowerShell version 5.1 or later installed
- SiteWise Edge gateway installer downloaded to the Windows Server where it will be provisioned

Step 1: Run PowerShell as administrator

1. On the Windows server where you want to install SiteWise Edge gateway, log in as administrator.
2. Enter **PowerShell** in the Windows search bar.
3. In the search results, open the context (right-click) menu on the Windows PowerShell app. Choose **Run as Administrator**.

Step 2: Install the SiteWise Edge gateway software

Run the following commands in a terminal window on your SiteWise Edge Gateway device.

1. Unblock the SiteWise Edge gateway installer.

```
unblock-file path-to-installer.ps1
```

2. Run the Installer.

```
./path-to-installer.ps1
```

> **Note**
>
> If the script execution is disabled on the system, change the script execution policy to `RemoteSigned`.
>
> ```
> Set-ExecutionPolicy RemoteSigned
> ```

The next step depends on the *type* of self-hosted gateway you need. Continue to MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge or Classic streams, V2 gateways for AWS IoT SiteWise Edge.

MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge

AWS IoT SiteWise can use MQTT-enabled, V3 gateways, representing a significant advancement in the SiteWise Edge gateway architecture. This gateway type leverages the MQTT (Message Queuing Telemetry Transport) protocol for data communication, providing enhanced flexibility and efficiency in industrial IoT deployments.

The MQTT-enabled, V3 gateway uses MQTT for data transfer, enabling a lightweight, publish-subscribe network protocol that efficiently transports messages between devices and the cloud. You can set up various data destinations, including real-time data ingestion directly into AWS IoT SiteWise and buffered data ingestion using Amazon S3. To enable precise data collection, you can implement path filters to subscribe to specific MQTT topics.

MQTT-enabled, V3 gateways come with a pre-configured real-time destination with filters set to "#" (all topics), which you can customize or remove as needed. To streamline data management, only one real-time destination can exist in each gateway.

The MQTT-enabled architecture differs significantly from the Classic streams, V2 gateway. While V2 uses a stream-based approach, V3 employs MQTT, offering more configurable data destinations and filtering options. However, note that V3 does not support the data processing pack, which is available in V2.

The MQTT-enabled, V3 gateway offers several advantages:

- Improved scalability, due to MQTT's lightweight nature, enabling better handling of numerous devices and high-frequency data transmission.
- Enhanced data control through path filters, enabling granular management of data collection and reducing unnecessary data transfer and processing.
- Flexible data handling, allowing configuration between real-time processing and buffered storage based on specific needs.
- Alignment with modern IoT communication standards, setting the stage for future enhancements and integrations.

Consider adopting the MQTT-enabled, V3 gateway for new deployments, especially when you require flexible data ingestion options and precise control over data collection.

Note

For existing deployments or scenarios requiring the data processing pack, the Classic streams, V2 gateway remains a viable option.

By offering both gateway types, AWS IoT SiteWise ensures that you can choose the solution that best fits your specific industrial IoT needs, whether you prioritize advanced MQTT capabilities or compatibility with existing systems.

Destinations and path filters

View the following topics to learn more about destinations and path filters in MQTT-enabled gateways:

- Understand AWS IoT SiteWise Edge destinations

- [Add an AWS IoT SiteWise Edge real-time destination]
- [Add an AWS IoT SiteWise buffered destination using Amazon S3]
- [Understand path filters for AWS IoT SiteWise Edge destinations]
- [Add path filters to AWS IoT SiteWise Edge destinations]
- [Manage AWS IoT SiteWise Edge destinations]

Connect external applications to the EMQX broker

This guide explains how to connect external applications to your AWS IoT SiteWise Edge gateway through an EMQX broker on your deployed MQTT-enabled, V3 gateway. External applications might include custom monitoring tools, third-party visualization software, or legacy systems that need to interact with your industrial data at the edge.

We'll cover the configuration steps for both Linux and Microsoft Windows environments, including EMQX deployment configuration, TLS setup for secure connections, and authorization rules to control access to specific topics.

> **Note**
>
> EMQX is not a vendor or supplier for AWS IoT SiteWise Edge.

> **Important**
>
> For securing connections to your gateway, we strongly recommend using certificate-based authentication through the AWS IoT Greengrass client device authentication feature. This method provides robust security through mutual TLS (mTLS) authentication. For more information, see Connect client devices to core devices in the *AWS IoT Greengrass Version 2 Developer Guide*.

If you are not able to use certificate based authentication, follow this guide to setup authentication using usernames and passwords.

Prerequisites

- A SiteWise Edge MQTT-enabled, V3 gateway that has been deployed and is online

- Access to the gateway host
- Access to the AWS IoT SiteWise and AWS IoT Greengrass consoles

Topics

- Message payload format for the EMQX broker on AWS IoT SiteWise Edge
- Configure the EMQX broker
- Connect an application to the EMQX broker on AWS IoT SiteWise Edge
- Set up authorization rules for AWS IoT SiteWise Edge in EMQX

Message payload format for the EMQX broker on AWS IoT SiteWise Edge

For the IoT SiteWise publisher component to consume data from your external application and publish it to the AWS IoT SiteWise cloud, the payload sent to the broker must meet specific requirements.

Understanding the payload format is key to successful MQTT communication with AWS IoT SiteWise Edge. While the connection setup process is covered in later sections, we present the payload requirements first to help you plan your implementation.

MQTT topic requirements

There are no restrictions on MQTT topic structure, including the number of levels or characters used. However, we recommend that the topic matches the `propertyAlias` field in the payload.

Example Example property alias

If the MQTT topic is `site1/line1/compressor1/temperature`, ensure the `propertyAlias` matches.

```
{
  "assetId": "compressor_asset_01",
  "propertyAlias": "site1/line1/compressor1/temperature",
  "propertyId": "temperature_sensor_01",
  "propertyValues": [
    {
      "quality": "GOOD",
      "timestamp": {
        "offsetInNanos": 0,
```

```
          "timeInSeconds": 1683000000
        },
        "value": {
          "doubleValue": 23.5
        }
      }
    ]
}
```

JSON payload structure

The MQTT message payload are written in JSON and follow the `PutAssetPropertyValueEntry` message format defined in the AWS IoT SiteWise API Reference.

```
{
   "assetId": "string",
   "propertyAlias": "string",
   "propertyId": "string",
   "propertyValues": [
      {
         "quality": "string",
         "timestamp": {
            "offsetInNanos": number,
            "timeInSeconds": number
         },
         "value": {
            "booleanValue": boolean,
            "doubleValue": number,
            "integerValue": number,
            "stringValue": "string"
         }
      }
   ]
}
```

Note

For a message to be considered valid, only one of the following conditions can be true:

- The `propertyAlias` is set, or
- Both `assetId` and `propertyId` are set

The `PutAssetPropertyValueEntry` has an `entryId` field that is not required in this context.

Configure the EMQX broker

This section covers how to add usernames and passwords. It also covers how to establish a TLS connection from an external source using the added username and password. You can configure the EMQX broker using Linux or Microsoft Windows.

Note

To configure the broker, you need a core device that is setup with the default EMQX configuration in your MQTT-enabled, V3 gateway.

Important

After completing this procedure, we highly recommend configuring authorization rules. For more information, see Set up authorization rules for AWS IoT SiteWise Edge in EMQX. Authorization rules for added users enhances security.

Update the EMQX deployment configuration for authentication

To update the EMQX deployment configuration for authentication

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Choose the gateway to configure.
4. In the **Edge gateway configuration** section, copy your **Greengrass core device** value. Save it for later use.
5. Open the AWS IoT console.
6. On the left navigation, under the **Manage** section, choose **Greengrass devices**, then **Deployments**.
7. Find the core device value you saved earlier and choose that link to open the deployment.

8. Choose the **Actions** dropdown button, then **Revise**.
9. Read the message that appears and then choose **Revise deployment**. The **Specify target** page appears.
10. Choose **Next** until you reach the **Configure components** step.
11. Select the `aws.greengrass.clientdevices.mqtt.EMQX` radio button.
12. Choose the **Configure component** button. A configuration page appears for the component.
13. Under **Configuration update**, choose **Reset to default configuration for component version: 2.*.***.
14. Enter the following configuration in the **Configuration to merge** section based on your OS.

Linux

```
{
    "emqxConfig": {
        "authorization": {
            "no_match": "allow"
        },
        "listeners": {
            "tcp": {
                "default": {
                    "enabled": true,
                    "enable_authn": false
                }
            },
            "ssl": {
                "default": {
                    "enabled": true,
                    "enable_authn": true,
                    "ssl_options": {
                        "verify": "verify_none",
                        "fail_if_no_peer_cert": false
                    }
                }
            }
        },
        "authentication": {
            "enable": true,
            "backend": "built_in_database",
            "mechanism": "password_based",
            "password_hash_algorithm": {
```

```
                "iterations": 210000,
                "mac_fun": "sha512",
                "name": "pbkdf2"
            },
            "user_id_type": "username"
        },
        "dashboard": {
            "listeners": {
                "http": {
                    "bind": 18083
                }
            }
        }
    },
    "authMode": "bypass",
    "dockerOptions": "-p 8883:8883 -p 127.0.0.1:1883:1883
 -p 127.0.0.1:18083:18083 -v emqx-data:/opt/emqx/data -e
 EMQX_NODE__NAME=emqx@local",
    "requiresPrivilege": "true"
}
```

Windows

```
{
    "emqxConfig": {
        "authorization": {
            "no_match": "allow"
        },
        "listeners": {
            "tcp": {
                "default": {
                    "enabled": true,
                    "enable_authn": false
                }
            },
            "ssl": {
                "default": {
                    "enabled": true,
                    "enable_authn": true,
                    "ssl_options": {
                        "verify": "verify_none",
                        "fail_if_no_peer_cert": false
                    }
```

```
                }
            }
        },
        "authentication": {
            "enable": true,
            "backend": "built_in_database",
            "mechanism": "password_based",
            "password_hash_algorithm": {
                "iterations": 210000,
                "mac_fun": "sha512",
                "name": "pbkdf2"
            },
            "user_id_type": "username"
        },
        "dashboard": {
            "listeners": {
                "http": {
                    "bind": 18083
                }
            }
        }
    },
    "authMode": "bypass",
    "requiresPrivilege": "true"
}
```

The `dockerOptions` field is only for Linux gateways.

15. Choose **Confirm**.
16. Choose **Next** until you reach the **Review** step.
17. Choose **Deploy**.
18. After the deployment succeeds, proceed to the next step.

Enable username and password authentication

This section shows you how to add usernames and passwords through the EMQX dashboard GUI.

> **Note**
>
> The EMQX-related instructions provided are for reference only. As EMQX documentation and features may change over time, and we do not maintain their documentation, we recommend consulting EMQX's official documentation for the most current information.

EMQX Dashboard

To enable username and password authentication through the EMQX dashboard

1. Ensure that you are within the gateway host.
2. Open a browser window and visit http://localhost:18083/.
3. Enter the default username of **`admin`** and the default password of **`public`**. For more information, see EMQX Dashboard in the *EMQX Docs*.
4. After login, you are prompted to change your password. Update your password to continue to the EMQX Dashboard.
5. In the left navigation, choose the shield icon, then **Authentication**.
6. In the **Built-in Database** row, choose the **Users** button.
7. Choose the plus sign icon button to add users. An **Add** screen appears.
8. Enter a username and password for the user of the external application.
9. Choose **Save**. The username you chose appears in the **Authentication** page's table.

> **Note**
>
> Existing or default authorization rules apply to the new user. It's recommended to review and adjust them to your external application needs.

EMQX Management with Linux

Use the AWS IoT SiteWise EMQX CLI tool at `/greengrass/v2/bin/swe-emqx-cli`.

To enable username and password authentication through EMQX Management using Linux

1. Change the admin password by running the following command:

```
/greengrass/v2/bin/swe-emqx-cli admin change-pwd
```

2. When prompted, do the following:

   1. Enter your current administrator user (default is `admin`) and password (default is `public`).
   2. Enter and confirm your new password.

   If successful, you see the following message:

```
admin password changed successfully
```

3. Add users for external applications by running the following command:

```
/greengrass/v2/bin/swe-emqx-cli users add
```

4. When prompted, do the following:

   1. Enter the username for the new user.
   2. Enter and confirm the password for the new user.

   If successful, you see the following message:

```
User '[username]' created successfully
```

5. Verify user configuration by running the following command:

```
/greengrass/v2/bin/swe-emqx-cli users list
```

   The output shows all configured users:

```
Users:
- [your-added-username]

Total users: 1
```

EMQX Management with Windows

Use the AWS IoT SiteWise EMQX CLI tool at one of the following locations:

- PowerShell: `C:\greengrass\v2\bin\swe-emqx-cli.ps1`
- Command Prompt: `C:\greengrass\v2\bin\swe-emqx-cli.bat`

To enable username and password authentication through EMQX Management using Windows

1. Change the admin password by running the following command:

   ```
   C:\greengrass\v2\bin\swe-emqx-cli.ps1 admin change-pwd
   ```

2. When prompted, do the following:

   1. Enter your current administrator user (default is `admin`) and password (default is `public`).
   2. Enter and confirm your new password.

   If successful, you see the following message:

   ```
   admin password changed successfully
   ```

3. Add users for external applications by running the following command:

   ```
   C:\greengrass\v2\bin\swe-emqx-cli.ps1 users add
   ```

4. When prompted, do the following:

   1. Enter the username for the new user.
   2. Enter and confirm the password for the new user.

   If successful, you see the following message:

   ```
   User '[username]' created successfully
   ```

5. Verify user configuration by running the following command:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 users list
```

The output shows all configured users:

```
Users:
- [your-added-username]

Total users: 1
```

Connect an application to the EMQX broker on AWS IoT SiteWise Edge

The EMQX broker uses Transport Layer Security (TLS) on port 8883 to encrypt all communications, ensuring your data remains protected during transmission. This section walks you through the steps to establish connections between your applications and the EMQX broker. Following these steps helps maintain the integrity and confidentiality of your industrial data. The connection process involves two main approaches: using automated IP discovery through components, or manually configuring DNS names and IP addresses as Subject Alternative Names (SANs) in your TLS certificates. Each method has its own advantages depending on your network setup and security requirements. This documentation will guide you through both options.

Topics

- Configure TLS for secure connections to the EMQX broker on AWS IoT SiteWise Edge
- Test the EMQX broker connection on AWS IoT SiteWise Edge
- Use your own CA
- Open port 8883 for external firewall connections

Configure TLS for secure connections to the EMQX broker on AWS IoT SiteWise Edge

By default, AWS IoT Greengrass generates a TLS server certificate for the EMQX broker that is signed by the core device certificate authority (CA). For more information, see Connecting client devices to an AWS IoT Greengrass Core device with an MQTT broker.

Retrieve the TLS certificate

To get the CA certificate run the following command on the gateway host:

Linux

Run the following command in a shell session on the gateway host:

```
/greengrass/v2/bin/swe-emqx-cli cert
```

This command displays the certificate location and prints the certificate's content.

You can alternatively save the certificate to a file using this command:

```
/greengrass/v2/bin/swe-emqx-cli cert --output /path/to/certificate.pem
```

Windows

Run the following command in a PowerShell session on the gateway host:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 cert
```

This command displays the certificate location and prints the certificate's content.

You can alternatively save the certificate to a file using this command:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 cert --output C:\path\to\certificate.pem
```

The CLI automatically locates the certificate regardless of the exact path on your system.

Copy the contents of the ca.pem file to the external application that you're connecting to the broker. Save it as `BrokerCoreDeviceCA.pem`.

Add custom DNS names/IP addresses to the TLS server certificate

The subject alternative name (SAN) on the cert generated by AWS IoT Greengrass is `localhost`. When establishing a TLS connection from outside of the gateway host, the TLS verification step fails because the broker's hostname does not match the hostname of `localhost` on the server certificate.

To address mismatched hostname issue, AWS IoT Greengrass provides two ways of managing core device endpoints. This section covers both options. For more detailed information, see [Manage core device endpoints](#) in the *AWS IoT Greengrass Version 2 Developer Guide*.

- To connect to the EMQX broker using the core device's IP address, use the Automated IP discovery section.
- To connect to the EMQX broker using a DNS name instead of IP address, you use the Manual management section.

Automated IP discovery

This option allows your core device to automatically discover its IP address and add it as a Subject Alternative Name (SAN) to the broker certificate.

1. Add the `aws.greengrass.clientdevices.IPDetector` component to your core device’s deployment.
2. Deploy the changes to your device
3. Wait for deployment to complete.

   After the deployment completes, you can establish a secure TLS connection using the broker's IP address.

   The IP address is automatically added as a SAN to the broker certificate.

Manual DNS and IP Configuration

You can manually add DNS names and IP addresses as Subject Alternative Names (SANs) to your TLS certificate. This method is useful when you have configured a DNS name for your gateway host.

> **Important**
>
> If you are using the IPDetector component, remove it from your deployment before proceeding. The IPDetector component overrides manual endpoint configurations.

To manually configure endpoints

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Choose the gateway to configure.

4. In the **Edge gateway configuration** section, choose your **Greengrass core device** url. The core device's page appears.
5. Choose the **Client devices** tab.
6. Choose **Manage endpoints**.
7. In the Manage endpoints dialog box, enter the DNS name(s) and any IP addresses you want to add as SANs. Use port 8883.
8. Choose **Update**.

The broker's TLS server certificate updates automatically to include your new endpoints.

To verify the TLS server certificate update using Linux

1. Start a shell session on your gateway host.

```
docker exec emqx openssl x509 -in ./data/cert.pem -text -noout | grep -A1
 "Subject Alternative Name"
```

2. The command returns an output similar to the following:

```
X509v3 Subject Alternative Name:
DNS:endpoint_you_added, DNS:localhost
```

3. Verify that your endpoint appears in the list of SANs.

To verify the TLS server certificate update using Windows

1. Start a shell session on your gateway host.

```
(Get-PfxCertificate -FilePath "C:\greengrass\v2\work
\aws.greengrass.clientdevices.mqtt.EMQX\v2\data\cert.pem").Extensions | Where-
Object { $_.Oid.FriendlyName -eq "Subject Alternative Name" } | ForEach-Object
 { "Subject Alternative Name:", ($_.Format($true) -split "`n")[0..1] }
```

2. The command returns an output similar to the following:

```
Subject Alternative Name:
DNS Name=your-endpoint
DNS Name=localhost
```

3. Verify that the endpoint you added is in the list of SANs.

Test the EMQX broker connection on AWS IoT SiteWise Edge

After configuring your EMQX broker with TLS certificates and authentication credentials, it's important to verify that your setup works correctly. Testing the connection helps ensure that your security configurations are properly implemented and that clients can successfully establish encrypted connections to the broker. This section demonstrates how to test your broker connection using the Mosquitto command line interface (CLI) client, a widely-used MQTT client tool that supports TLS encryption and authentication.

Use Mosquitto CLI client to test the EMQX broker connection

In this step we will use the mosquitto CLI client to test our setup and make sure we can connect successfully to the broker using the username and password we created earlier. To get the `BrokerCoreDeviceCA.pem` follow steps under Step 3: Setting up TLS.

```
mosquitto_sub -h hostname|ip address \
    -p 8883 \
    -t "#" \
    -q 1 \
    -u username -P password \
    --cafile BrokerCoreDeviceCA.pem
```

> **Note**
>
> You may get an SSL:verify error if the hostname/IP address you are connecting to does not match the Subject Alternative Name (SAN) that is on the CA cert you're passing to the client. See "Adding custom DNS names/IP addresses to the TLS server cert" under Step 3: Setting up TLS for how to get a certificate with the correct SAN.

At this point, all users have access to publish and subscribe to all topics on the broker. Proceed to [Set up authorization rules for AWS IoT SiteWise Edge in EMQX](#).

Use your own CA

AWS IoT Greengrass outlines how to configure your own client device auth component to use your own certificate authority (CA). The client device auth component (`aws.greengrass.clientdevices.Auth`) authenticates client devices and authorizes client device actions. For more information, see Using your own certificate authority in the *AWS IoT Greengrass Version 2 Developer Guide*.

To use your own CA, add the `aws.greengrass.clientdevices.Auth` component to your deployment so that you can specify a custom configuration.

Open port 8883 for external firewall connections

Linux

In your Linux host firewall rule, add an inbound rule for port 8883 to allow incoming connections from outside of the gateway host. If there are any firewalls in place, ensure that incoming TLS connections on port 8883 are allowed.

Windows

In your Microsoft Windows host firewall rule, add an inbound rule for port 8883 to allow incoming connections from outside of the gateway host. Ensure the rule is an allow rule, of type port, specifying port 8883. You can configure this according to your network configuration to allow connections from your external applications to the broker.

Set up authorization rules for AWS IoT SiteWise Edge in EMQX

EMQX supports adding authorization rules based on identifiers such as username, IP address or client ID. This is useful if you want to limit the number of external applications connecting to various operations or topics.

Topics

Configure authorization using the built-in database with Linux

When you configure authorization rules, there are two configuration choices that depend on your deployment setup.

- **Docker** – If you're running a standard Docker installation without Litmus Edge, use the **Docker bridge gateway** configuration. This is typically the case when you've only deployed AWS IoT SiteWise components.
- **Litmus Edge** – If you have Litmus Edge installed on your gateway, use the **Litmus Edge network subnet** configuration.

> **Note**
>
> If you initially configure the Docker bridge gateway and later install Litmus Edge, reconfigure the authorization rules using the Litmus Edge network subnet option to ensure proper communication between all components.

To add basic authorization rules

1. Verify that the EMQX broker is deployed and running.
2. Start a shell session on your gateway host.

   Docker without Litmus Edge

   For standard Docker installation without Litmus Edge, run:

   ```
   /greengrass/v2/bin/swe-emqx-cli acl init
   ```

   Litmus Edge network subnet

   If you're using Litmus Edge, determine the Litmus Edge network subnet IP:

   ```
   docker network inspect LitmusNetwork | grep IPAM -A9
   ```

   Note the Subnet value from the output and run the following command. Replace `litmus_subnet_ip` with the Subnet value from the previous step.

   ```
   /greengrass/v2/bin/swe-emqx-cli acl init litmus_subnet_ip
   ```

The tool automatically creates and applies authorization rules to allow connections from the provided IP address to the broker. It allows access to all topics. This includes the IoT SiteWise OPC UA collector and IoT SiteWise publisher.

3. Proceed to Update the EMQX deployment configuration for authorization.

Configure authorization using the built-in database with Windows

This section covers configuring authorization rules using the built-in database for Windows deployments.

To add basic authorization rules

1. Verify that the EMQX broker is deployed and running.
2. Run the AWS IoT SiteWise EMQX CLI tool:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 acl init
```

   The tool automatically creates and applies ACL rules allowing connections from localhost (127.0.0.1) to the broker. It allows access to all topics. This includes the IoT SiteWise OPC UA collector and IoT SiteWise publisher.
3. Proceed to Update the EMQX deployment configuration for authorization.

Update the EMQX deployment configuration for authorization

To update the EMQX deployment configuration for authorization

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Choose the gateway to configure.
4. In the **Edge gateway configuration** section, copy your **Greengrass core device** value. Save it for later use.
5. Open the AWS IoT console.
6. On the left navigation, under the **Manage** section, choose **Greengrass devices**, then **Deployments**.

7. Find the core device value you saved earlier and choose that link to open the deployment.
8. Choose the **Actions** dropdown button, then **Revise**.
9. Read the message that appears and then choose **Revise deployment**. The **Specify target** page appears.
10. Choose **Next** until you reach the **Configure components** step.
11. Select the `aws.greengrass.clientdevices.mqtt.EMQX` radio button.
12. Choose the **Configure component** button. A configuration page appears for the component.
13. Under **Configuration update**, choose **Reset to default configuration for component version: 2.*.***.
14. Paste the following content in the **Configuration to merge** section based on your OS.

    Linux

```
{
    "emqxConfig": {
        "authorization": {
            "no_match": "deny",
            "sources": [
                {
                    "type": "built_in_database"
                },
                {
                    "type": "file",
                    "path": "data/authz/acl.conf"
                }
            ]
        },
        "listeners": {
            "tcp": {
                "default": {
                    "enabled": true,
                    "enable_authn": false
                }
            },
            "ssl": {
                "default": {
                    "enabled": true,
                    "enable_authn": true,
                    "ssl_options": {
                        "verify": "verify_none",
```

```
                        "fail_if_no_peer_cert": false
                    }
                }
            }
        },
        "authentication": {
            "enable": true,
            "backend": "built_in_database",
            "mechanism": "password_based",
            "password_hash_algorithm": {
                "iterations": 210000,
                "mac_fun": "sha512",
                "name": "pbkdf2"
            },
            "user_id_type": "username"
        },
        "dashboard": {
            "listeners": {
                "http": {
                    "bind": 18083
                }
            }
        }
    },
    "authMode": "bypass",
    "dockerOptions": "-p 8883:8883 -p 127.0.0.1:1883:1883
 -p 127.0.0.1:18083:18083 -v emqx-data:/opt/emqx/data -e
 EMQX_NODE__NAME=emqx@local",
    "requiresPrivilege": "true"
}
```

Windows

```
{
    "emqxConfig": {
        "authorization": {
            "no_match": "deny",
            "sources": [
                {
                    "type": "built_in_database"
                },
                {
                    "type": "file",
```

```
                    "path": "C:\\greengrass\\v2\\work\
\aws.greengrass.clientdevices.mqtt.EMQX\\v2\\data\\authz\\acl.conf"
                }
            ]
        },
        "listeners": {
            "tcp": {
                "default": {
                    "enabled": true,
                    "enable_authn": false
                }
            },
            "ssl": {
                "default": {
                    "enabled": true,
                    "enable_authn": true,
                    "ssl_options": {
                        "verify": "verify_none",
                        "fail_if_no_peer_cert": false
                    }
                }
            }
        },
        "authentication": {
            "enable": true,
            "backend": "built_in_database",
            "mechanism": "password_based",
            "password_hash_algorithm": {
                "iterations": 210000,
                "mac_fun": "sha512",
                "name": "pbkdf2"
            },
            "user_id_type": "username"
        },
        "dashboard": {
            "listeners": {
                "http": {
                    "bind": 18083
                }
            }
        }
    },
    "authMode": "bypass",
    "requiresPrivilege": "true"
```

```
}
```

15. Choose **Confirm**.
16. Choose **Next** until you reach the **Review** step.
17. Choose **Deploy**.

> **Note**
>
> From this point onward, you can't edit the ACL file to update the authorization rules. Alternatively, you can proceed to Add authorization rules through the EMQX Dashboard for users after a successful deployment.

Add authorization rules through the EMQX Dashboard for users

You can add or update authorization rules using the EMQX Dashboard or the AWS IoT SiteWise EMQX CLI tool. The AWS IoT SiteWise EMQX CLI tool manages authorization using EMQX's built-in database.

> **Note**
>
> Adding authorization rules is an advanced configuration step that requires understanding of MQTT topic patterns and access control. For more information about creating authorization rules using EMQX's built-in database, see Use Built-in Database in the *EMQX Docs*.

> **Note**
>
> The EMQX-related instructions provided are for reference only. As EMQX documentation and features may change over time, and we do not maintain their documentation, we recommend consulting EMQX's official documentation for the most current information.

EMQX dashboard

This procedure shows how you can add authorization rules on the EMQX dashboard.

The EMQX dashboard is only accessible from within the gateway host. If you try to connect from outside of the gateway host, you can't access the dashboard.

To add authorization rules using the EMQX Dashboard

1. Ensure that you are within the gateway host.
2. Open a browser window and visit http://localhost:18083/.
3. Login to the the EMQX dashboard. This procedure assumes that you've changed your default login credentials to something of your choosing. For more information on intial setup, see Enable username and password authentication.
4. Choose the shield icon, then **Authorization** from the dropdown menu.
5. Choose the **Permissions** button on the **Built-in Database** row.
6. In the Built-in Database authorization section, add or update the user authorization rules for your business needs. For more guidance on creating rules, see the Use Built-in Database section in the *EMQX Docs*.

AWS IoT SiteWise CLI tool using Linux

To manage authorization rules using the AWS IoT SiteWise EMQX CLI tool in Linux:

- Add authorization rules for a user using the following format:

  ```
  /greengrass/v2/bin/swe-emqx-cli auth add your-username your-action your-
  permission your-topic [your-action-permission-topic]
  ```

Example Add authorization rules for a user

This example shows how to add rules for a user named `system1`:

```
/greengrass/v2/bin/swe-emqx-cli auth add system1 \
    publish allow "sensors/#" \
    subscribe allow "control/#" \
    all deny "#"
```

Example : View authorization rules for a user

To view authorization rules for the `system1` users, run the following command:

```
/greengrass/v2/bin/swe-emqx-cli auth list system1
```

Example : View all existing authorization rules

To view all of the authorization rules you currently have, run the following command:

```
/greengrass/v2/bin/swe-emqx-cli auth list
```

Example : Delete all authorization rules for a user

To delete all of the authorization rules applied to a particular user, run the following command:

```
/greengrass/v2/bin/swe-emqx-cli auth delete system1
```

You are prompted to confirm the deletion.

AWS IoT SiteWise CLI tool using Windows

To manage authorization rules using the AWS IoT SiteWise EMQX CLI tool in Windows PowerShell:

- Add authorization rules for a user using the following format:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 auth add your-username your-action your-permission your-topic [your-action-permission-topic]
```

Example : Add authorization rules for a user

This example shows how to add rules for a user named `system1`:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 auth add system1 `
    publish allow "sensors/#" `
    subscribe allow "control/#" `
    all deny "#"
```

Example : View authorization rules for a user

To view authorization rules for the `system1` users, run the following command:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 auth list system1
```

Example : View all existing authorization rules

To view all of the authorization rules you currently have, run the following command:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 auth list
```

Example : Delete all authorization rules for a user

To delete all of the authorization rules applied to a particular user, run the following command:

```
C:\greengrass\v2\bin\swe-emqx-cli.ps1 auth delete system1
```

You are prompted to confirm the deletion.

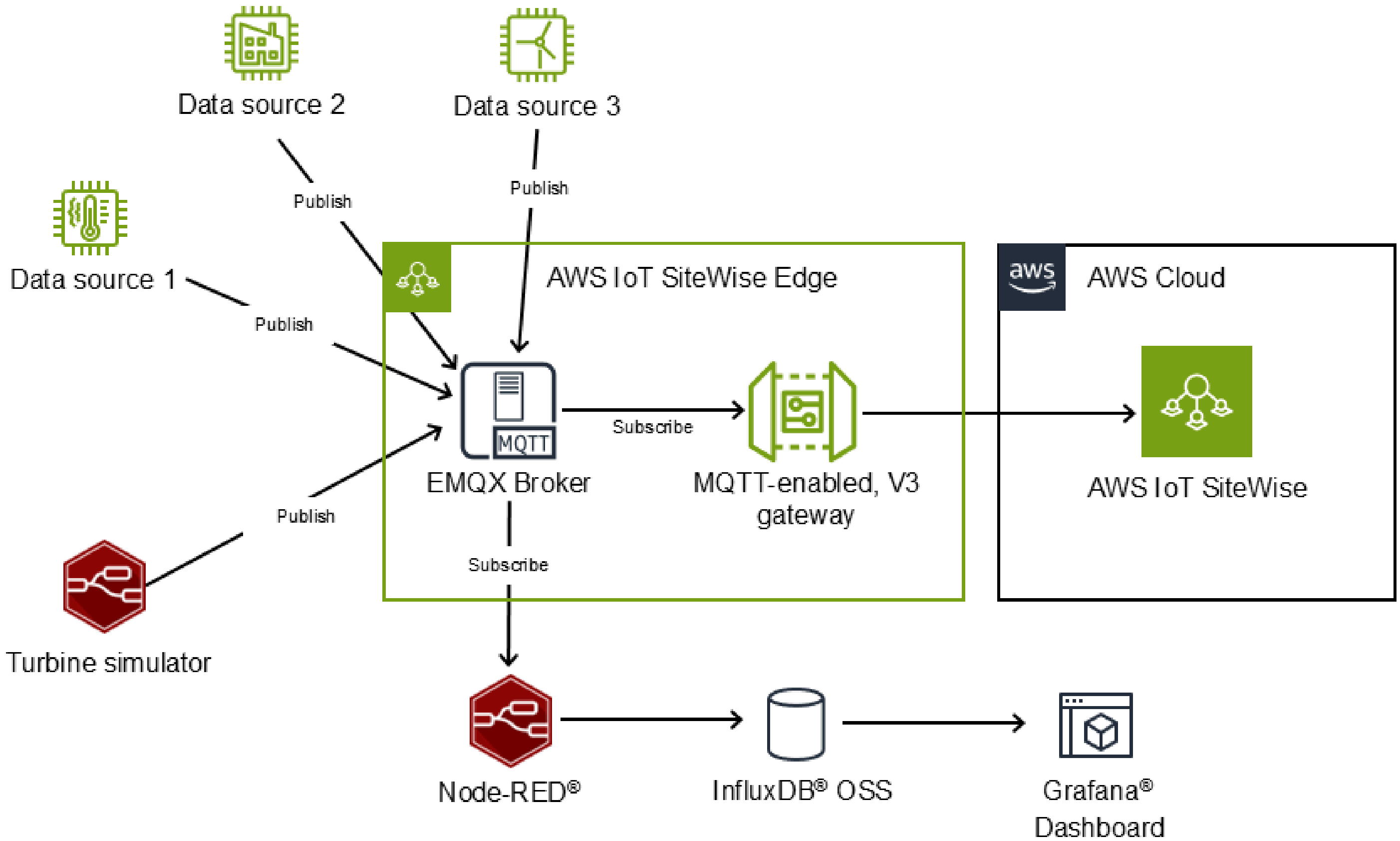
Process and visualize data with SiteWise Edge and open-source tools

Configure AWS IoT SiteWise Edge MQTT-enabled gateways with open-source tools for local processing and visualization to enhance your industrial data management capabilities.

With SiteWise Edge, you can create a local data processing pipeline using external, open-source tools. Use Node-RED® to store time-series data with InfluxDB®, and monitor operations through Grafana® dashboards.

Node-RED processes and transforms your data flows, while InfluxDB provides time-series data storage. Grafana displays your real-time operational data. Use these tools with SiteWise Edge to synchronize data between your local environment and the AWS Cloud, giving you both immediate local insights and long-term cloud-based analytics capabilities.

> **Note**
> Node-RED®, InfluxDB®, and Grafana® are not vendors or suppliers for SiteWise Edge.



> **Note**
>
> In this guide, we're using the open-source version of Grafana for SiteWise Edge as opposed to the Amazon Managed Grafana service.

Deployment options

You can deploy this solution using one of two approaches. With a Microsoft Windows manual setup, you control component configuration and integration with your infrastructure. With Linux, you can use Docker to deploy pre-configured components in containers.

Choose the method that meets your operational requirements.

- Set up open source integrations manually (Windows) – For custom configurations or existing infrastructure
- Set up open-source integrations with Docker (Linux) – For rapid deployment with pre-configured components

Wind farm example overview

This guide uses a wind farm example to demonstrate how you can monitor wind speed for a turbine on a wind farm. This practical scenario illustrates common industrial monitoring needs where both local and cloud-based visibility are valuable for operational efficiency.

With this integration, you can:

- Collect data from industrial equipment using an AWS IoT SiteWise Edge gateway
- Process data locally with Node-RED, InfluxDB, and Grafana
- Store data locally using InfluxDB
- Monitor data in real time using Grafana dashboards

Throughout this guide, we use the example of a windfarm. We use Node-RED to simulate a turbine that generates wind speed data. Node-RED translates the data payload, publishes the data to the SiteWise Edge MQTT broker, subscribes to receive data from the broker, and stores the data locally in InfluxDB. This approach ensures that all of the operational data is available both locally for immediate access and in the cloud for further analytics. By implementing this pattern, you gain resilience against network disruptions while maintaining the ability to perform advanced analytics in the AWS Cloud. Grafana connects to InfluxDB for local monitoring, providing operators with real-time visibility into metrics without cloud dependencies. A SiteWise Edge MQTT-enabled gateway connects to the same MQTT broker to send data to AWS IoT SiteWise, creating a bridge between your edge operations and cloud-based services.

You can use your own data and configurations to create a similar workflow tailored to your specific industrial requirements, whether you're monitoring manufacturing equipment, utility infrastructure, or other industrial assets.

Requirements for open-source integrations

Before implementing open-source integrations with SiteWise Edge, ensure your environment meets the necessary requirements.

- **Hardware requirements** - Your gateway hardware must meet the requirements for SiteWise Edge gateways. For more information, see AWS IoT SiteWise Edge self-hosted gateway requirements for MQTT-enabled, V3 gateways and Requirements for the AWS IoT SiteWise Edge application.

> **Important**
>
> When deploying additional open-source components, ensure your hardware meets the requirements for InfluxDB, Node-RED, and Grafana.

- Your network configuration must support both local communication between components and cloud connectivity for SiteWise Edge.
- All services must run on the same host.

Security considerations

We recommend that you encrypt all communications between components, especially when accessing interfaces from non-local networks. Implement proper access controls for each component and follow AWS best practices for AWS IoT SiteWise Edge gateway configuration and AWS account security.

Development environment

This guide demonstrates Node-RED, InfluxDB, and Grafana running and accessed locally on a gateway host. For production deployments that require external access, implement security measures including TLS encryption, authentication, and authorization. Follow each application's security best practices.

Third-party software

This solution uses third-party software not maintained by AWS, including InfluxDB, Node-RED, Grafana, and the `node-red-contrib-influxdb` plugin. Before deployment, ensure these components comply with your organization's security requirements, compliance standards, and governance policies.

> **Important**
>
> This guide references and uses third-party software not owned or maintained by AWS. Before implementation, ensure that all components meet your security, compliance, and governance requirements. Keep all software updated with the latest security patches and follow best practices for securing your edge deployment.
> InfluxDB, Node-RED, Grafana are not vendors or suppliers for SiteWise Edge.

Other considerations

Consider these additional factors when implementing open-source integrations with SiteWise Edge.

- Use the latest versions of all services, tools, and components.
- Filter and aggregate data locally before cloud transmission to reduce AWS IoT SiteWise data ingestion costs. Configure appropriate data retention periods in InfluxDB and properly size your gateway hardware. For more information, see AWS IoT SiteWise pricing.
- Implement regular backup procedures for all data.
- Monitor resource usage on your gateway and configure appropriate resource limits for each component. Implement data retention policies in InfluxDB to manage disk usage.

Set up open source integrations manually (Windows)

Use this guide to manually create a time series bucket for wind speed data that connects with Grafana® and Node-RED®.

Manually install and configure Node-RED, InfluxDB®, and Grafana on Microsoft Windows to control your deployment configuration. You can store and manage time series data from your devices using InfluxDB.

Manual setup prerequisites

Before you begin, complete these requirements:

> **Note**
>
> Run all services (SiteWise Edge, InfluxDB, Node-RED, and Grafana) on the same host.

- Install an MQTT-enabled, V3 gateway. For more information, see MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge.
- Install and run these services locally:
  - InfluxDB OSS v2. For installation steps, see Install InfluxDB.
  - Node-RED. For installation steps, see Install Node-RED locally.
  - Grafana. For installation steps, see Install Grafana.

Set up local storage with InfluxDB

With InfluxDB®, you can store time series data from your devices locally. The purpose of local storage capability is to maintain operational visibility during network disruptions and reduce latency for time-critical applications. You can perform analysis and visualization at the edge while still having the option to selectively forward data to the cloud.

In this section, you create a time series bucket for turbine wind speed data and generate an API token for Grafana® and Node-RED® connectivity. The InfluxDB bucket serves as a dedicated storage container for your time series data, similar to a database in traditional systems. The API token enables secure programmatic access to your data.

To set up InfluxDB

1. After completing the prerequisite steps and ensuring all tools are running on the same host, open your web browser and go to http://127.0.0.1:8086.
2. (Optional) Enable TLS encryption for enhanced security. For more information, see Enable TLS encryption in the *InfluxData Documentation*.
3. Create a time series InfluxDB bucket to store data from Node-RED. The bucket will serve as a dedicated container for your wind farm data, allowing you to organize and manage retention policies specific to this dataset. For more information, see Manage buckets in the *InfluxData Documentation*.
4. (Optional) Configure the data retention period for your edge location. Setting appropriate retention periods helps manage storage resources efficiently by automatically removing older data that's no longer needed for local operations.

   For information about data retention, see Data retention in InfluxDB in the *InfluxData Documentation*.
5. Generate an API token for the bucket. This token will enable secure communication between InfluxDB and other components like Node-RED and Grafana. This way, only authorized services can read from or write to your data store. For more information, see Create a token in the *InfluxData Documentation*.

After you complete these steps, you can store time series data in your InfluxDB instance, providing a foundation for local data persistence and analysis in your edge environment.

Configure Node-RED flows for AWS IoT SiteWise data integration

With Node-RED®, you can implement two flows to manage data between your devices and AWS IoT SiteWise. These flows work together to create a comprehensive data management solution that addresses both local and cloud data flow.

- **Data publish flow** – Publishes to the cloud. The data publish flow sends data to AWS IoT SiteWise. This flow simulates a turbine device by generating sensor data, translating it to AWS IoT SiteWise format, and publishing to the SiteWise Edge MQTT broker. This enables you to leverage AWS IoT SiteWise's cloud capabilities for storage, analytics, and integration with other AWS services.

  For more information, see Configure the data publish flow.
- **Data retention flow** – Stores data at the edge. The data retention flow subscribes to the SiteWise Edge MQTT broker to receive data, translate it into InfluxDB® format, and stores it locally for monitoring. This local storage provides immediate access to operational data, reduces latency for time-critical applications, and ensures continuity during network disruptions.

  For more information, see Configure the data retention flow.

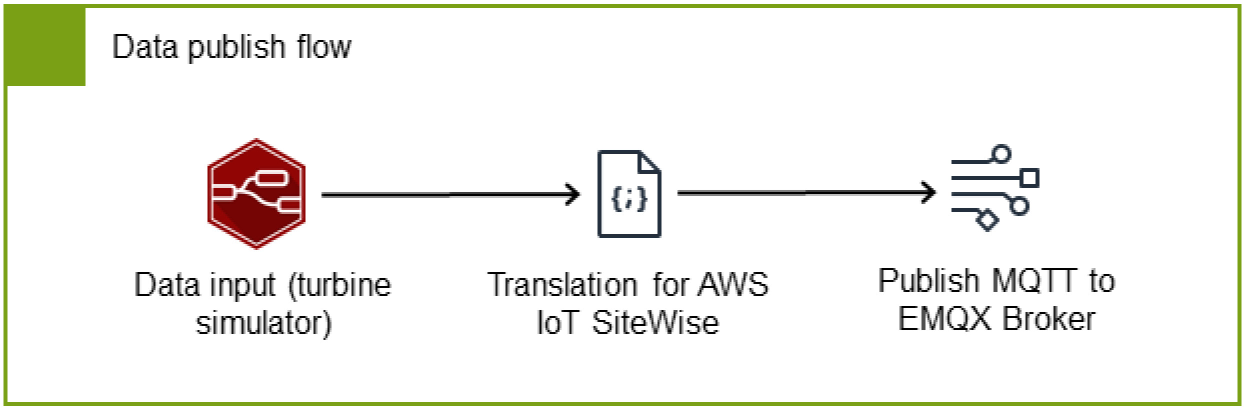
These two flows work together to ensure data is both sent to AWS IoT SiteWise and stored locally for immediate access.

To access your Node-RED console, go to http://127.0.0.1:1880. For information about enabling TLS, see Enable TLS encryption.

Configure the data publish flow

The data publish flow uses three nodes to create a pipeline that sends your industrial data to the cloud. This flow is essential for enabling cloud-based analytics, long-term storage, and integration with other AWS services. First, simulated device data is sent to the SiteWise Edge MQTT broker. The gateway picks up the data from the broker which allows for transmission to the AWS IoT SiteWise cloud, where you can leverage powerful analytics and visualization capabilities.

- **Data input** - Receives device data from your industrial equipment or simulators
- **Data translator for AWS IoT SiteWise** - Translates data to AWS IoT SiteWise format to ensure compatibility with the SiteWise Edge gateway
- **MQTT publisher** - Publishes data to SiteWise Edge MQTT broker, making it available to both local and cloud consumers



Configure the data input node

In this example, the data input node uses a simulated wind turbine device that generates wind speed data. This node serves as the entry point for your industrial data, whether it comes from simulated sources (as in our example) or from actual industrial equipment in production environments.

We use a custom JSON format for the data payload to provide a standardized structure that works efficiently with both local processing tools and the AWS IoT SiteWise cloud service. This format includes essential metadata like timestamps and quality indicators alongside the actual measurement values, enabling comprehensive data management and quality tracking throughout your pipeline. Import the inject node to receive simulated data in this standardized JSON format with timestamps, quality indicators, and values.

For more information on the Node-RED inject node, see the [Inject](#) section in the *Node-RED Documentation*.

The turbine simulator generates wind speed data every second in this standardized JSON format:

Example : Turbine data payload

```
{
    name: string,          // Property name/identifier
    timestamp: number,     // Epoch time in nanoseconds
    quality: "GOOD" | "UNCERTAIN" | "BAD",
    value: number | string | boolean
}
```

This format provides several benefits:

- The name field identifies the specific property or measurement, allowing you to track multiple data points from the same device
- The timestamp in nanoseconds ensures precise time tracking for accurate historical analysis
- The quality indicator helps you filter and manage data based on its reliability
- The flexible value field supports different data types to accommodate various sensor outputs

Example : Inject node of a turbine simulator

```
[
    {
        "id": "string",
        "type": "inject",
        "z": "string",
        "name": "Turbine Simulator",
        "props": [
            {
                "p": "payload.timestamp",
                "v": "",
                "vt": "date"
            },
            {
                "p": "payload.quality",
                "v": "GOOD",
                "vt": "str"
            },
            {
                "p": "payload.value",
                "v": "$random()",
                "vt": "jsonata"
            },
            {
                "p": "payload.name",
                "v": "/Renton/WindFarm/Turbine/WindSpeed",
                "vt": "str"
            }
        ],
        "repeat": "1",
        "crontab": "",
        "once": false,
        "onceDelay": "",
        "topic": "",
```

```
        "x": 270,
        "y": 200,
        "wires": [
            [
                "string"
            ]
        ]
    }
]
```

Configure a node for data translation

The SiteWise Edge gateway requires data in a specific format to ensure compatibility with AWS IoT SiteWise cloud. The translator node is an important component that converts your input data to the required AWS IoT SiteWise payload format. This translation step ensures that your industrial data can be properly processed, stored, and later analyzed in the AWS IoT SiteWise cloud environment.

By standardizing the data format at this stage, you enable integration between your edge devices and the cloud service where you can use asset modeling, analytics, and visualization capabilities. Use this structure:

Example : Payload structure for SiteWise Edge data parsing

```
{
  "propertyAlias": "string",
  "propertyValues": [
    {
      "value": {
          "booleanValue": boolean,
          "doubleValue": number,
          "integerValue": number,
          "stringValue": "string"
     },
      "timestamp": {
          "timeInSeconds": number,
          "offsetInNanos": number
      },
      "quality": "GOOD" | "UNCERTAIN" | "BAD",
  }]
}
```

> **Note**
>
> Match the `propertyAlias` to your MQTT topic hierarchy (for example, `/Renton/WindFarm/Turbine/WindSpeed`). This ensures that your data is properly associated with the correct asset property in AWS IoT SiteWise. For more information, see the "Data stream alias" concept in AWS IoT SiteWise concepts.

1. Import the example function node for AWS IoT SiteWise payload translation. This function handles the conversion from your standardized input format to the AWS IoT SiteWise-compatible format, ensuring proper timestamp formatting, quality indicators, and value typing.

```
[
    {
        "id": "string",
        "type": "function",
        "z": "string",
        "name": "Translate to SiteWise payload",
        "func": "let input = msg.payload;\nlet output = {};\n
\noutput[\"propertyAlias\"] = input.name;\n\nlet propertyVal = {}\n\nlet
 timeInSeconds = Math.floor(input.timestamp / 1000);\nlet offsetInNanos =
 (input.timestamp % 1000) * 1000000;\n\npropertyVal[\"timestamp\"] = {\n
 \"timeInSeconds\": timeInSeconds,\n    \"offsetInNanos\": offsetInNanos,\n};
\n\npropertyVal[\"quality\"] = input.quality\n\nlet typeNameConverter = {\n
   \"number\": (x) => Number.isInteger(x) ? \"integerValue\" : \"doubleValue
\",\n    \"boolean\": (x) => \"booleanValue\",\n    \"string\": (x) =>
 \"stringValue\", \n}\nlet typeName = typeNameConverter[typeof input.value]
(input.value)\npropertyVal[\"value\"] = {}\npropertyVal[\"value\"][typeName]
 = input.value;\n\noutput[\"propertyValues\"] = [propertyVal]\n\nreturn {\n
 payload: JSON.stringify(output)\n};",
        "outputs": 1,
        "timeout": "",
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 530,
        "y": 200,
        "wires": [
            [
```

```
                "string"
            ]
        ]
    }
]
```

2. Verify that the JavaScript code translates wind speed data correctly. The function performs several important tasks:

   - Extracts the property name from the input and sets it as the propertyAlias
   - Converts the timestamp from milliseconds to the required seconds and nanoseconds format
   - Preserves the quality indicator for data reliability tracking
   - Automatically detects the value type and formats it according to AWS IoT SiteWise requirements

3. Connect the node to your flow, linking it between the data input node and the MQTT publisher.

For guidance on writing a function specific to your business needs, see Writing Functions in the *Node-RED Documentation*

Configure the MQTT publisher

After translation, the data is ready for publication to the SiteWise Edge MQTT broker.

Configure the MQTT publisher with these settings to send data to the SiteWise Edge MQTT broker:

To import the MQTT out node

1. Import an MQTT out configuration node using `"type": "mqtt out"`. MQTT out nodes let you share a broker's configuration.
2. Enter key-value pairs for information relevant to MQTT broker connection and message routing.

Import the example `mqtt out` node.

Example

```
[
    {
```

```
        "id": "string",
        "type": "mqtt out",
        "z": "string",
        "name": "Publish to MQTT broker",
        "topic": "/Renton/WindFarm/Turbine/WindSpeed",
        "qos": "1",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correl": "",
        "expiry": "",
        "broker": "string",
        "x": 830,
        "y": 200,
        "wires": []
    },
    {
        "id": "string",
        "type": "mqtt-broker",
        "name": "emqx",
        "broker": "127.0.0.1",
        "port": "1883",
        "clientid": "",
        "autoConnect": true,
        "usetls": false,
        "protocolVersion": "5",
        "keepalive": 15,
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "birthMsg": {},
        "closeTopic": "",
        "closePayload": "",
        "closeMsg": {},
        "willTopic": "",
        "willQos": "0",
        "willPayload": "",
        "willMsg": {},
        "userProps": "",
        "sessionExpiry": ""
    }
```

```
]
```

The example MQTT out node creates the MQTT connection with the following information:

- Server: `127.0.0.1`
- Port: `1883`
- Protocol: `MQTT V5`

Then, the MQTT out node configures message routing with the following information:

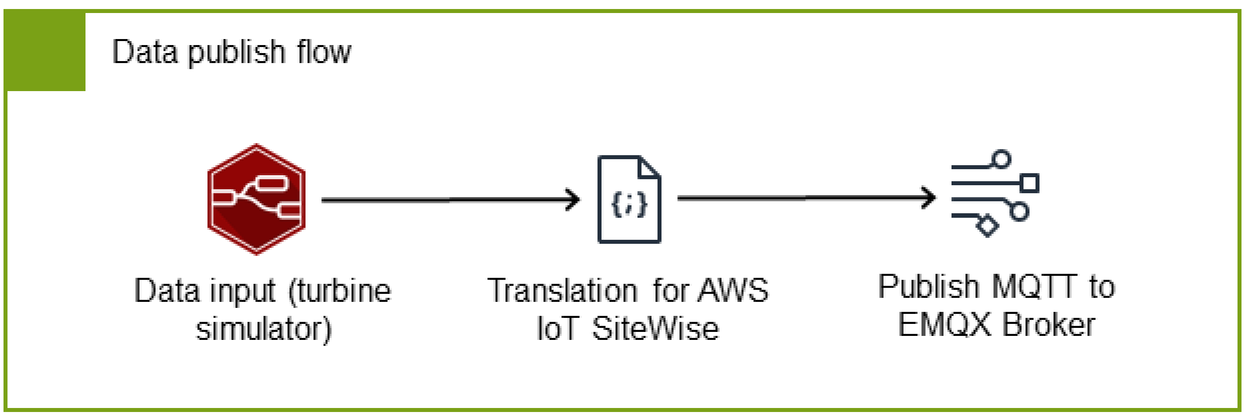
- Topic: `/Renton/WindFarm/Turbine/WindSpeed`
- QoS: `1`

Deploy and verify the nodes

After configuring the three data publish flow nodes, follow these steps to deploy the flow and verify that data is being transmitted correctly to AWS IoT SiteWise

To deploy and verify connections

1. Connect the three nodes as shown in the data publish flow.



2. Choose **Deploy** to apply all node connection changes.
3. Navigate to the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/) and choose **Data streams**.
4. Ensure **Alias prefix** is selected in the dropdown menu. Then, search for the `/Renton/WindFarm/Turbine/WindSpeed` alias.

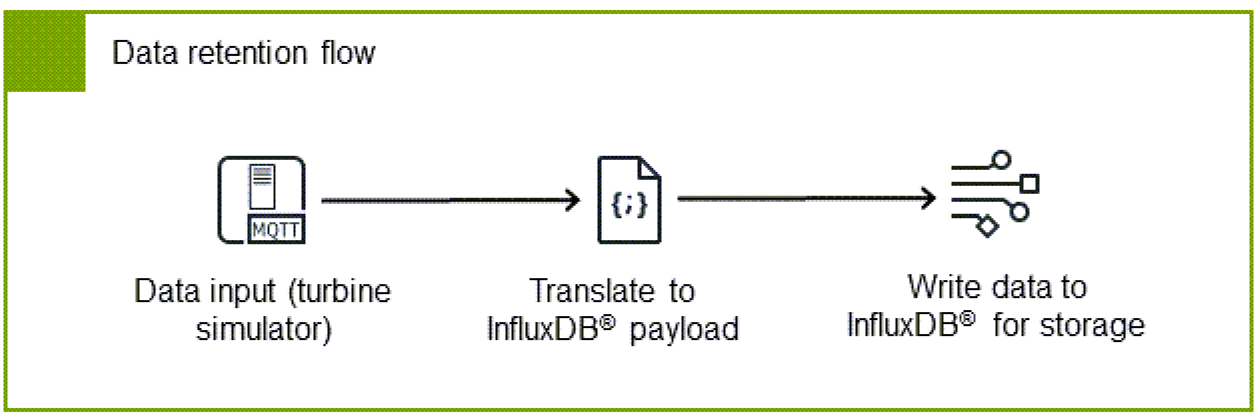
If you see the correct alias in your search, you have deployed the flow and verified data transmission.

Configure the data retention flow

The data retention flow is can be used to maintain operational visibility at the edge. This is useful during network disruptions or when you need immediate access to your data. This flow subscribes to the MQTT broker to receive device data, converts it to InfluxDB® format, and stores it locally. By implementing this flow, you create a resilient local data store that operators can access without cloud dependencies, enabling real-time monitoring and decision-making at the edge.

The flow consists of three key components working together to ensure your data is properly captured and stored:

- **MQTT subscription client** - Receives data from the broker, ensuring you capture all relevant industrial data
- **InfluxDB translator** - Converts AWS IoT SiteWise payload to InfluxDB format, preparing the data for efficient time-series storage
- **InfluxDB writer** - Handles local storage, ensuring data persistence and availability for local applications



Set up the MQTT subscription client

- Configure the MQTT subscription client in Node-RED to receive data from the MQTT EMQX broker in AWS IoT SiteWise by importing the example below.

Example : MQTT in node

```
[
    {
        "id": "string",
        "type": "mqtt in",
        "z": "string",
        "name": "Subscribe to MQTT broker",
        "topic": "/Renton/WindFarm/Turbine/WindSpeed",
        "qos": "1",
        "datatype": "auto-detect",
        "broker": "string",
        "nl": false,
        "rap": true,
        "rh": 0,
        "inputs": 0,
        "x": 290,
        "y": 340,
        "wires": [
            [
                "string"
            ]
        ]
    },
    {
        "id": "string",
        "type": "mqtt-broker",
        "name": "emqx",
        "broker": "127.0.0.1",
        "port": "1883",
        "clientid": "",
        "autoConnect": true,
        "usetls": false,
        "protocolVersion": "5",
        "keepalive": 15,
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "birthMsg": {},
        "closeTopic": "",
        "closePayload": "",
```

```
        "closeMsg": {},
        "willTopic": "",
        "willQos": "0",
        "willPayload": "",
        "willMsg": {},
        "userProps": "",
        "sessionExpiry": ""
    }
]
```

This subscription ensures that all relevant data published to the broker is captured for local storage, providing a complete record of your industrial operations. The node uses the same MQTT connection parameters as the Configure the MQTT publisher section, with the following subscription settings:

- Topic – `/Renton/WindFarm/Turbine/WindSpeed`
- QoS – 1

For more information, see Connect to an MQTT Broker in the *Node-RED Documentation*.

Configure the InfluxDB translator

InfluxDB organizes data using tags for indexing and fields for values. This organization optimizes query performance and storage efficiency for time-series data. Import the example function node that contains JavaScript code to convert AWS IoT SiteWise payload to InfluxDB format. The translator splits the properties into two groups:

- Tags – Quality and name properties for efficient indexing
- Fields – Timestamp (in milliseconds since epoch) and value

Example : Function node of translating to an InfluxDB payload

```
[
    {
        "id": "string",
        "type": "function",
        "z": "string",
        "name": "Translate to InfluxDB payload",
```

```
        "func": "let data = msg.payload;\n\nlet timeInSeconds =
 data.propertyValues[0].timestamp.timeInSeconds;\nlet offsetInNanos =
 data.propertyValues[0].timestamp.offsetInNanos;\nlet timestampInMilliseconds =
 (timeInSeconds * 1000) + (offsetInNanos / 1000000);\n\nmsg.payload = [\n    {\n
      \"timestamp(milliseconds_since_epoch)\": timestampInMilliseconds,\n
 \"value\": data.propertyValues[0].value.doubleValue\n    },\n    {\n        \"name\":
 data.propertyAlias,\n        \"quality\": data.propertyValues[0].quality\n    }\n]\n
\nreturn msg",
        "outputs": 1,
        "timeout": "",
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 560,
        "y": 340,
        "wires": [
            [
                "string"
            ]
        ]
    }
]
```


For additional configuration options, see the node-red-contrib-influxdb in the Node-RED GitHub repository.

Set up the InfluxDB writer

The InfluxDB writer node is the final component in your data retention flow, responsible for storing your industrial data in the local InfluxDB database. This local storage is important for maintaining operational visibility during network disruptions and providing immediate access to data for time-critical applications.

1. Install the node-red-contrib-influxdb package through the Manage palette option. This package provides the necessary nodes for connecting Node-RED with InfluxDB.
2. Add an InfluxDB out node to your flow. This node will handle the actual writing of data to your InfluxDB database.
3. Configure the server properties to establish a secure connection to your InfluxDB instance:

    a. Set Version to 2.0 - This specifies that you're connecting to InfluxDB v2.x, which uses a different API than earlier versions

   b. Set URL to `http://127.0.0.1:8086` - This points to your local InfluxDB instance
   c. Enter your InfluxDB authentication token. This secure token authorizes the connection to your database. You generated the token during the Set up local storage with InfluxDB procedure.

4. Specify the storage location parameters to define where and how your data will be stored:

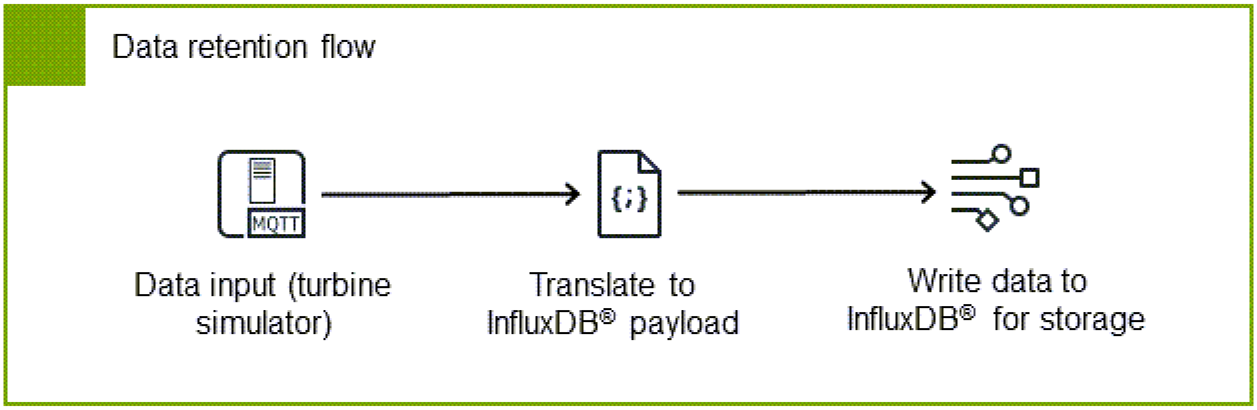
   a. Enter your InfluxDB Organization name – The organization is a workspace for a group of users, where your buckets and dashboards belong. For more information, see Manage organizations in the *InfluxData Documentation*.
   b. Specify the InfluxDB Bucket (for example, `WindFarmData`) – The bucket is equivalent to a database in traditional systems, serving as a container for your time series data
   c. Set the InfluxDB Measurement (for example, `TurbineData`) – The measurement is similar to a table in relational databases, organizing related data points

> **Note**
>
> Find your organization name in the InfluxDB instance's left sidebar. The organization, bucket, and measurement concepts are fundamental to InfluxDB's data organization model. For more information, see the InfluxDB documentation.

Deploy and verify the retention flow

After configuring all components of the data retention flow, you need to deploy and verify that the system is working correctly. This verification ensures that your industrial data is being properly stored locally for immediate access and analysis.

1. Connect the three nodes as shown in the data retention flow diagram. This creates a complete pipeline from data subscription to local storage.



2. Choose **Deploy** to apply your changes and activate the flow. This starts the data collection and storage process.
3. Use the InfluxDB Data Explorer to query and visualize your data. This tool allows you to verify that data is being properly stored and to create initial visualizations of your time series data.

   In the Data Explorer, you should be able to see your wind speed measurements being recorded over time, confirming that the entire pipeline from data generation to local storage is functioning correctly.

   For more information, see [Query in Data Explorer](#) in the *InfluxData Documentation*.

With both the data publish flow and data retention flow deployed, you now have a complete system that sends data to the AWS IoT SiteWise cloud while maintaining a local copy for immediate access and resilience. This dual-path approach ensures that you get the benefits of cloud-based analytics and storage while maintaining operational visibility at the edge.

Set up Grafana for SiteWise Edge

Grafana® lets you create local real-time monitoring dashboards for your industrial data. By visualizing the data stored in InfluxDB®, you can provide operators with immediate insights into equipment performance, process efficiency, and potential issues. This visibility at the edge is important for time-sensitive operations and maintaining continuity during network disruptions.

Configure the data source

Connecting Grafana to your InfluxDB database creates a powerful visualization layer for your industrial data. This connection enables real-time monitoring dashboards that operators can use to make informed decisions without cloud dependencies.

1. Access your Grafana instance locally by navigating to http://127.0.0.1:3000 in your browser. If enabling TLS is required, you can refer to Set up Grafana HTTPS for secure web traffic in the *Grafana Labs Documentation*.
2. Add an InfluxDB data source pointing to the InfluxDB time series bucket where Node-RED writes data. For example, `WindFarmData`. This connection establishes the link between your stored data and the visualization platform.
3. For detailed instructions, see Configure the InfluxDB data source in the *Grafana Labs Documentation*.

Create a Grafana dashboard for SiteWise Edge data

Creating a dashboard is the final step in building your local monitoring solution. Dashboards provide visual representations of your industrial data, making it easier to identify trends, anomalies, and potential issues at a glance.

- Follow the guide to create a dashboard. For more information, see Build your first dashboard in the *Grafana Labs Documentation*. This template assumes your bucket is named `WindFarmData` and measurement is `TurbineData`.

  You can also use the quick start guide by importing the provided example dashboard template to quickly create a dashboard with a time series plot for the data that Node-RED generates in previous section. This template provides a starting point that you can customize to meet your specific monitoring needs.

```
{
  "__inputs": [
    {
      "name": "DS_WINDFARM-DEMO",
      "label": "windfarm-demo",
      "description": "",
      "type": "datasource",
      "pluginId": "influxdb",
      "pluginName": "InfluxDB"
    }
  ],
  "__elements": {},
  "__requires": [
    {
      "type": "grafana",
```

```
      "id": "grafana",
      "name": "Grafana",
      "version": "11.6.0-pre"
    },
    {
      "type": "datasource",
      "id": "influxdb",
      "name": "InfluxDB",
      "version": "1.0.0"
    },
    {
      "type": "panel",
      "id": "timeseries",
      "name": "Time series",
      "version": ""
    }
  ],
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": {
          "type": "grafana",
          "uid": "-- Grafana --"
        },
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "fiscalYearStartMonth": 0,
  "graphTooltip": 0,
  "id": null,
  "links": [],
  "panels": [
    {
      "datasource": {
        "type": "influxdb",
        "uid": "${DS_WINDFARM-DEMO}"
      },
```

```
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "palette-classic"
          },
          "custom": {
            "axisBorderShow": false,
            "axisCenteredZero": false,
            "axisColorMode": "text",
            "axisLabel": "",
            "axisPlacement": "auto",
            "barAlignment": 0,
            "barWidthFactor": 0.6,
            "drawStyle": "line",
            "fillOpacity": 0,
            "gradientMode": "none",
            "hideFrom": {
              "legend": false,
              "tooltip": false,
              "viz": false
            },
            "insertNulls": false,
            "lineInterpolation": "linear",
            "lineWidth": 1,
            "pointSize": 5,
            "scaleDistribution": {
              "type": "linear"
            },
            "showPoints": "auto",
            "spanNulls": false,
            "stacking": {
              "group": "A",
              "mode": "none"
            },
            "thresholdsStyle": {
              "mode": "off"
            }
          },
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green"
```

```
              },
              {
                "color": "red",
                "value": 80
              }
            ]
          }
        },
        "overrides": []
      },
      "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "id": 1,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "bottom",
          "showLegend": true
        },
        "tooltip": {
          "hideZeros": false,
          "mode": "single",
          "sort": "none"
        }
      },
      "pluginVersion": "11.6.0-pre",
      "targets": [
        {
          "datasource": {
            "type": "influxdb",
            "uid": "${DS_WINDFARM-DEMO}"
          },
          "query": "from(bucket: \"WindFarmData\")\n  |> range(start:
 v.timeRangeStart, stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement
\"] == \"TurbineData\")\n  |> filter(fn: (r) => r[\"_field\"] == \"value\")\n
 |> filter(fn: (r) => r[\"name\"] == \"/Renton/WindFarm/Turbine/WindSpeed\")\n
  |> filter(fn: (r) => r[\"quality\"] == \"GOOD\")\n  |> aggregateWindow(every:
 v.windowPeriod, fn: mean, createEmpty: false)\n  |> yield(name: \"mean\")",
          "refId": "A"
```

```
        }
      ],
      "title": "Panel Title",
      "type": "timeseries"
    }
  ],
  "schemaVersion": 41,
  "tags": [],
  "templating": {
    "list": []
  },
  "time": {
    "from": "now-6h",
    "to": "now"
  },
  "timepicker": {},
  "timezone": "browser",
  "title": "demo dashboard",
  "uid": "fejc0t08o6d4wb",
  "version": 1,
  "weekStart": ""
}
```

Set up open-source integrations with Docker (Linux)

For a streamlined deployment process, you can use Docker to set up Node-RED®, InfluxDB®, and Grafana® on a Linux environment. This method uses pre-configured containers, allowing for rapid deployment and easier management of the components.

Docker setup prerequisites

Before you begin, verify that have the following:

- An MQTT-enabled, V3 gateway. For more information, see MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge.
- The Docker Compose plugin. For installation steps, see Install the Docker Compose plugin in the *Docker* Manuals documentation.

Deploy the services

This deployment runs SiteWise Edge, InfluxDB, Node-RED, and Grafana on the same host.

Set up the environment

1. Gain root access:

```
sudo -i
```

2. Create a .env file or export these environment variables:

```
export INFLUXDB_PASSWORD=your-secure-influxdb-password
export INFLUXDB_TOKEN=your-secure-influxdb-token
export GRAFANA_PASSWORD=your-secure-grafana-password
```

Configure the Docker network

- Create a bridge network using the name `SiteWiseEdgeNodeRedDemoNetwork`.

```
docker network create --driver=bridge SiteWiseEdgeNodeRedDemoNetwork
```

Prepare the Docker Compose file

Copy the contents of the following YAML file to your SiteWise Edge gateway device.

Expand to view the Docker Compose YAML file example

```
services:
  influxdb:
    image: influxdb:latest
    container_name: influxdb
    ports:
      - "127.0.0.1:8086:8086"
    volumes:
      - influxdb-storage:/.influxdbv2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=${INFLUXDB_PASSWORD}
      - DOCKER_INFLUXDB_INIT_ORG=iot-sitewise-edge
      - DOCKER_INFLUXDB_INIT_BUCKET=WindFarmData
      - DOCKER_INFLUXDB_INIT_RETENTION=0
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=${INFLUXDB_TOKEN}
    networks:
```

```
      - SiteWiseEdgeNodeRedDemoNetwork
    restart: unless-stopped

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "127.0.0.1:3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD}
      - GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-simple-json-datasource
      - GF_PATHS_PROVISIONING=/etc/grafana/provisioning
      - GF_PATHS_CONFIG=/etc/grafana/grafana.ini
      - GF_LOG_LEVEL=info
    configs:
      - source: grafana_datasource
        target: /etc/grafana/provisioning/datasources/influxdb.yaml
      - source: grafana_preload_dashboard_config
        target: /etc/grafana/provisioning/dashboards/dashboard.yml
      - source: grafana_preload_dashboard
        target: /etc/grafana/provisioning/dashboards/demo_dashboard.json
    depends_on:
      - influxdb
    networks:
      - SiteWiseEdgeNodeRedDemoNetwork
    restart: unless-stopped

  nodered:
    build:
      context: .
      dockerfile_inline: |
        FROM nodered/node-red:latest
        RUN npm install node-red-contrib-influxdb
    container_name: nodered
    ports:
      - "127.0.0.1:1880:1880"
    volumes:
      - node_red_data:/data
    environment:
      - NODE_RED_ENABLE_SAFE_MODE=false
```

```
      - NODE_RED_ENABLE_PALETTE_EDIT=true
      - NODE_RED_AUTO_INSTALL_MODULES=true
    configs:
      - source: nodered_flows
        target: /data/flows.json
      - source: nodered_settings
        target: /data/settings.js
      - source: nodered_flows_cred
        target: /data/flows_cred.json
    depends_on:
      - influxdb
    networks:
      - SiteWiseEdgeNodeRedDemoNetwork
    restart: unless-stopped

volumes:
  influxdb-storage:
  grafana-storage:
  node_red_data:

networks:
  SiteWiseEdgeNodeRedDemoNetwork:
    external: true

configs:
  grafana_datasource:
    content: |
      apiVersion: 1
      datasources:
        - name: windfarm-demo
          type: influxdb
          access: proxy
          url: http://influxdb:8086
          jsonData:
            version: Flux
            organization: iot-sitewise-edge
            defaultBucket: WindFarmData
            tlsSkipVerify: true
          secureJsonData:
            token: ${INFLUXDB_TOKEN}
          editable: false

  grafana_preload_dashboard_config:
    content: |
```

```
    apiVersion: 1
    providers:
      - name: "Dashboard provider"
        orgId: 1
        type: file
        options:
          path: /etc/grafana/provisioning/dashboards

grafana_preload_dashboard:
  content: |
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": {
              "type": "grafana",
              "uid": "-- Grafana --"
            },
            "enable": true,
            "hide": true,
            "iconColor": "rgba(0, 211, 255, 1)",
            "name": "Annotations & Alerts",
            "type": "dashboard"
          }
        ]
      },
      "editable": true,
      "fiscalYearStartMonth": 0,
      "graphTooltip": 0,
      "id": 1,
      "links": [],
      "panels": [
        {
          "datasource": {
            "type": "influxdb",
            "uid": "PEB0DCBF338B3CEB2"
          },
          "fieldConfig": {
            "defaults": {
              "color": {
                "mode": "palette-classic"
              },
              "custom": {
```

```
                    "axisBorderShow": false,
                    "axisCenteredZero": false,
                    "axisColorMode": "text",
                    "axisLabel": "",
                    "axisPlacement": "auto",
                    "barAlignment": 0,
                    "barWidthFactor": 0.6,
                    "drawStyle": "line",
                    "fillOpacity": 0,
                    "gradientMode": "none",
                    "hideFrom": {
                      "legend": false,
                      "tooltip": false,
                      "viz": false
                    },
                    "insertNulls": false,
                    "lineInterpolation": "linear",
                    "lineWidth": 1,
                    "pointSize": 5,
                    "scaleDistribution": {
                      "type": "linear"
                    },
                    "showPoints": "auto",
                    "spanNulls": false,
                    "stacking": {
                      "group": "A",
                      "mode": "none"
                    },
                    "thresholdsStyle": {
                      "mode": "off"
                    }
                  },
                  "mappings": [],
                  "thresholds": {
                    "mode": "absolute",
                    "steps": [
                      {
                        "color": "green"
                      },
                      {
                        "color": "red",
                        "value": 80
                      }
                    ]
```

```
          }
        },
        "overrides": []
      },
      "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "id": 1,
      "options": {
        "legend": {
          "calcs": [],
          "displayMode": "list",
          "placement": "bottom",
          "showLegend": true
        },
        "tooltip": {
          "hideZeros": false,
          "mode": "single",
          "sort": "none"
        }
      },
      "pluginVersion": "11.6.0",
      "targets": [
        {
          "datasource": {
            "type": "influxdb",
            "uid": "PEB0DCBF338B3CEB2"
          },
          "query": "from(bucket: \"WindFarmData\")\n  |> range(start:
 v.timeRangeStart, stop: v.timeRangeStop)\n  |> filter(fn: (r) => r[\"_measurement
\"] == \"TurbineData\")\n  |> filter(fn: (r) => r[\"_field\"] == \"value\")\n
 |> filter(fn: (r) => r[\"name\"] == \"/Renton/WindFarm/Turbine/WindSpeed\")\n
  |> filter(fn: (r) => r[\"quality\"] == \"GOOD\")\n  |> aggregateWindow(every:
 v.windowPeriod, fn: mean, createEmpty: false)\n  |> yield(name: \"mean\")",
          "refId": "A"
        }
      ],
      "title": "Wind Speed",
      "type": "timeseries"
    }
  ],
```

```
        "preload": false,
        "schemaVersion": 41,
        "tags": [],
        "templating": {
          "list": []
        },
        "time": {
          "from": "now-6h",
          "to": "now"
        },
        "timepicker": {},
        "timezone": "browser",
        "title": "Demo Dashboard",
        "uid": "eejtureqjo9a8c",
        "version": 2
      }

  nodered_flows:
    content: |
      [
        {
          "id": "95fce448fdd43b47",
          "type": "tab",
          "label": "Demo Flow",
          "disabled": false,
          "info": ""
        },
        {
          "id": "5f63740b66af3386",
          "type": "mqtt out",
          "z": "95fce448fdd43b47",
          "name": "Publish to MQTT broker",
          "topic": "/Renton/WindFarm/Turbine/WindSpeed",
          "qos": "1",
          "retain": "",
          "respTopic": "",
          "contentType": "",
          "userProps": "",
          "correl": "",
          "expiry": "",
          "broker": "5744207557fa19be",
          "x": 830,
          "y": 200,
          "wires": []
```

```
        },
        {
          "id": "8f2eb590d596679b",
          "type": "function",
          "z": "95fce448fdd43b47",
          "name": "Translate to SiteWise payload",
          "func": "let input = msg.payload;\nlet output = {};\n\noutput[\"propertyAlias
\"] = input.name;\n\nlet propertyVal = {}\n\nlet timeInSeconds =
 Math.floor(input.timestamp / 1000);\nlet offsetInNanos = (input.timestamp % 1000) *
 1000000;\n\npropertyVal[\"timestamp\"] = {\n    \"timeInSeconds\": timeInSeconds,\n
    \"offsetInNanos\": offsetInNanos,\n};\n\npropertyVal[\"quality\"] = input.quality
\n\nlet typeNameConverter = {\n    \"number\": (x) => Number.isInteger(x) ?
 \"integerValue\" : \"doubleValue\",\n    \"boolean\": (x) => \"booleanValue\",\n
   \"string\": (x) => \"stringValue\", \n}\nlet typeName = typeNameConverter[typeof
 input.value](input.value)\npropertyVal[\"value\"] = {}\npropertyVal[\"value\"]
[typeName] = input.value;\n\noutput[\"propertyValues\"] = [propertyVal]\n\nreturn {\n
  payload: JSON.stringify(output)\n};",
          "outputs": 1,
          "timeout": "",
          "noerr": 0,
          "initialize": "",
          "finalize": "",
          "libs": [],
          "x": 530,
          "y": 200,
          "wires": [
            [
              "5f63740b66af3386"
            ]
          ]
        },
        {
          "id": "4b78cbdea5e3258c",
          "type": "inject",
          "z": "95fce448fdd43b47",
          "name": "Turbine Simulator",
          "props": [
            {
              "p": "payload.timestamp",
              "v": "",
              "vt": "date"
            },
            {
              "p": "payload.quality",
```

```
          "v": "GOOD",
          "vt": "str"
        },
        {
          "p": "payload.value",
          "v": "$$random()",
          "vt": "jsonata"
        },
        {
          "p": "payload.name",
          "v": "/Renton/WindFarm/Turbine/WindSpeed",
          "vt": "str"
        }
      ],
      "repeat": "1",
      "crontab": "",
      "once": false,
      "onceDelay": "",
      "topic": "",
      "x": 270,
      "y": 200,
      "wires": [
        [
          "8f2eb590d596679b"
        ]
      ]
    },
    {
      "id": "b658bf337ea2e316",
      "type": "influxdb out",
      "z": "95fce448fdd43b47",
      "influxdb": "2f1c38495035d2e4",
      "name": "Store data in InfluxDB",
      "measurement": "TurbineData",
      "precision": "",
      "retentionPolicy": "",
      "database": "",
      "retentionPolicyV18Flux": "",
      "org": "iot-sitewise-edge",
      "bucket": "WindFarmData",
      "x": 840,
      "y": 340,
      "wires": []
    },
```

```
        {
          "id": "9432d39af35b202f",
          "type": "function",
          "z": "95fce448fdd43b47",
          "name": "Translate to InfluxDB payload",
          "func": "let data = msg.payload;\n\nlet timeInSeconds =
 data.propertyValues[0].timestamp.timeInSeconds;\nlet offsetInNanos =
 data.propertyValues[0].timestamp.offsetInNanos;\nlet timestampInMilliseconds =
 (timeInSeconds * 1000) + (offsetInNanos / 1000000);\n\nmsg.payload = [\n    {\n
       \"timestamp(milliseconds_since_epoch)\": timestampInMilliseconds,\n
 \"value\": data.propertyValues[0].value.doubleValue\n    },\n    {\n        \"name\":
 data.propertyAlias,\n        \"quality\": data.propertyValues[0].quality\n    }\n]\n
\nreturn msg",
          "outputs": 1,
          "timeout": "",
          "noerr": 0,
          "initialize": "",
          "finalize": "",
          "libs": [],
          "x": 560,
          "y": 340,
          "wires": [
            [
              "b658bf337ea2e316"
            ]
          ]
        },
        {
          "id": "b689403d2c80816b",
          "type": "mqtt in",
          "z": "95fce448fdd43b47",
          "name": "Subscribe to MQTT broker",
          "topic": "/Renton/WindFarm/Turbine/WindSpeed",
          "qos": "1",
          "datatype": "auto-detect",
          "broker": "5744207557fa19be",
          "nl": false,
          "rap": true,
          "rh": 0,
          "inputs": 0,
          "x": 290,
          "y": 340,
          "wires": [
            [
```

```
            "9432d39af35b202f"
          ]
        ]
      },
      {
        "id": "4f59bed8e829fc35",
        "type": "comment",
        "z": "95fce448fdd43b47",
        "name": "Data Publish Flow",
        "info": "dfgh",
        "x": 270,
        "y": 160,
        "wires": []
      },
      {
        "id": "b218c7fc58c8b6e7",
        "type": "comment",
        "z": "95fce448fdd43b47",
        "name": "Data Retention flow",
        "info": "",
        "x": 270,
        "y": 300,
        "wires": []
      },
      {
        "id": "5744207557fa19be",
        "type": "mqtt-broker",
        "name": "emqx",
        "broker": "emqx",
        "port": "1883",
        "clientid": "",
        "autoConnect": true,
        "usetls": false,
        "protocolVersion": "5",
        "keepalive": 15,
        "cleansession": true,
        "autoUnsubscribe": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "birthMsg": {},
        "closeTopic": "",
        "closePayload": "",
        "closeMsg": {},
```

```
          "willTopic": "",
          "willQos": "0",
          "willPayload": "",
          "willMsg": {},
          "userProps": "",
          "sessionExpiry": ""
        },
        {
          "id": "2f1c38495035d2e4",
          "type": "influxdb",
          "hostname": "influxdb",
          "port": 8086,
          "protocol": "http",
          "database": "",
          "name": "InfluxDB",
          "usetls": false,
          "tls": "",
          "influxdbVersion": "2.0",
          "url": "http://influxdb:8086",
          "timeout": "",
          "rejectUnauthorized": false
        }
      ]

  nodered_flows_cred:
    content: |
      {
        "2f1c38495035d2e4": {
          "token": "${INFLUXDB_TOKEN}"
        }
      }

  nodered_settings:
    content: |
      module.exports = {
        flowFile: 'flows.json',
        credentialSecret: false,
        adminAuth: null,
        editorTheme: {
          projects: {
            enabled: false
          }
        }
```

```
    }
```

Update the SiteWise Edge deployment

1. Navigate to the [AWS IoT console](https://console.aws.amazon.com/iot/)
2. Choose **Greengrass devices** in the left navigation menu under the **Manage** section, then **Core devices**.
3. Select the core device connected to your SiteWise Edge Gateway.
4. Choose the **Deployments** tab, then select the **Deployment ID** value.
5. Choose **Actions**, then select **Revise**.
6. Read the pop up message and then choose **Revise Deployment**.
7. In **Step 2 - Select components**, select the following components and then choose **Next**.

   - `aws.greengrass.clientdevices.mqtt.EMQX`
   - `aws.iot.SiteWiseEdgePublisher`
8. In **Step 3 - Configure components**, select the `aws.greengrass.clientdevices.mqtt.EMQX` component value and add the following network configuration:

```
{
    "emqxConfig": {
        "authorization": {
            "no_match": "allow"
        },
        "listeners": {
            "tcp": {
                "default": {
                    "enabled": true,
                    "enable_authn": false
                }
            }
        }
    },
    "authMode": "bypass",
    "dockerOptions": "-p 127.0.0.1:1883:1883 --
network=SiteWiseEdgeNodeRedDemoNetwork",
    "requiresPrivilege": "true"
}
```

9. Choose **Next**.
10. In **Step 4 - Configure advanced settings**, choose **Next**.
11. Choose **Deploy**

Launch the services

1. Start the services using the Docker Compose file. Run the following command under the directory containing the `compose.yaml` file.

```
docker compose up -d
```

2. Create an SSH tunnel to access the services:

```
ssh -i path_to_your_ssh_key -L 1880:127.0.0.1:1880 -L 3000:127.0.0.1:3000 -L
 8086:127.0.0.1:8086 username@gateway_ip_address
```

This deployment creates the following services in the `SiteWiseEdgeNodeRedDemoNetwork network`:

InfluxDB v2 (port 8086)

Includes pre-configured organization (iot-sitewise-edge), WindFarmData InfluxDB bucket, and admin credentials

Node-RED (port 1880)

Includes InfluxDB nodes and pre-configured flows for AWS IoT SiteWise integration

Grafana (port 3000)

Includes admin user, InfluxDB datasource, and monitoring dashboard

Access the services

After deployment, access the services using the following URLs and credentials:

> **Note**
>
> You can access each service from your host or the gateway machine.

Service access details

Service	URL	Credentials
Node-RED	http://127.0.0.1:1880	No credentials required
InfluxDB	http://127.0.0.1:8086	Username: admin Password: $INFLUXDB _PASSWORD
Grafana	http://127.0.0.1:3000	Username: admin Password: $GRAFANA_ PASSWORD

Verify the deployment

To ensure your deployment is successful, perform the following checks:

1. For Node-RED, verify the presence of two preloaded flows:
   - Data publish flow
   - Data retention flow
2. For AWS IoT SiteWise, in the AWS IoT SiteWise console, confirm the presence of a data stream with the alias `/Renton/WindFarm/Turbine/WindSpeed`.
3. For InfluxDB, use the Data Explorer to verify data storage in the `TurbineData` measurement within the `WindFarmData` bucket.
4. For Grafana, view the dashboard to confirm the display of time series data generated from Node-RED.

Process data for open source integrations

The data can be processed (such as transformation or aggregation), at different stages using various tools, each serving different monitoring requirements.

Process data with Node-RED nodes

Transform your data in real time using Node-RED® built-in processing nodes. Configure these nodes through the Node-RED console to create your data pipeline.

Data transformation nodes

Transform individual data points, similar to Transforms in AWS IoT SiteWise, using these nodes:

- **change node** - Performs simple value modifications on your data.
- **function node** - Enables custom JavaScript transformations for complex data processing.

Metrics calculation nodes

Combine multiple data points into a single output, similar to Metrics in AWS IoT SiteWise, using these nodes:

- **batch node** - Groups multiple messages for batch processing.
- **join node** - Combines multiple data streams into a single output.
- **aggregator node** - Calculates aggregate metrics from multiple data points.

For additional node options, see the Node-RED Library.

Create InfluxDB tasks

While Node-RED excels at basic data processing with quick setup, complex metric calculations may become challenging in flow-based programming. InfluxDB® Tasks provide an alternative through scheduled Flux scripts for advanced processing needs.

Use InfluxDB Tasks for:

- Statistical aggregations across large datasets
- Mathematical operations on multiple properties
- Derived measurements from multiple sources

Task features

- **Scheduled Execution** - Run tasks based on cron expressions

- **Batch Processing** - Optimize operations for time-series data
- **Error Recovery** - Automatically retry failed operations
- **Monitoring** - Track execution through detailed logs

Manage tasks through the InfluxDB UI, API, or CLI. For more information, see Process data with InfluxDB tasks.

Use Grafana transformations

Transform data visualization in Grafana® without modifying the source data in InfluxDB. Grafana transformations apply only to the visualization layer.

- **Visual Builder** - Create transformations without writing code
- **Live Preview** - View transformation results in real time
- **Multi-Source** - Process data from multiple database sources
- **Storage Efficient** - Transform data at visualization time without storing intermediary results

For more information, see Transform data.

Troubleshooting open-source integrations

For more information on troubleshooting topics related to open source integrations for SiteWise Edge gateways, see Troubleshooting open-source integrations at the Edge.

Classic streams, V2 gateways for AWS IoT SiteWise Edge

Understand the features and limitations of Classic streams, V2 gateways for AWS IoT SiteWise Edge.

The Classic streams, V2 gateway maintains traditional functionality familiar from earlier AWS IoT SiteWise deployments before the introduction of MQTT-enabled, V3 gateways. These SiteWise Edge gateways are considered Classic streams, V2 gateways. They maintain backward compatibility and are functional with the data processing pack. While the Classic streams, V2 gateway offers reliable performance for existing setups, it has limitations compared to newer gateway options. Specifically, this gateway type is not fully compatible with the advanced features available in the MQTT-enabled, V3 gateway destination. To use the MQTT messaging protocol, you can create a new MQTT-enabled, V3 gateway. For more information, see MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge.

Topics

Use packs to collect and process data in SiteWise Edge

AWS IoT SiteWise Edge 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 SiteWise Edge gateway.
- **Data processing pack –** Use this pack to enable SiteWise Edge 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 "Configure edge data processing".

Upgrading packs

> **Important**
>
> Upgrading data processing pack versions from before (and including) 2.0.x to version 2.1.x will result in data loss of locally stored measurements.

SiteWise Edge gateways use different packs to determine how to collect and process your data. You can use the AWS IoT SiteWise console to upgrade packs.

To upgrade packs (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.

3. In the **Gateways** list, choose the SiteWise Edge gateway with the packs you want to upgrade.
4. In the **Gateway configuration** section, choose **Software updates available**.
5. On the Edit software versions page, choose **Updates**.

   > **Note**
   >
   > You can only upgrade packs that are enabled. To find the list of packs that are enabled for this SiteWise Edge gateway, choose **Overview**, and then see the **Edge capabilities** section.

6. On the edit software versions page, in the **Gateway component updates** section, do the following:

   - To update the **OPC UA collector**, choose a version, and then choose **Deploy**.
   - To update the **Publisher**, choose a version, and then choose **Deploy**.
   - To update the **Data processing pack**, choose a version, and then choose **Deploy**.
7. When you're done deploying new versions, choose **Done**.

If you're experiencing problems upgrading the packs, see Unable to deploy packs to SiteWise Edge gateways.

Configure the AWS IoT SiteWise publisher component

After you create an AWS IoT SiteWise Edge gateway and install the software, you can set up the publisher component so your SiteWise Edge gateway can export data to the AWS Cloud. Use the publisher component to enable additional features or configure default settings. For more information, see AWS IoT SiteWise publisher in the *AWS IoT Greengrass Version 2 Developer Guide*.

> **Note**
>
> The publisher configuration differs based on the type of gateway you're using. For Classic stream, V2 gateways, use the `iotsitewise:publisher:2` namespace. For MQTT-enabled, V3 gateways, use the `iotsitewise:publisher:3` namespace.

Console

1. Navigate to the AWS IoT SiteWise console.

2. In the navigation pane, choose **Edge gateways**.
3. Select the SiteWise Edge gateway for which you want to configure the publisher.
4. In the **Publisher configuration** section, choose **Edit**
5. For **Publishing order**, choose one of the following:
   - **Publish oldest data first** – The SiteWise Edge gateway publishes the oldest data to the cloud first by default.
   - **Publish newest data first** – The SiteWise Edge gateway publishes the newest data to the cloud first.
6. (Optional) If you don't want the SiteWise Edge gateway to compress your data, unselect **Activate compression when uploading data**.
7. (Optional) If you don't want to publish old data, choose **Exclude expired data** and do the following:
   - For **Cutoff period**, enter a value and choose a unit. The cutoff period must be between five minutes and seven days. For example, if the cutoff period is three days, data that's older than three days isn't published to the cloud.
8. (Optional) To set custom settings about how data is handled on your local device, choose **Local storage settings** and do the following:
   a. For **Retention period**, enter a number and choose a unit. The retention period must be between one minute and 30 days, and greater than or equal to the rotation period. For example, if the retention period is 14 days, the SiteWise Edge gateway deletes any data at the edge that's older than the specified cutoff period after it's stored for 14 days.
   b. For **Rotation period**, enter a number and choose a unit. The rotation period must be greater than one minute, and equal to, or less than, the retention period. For example, say the rotation period is two days, the SiteWise Edge gateway batches up and saves data that is older than the cutoff period to a single file. For self-hosted gateways through AWS IoT Greengrass V2, the SiteWise Edge gateway transfers a batch of data to the following local directory once every two days: `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/exports`.
   c. For **Storage capacity**, enter a value that is greater than or equal to 1. If the storage capacity is 2 GB, the SiteWise Edge gateway starts deleting data when more than 2 GB of data is stored locally.
9. Choose **Save**.

AWS CLI

Use the UpdateGatewayCapabilityConfiguration API to configure the publisher.

Set the `capabilityNamespace` parameter to `iotsitewise:publisher:2`.

Example : Publisher configuration for Classic Stream, V2 gateways

The publisher namespace: `iotsitewise:publisher:2`

```
{
    "SiteWisePublisherConfiguration": {
        "publishingOrder": "TIME_ORDER",
        "enableCompression": true,
        "dropPolicy": {
            "cutoffAge": "7d",
            "exportPolicy": {
                "retentionPeriod": "7d",
                "rotationPeriod": "6h",
                "exportSizeLimitGB": 10
            }
        }
    },
    "SiteWiseS3PublisherConfiguration": {
        "accessRoleArn": "arn:aws:iam:123456789012:role/roleName",
        "streamToS3ConfigMapping": [
            {
                "streamName": "S3_OPC-UA_Data_Collector",
                "targetBucketArn": "arn:aws:s3:::amzn-s3-demo-bucket/dataCollector",
                "publishPolicy": {
                    "publishFrequency": "10m",
                    "localSizeLimitGB": 10
                },
                "siteWiseImportPolicy": {
                    "enableSiteWiseStorageImport": true,
                    "enableDeleteAfterImport": true
                }
            }
        ]
    }
}
```

The publisher provides the following configuration parameters that you can customize:

`SiteWisePublisherConfiguration`

`publishingOrder`

The order in which data is published to the cloud. The value of this parameter can be one of the following:

- `TIME_ORDER` (**Publish oldest data first**) – The earliest data is published to the cloud first, by default.
- `RECENT_DATA` (**Publish newest data first**) – The newest data is published to the cloud first.

`enableCompression`

Set this to `true` to compress data before publishing. Data compression can reduce bandwidth usage.

`dropPolicy`

(Optional) A policy that controls what data is published to the cloud.

`cutoffAge`

The maximum age of data to be published specified in days, hours, and minutes. For example, `7d` or `1d7h16m`. Data older than what you specify is not sent to AWS IoT SiteWise.

Data that is earlier than the cutoff period is not published to the cloud. The cutoff age must be between five minutes and seven days.

You can use `m`, `h`, and `d` when you specify a cutoff age. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

`exportPolicy`

(Optional) A policy that manages data storage at the edge. This policy applies to data that is earlier than the cutoff age.

`retentionPeriod`

Your SiteWise Edge gateway deletes any data at the edge that is earlier than the cutoff period from the local storage after it is stored for the specified retention period. The retention period must be between one minute and 30 days, and greater than or equal to the rotation period.

You can use `m`, `h`, and `d` when you specify a retention period. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

`rotationPeriod`

The time interval over which to batch up and save data that is earlier than the cutoff period to a single file. The SiteWise Edge gateway transfers one batch of data to the following local directory at the end of each rotation period: `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/exports`. The rotation period must be greater than one minute, and equal to or less than the retention period.

You can use `m`, `h`, and `d` when you specify a rotation period. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

`exportSizeLimitGB`

The maximum allowed size of data stored locally, in GB. If this quota is breached, the SiteWise Edge gateway starts deleting the earliest data until the size of data stored locally is equal to or less than the quota. The value of this parameter must be greater than or equal to 1.

`SiteWiseS3PublisherConfiguration`

`accessRoleArn`

The access role that gives AWS IoT SiteWise permission to manage the Amazon S3 bucket that you are publishing to.

`streamToS3ConfigMapping`

An array of configurations that maps a stream to an Amazon S3 configuration.

`streamName`

The stream to read from and publish to the Amazon S3 configuration.

`targetBucketArn`

The bucket ARN to publish to.

`publishPolicy`

`publishFrequency`

The frequency with which the SiteWise Edge gateway publishes to the Amazon S3 bucket.

`localSizeLimitGB`

The maximum size of the files written to local disk. If this threshold is breached, the publisher publishes all buffered data to its destination.

`siteWiseImportPolicy`

`enableSiteWiseStorageImport`

Set this to `true` to import data from an Amazon S3 bucket to AWS IoT SiteWise storage.

`enableDeleteAfterImport`

Set this to `true` to delete the file in the Amazon S3 bucket after ingestion into the AWS IoT SiteWise storage.

Destinations and AWS IoT Greengrass stream manager

AWS IoT Greengrass stream manager allows you 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*.

Example : Data stream message structure

The following example shows the required data stream message structure transmitted by the AWS IoT Greengrass stream manager.

```
{
   "assetId": "string",
   "propertyAlias": "string",
   "propertyId": "string",
   "propertyValues": [
      {
         "quality": "string",
         "timestamp": {
            "offsetInNanos": number,
            "timeInSeconds": number
         },
         "value": {
```

```
            "booleanValue": boolean,
            "doubleValue": number,
            "integerValue": number,
            "stringValue": "string"
         }
      }
   ]
}
```

Note

The data stream message must include either (`assetId` and `propertyId`) or `propertyAlias` in its structure.

`assetId`

(Optional) The ID of the asset to update.

`propertyAlias`

(Optional) The alias that identifies the property, such as an OPC UA server data stream path. For example:

```
/company/windfarm/3/turbine/7/temperature
```

For more information, see [Manage data streams](#) in the *AWS IoT SiteWise User Guide*.

`propertyId`

(Optional) The ID of the asset property for this entry.

`propertyValues`

(Required) The list of property values to upload. You can specify up to 10 `propertyValues` array elements.

`quality`

(Optional) The quality of the asset property value.

`timestamp`

(Required) The timestamp of the asset property value.

`offsetInNanos`

(Optional) The nanosecond offset from `timeInSeconds`.

`timeInSeconds`

(Required) The timestamp date, in seconds, in the Unix epoch format. Fractional nanosecond data is provided by `offsetInNanos`.

`value`

(Required) The value of the asset property.

> **Note**
>
> Only one of the following values can exist in the `value` field.

`booleanValue`

(Optional) Asset property data of type Boolean (`true` or `false`).

`doubleValue`

(Optional) Asset property data of type double (floating point number).

`integerValue`

(Optional) Asset property data of type integer (whole number).

`stringValue`

(Optional) Asset property data of type string (sequence of characters).

Configure edge capabilities on AWS IoT SiteWise Edge

You can use AWS IoT SiteWise Edge to collect and temporarily store data so that you can organize and process device data locally. By enabling edge processing, you can choose to send only aggregated data to the AWS Cloud to optimize your bandwidth usage and cloud storage costs. Using AWS IoT SiteWise components with AWS IoT Greengrass, you can collect and process data at the edge before sending it to the AWS Cloud, or manage it on-premises using SiteWise Edge APIs.

Data collection happens through data packs and AWS IoT SiteWise components that run on AWS IoT Greengrass.

> **Note**
>
> - AWS IoT SiteWise retains your edge data on your SiteWise Edge gateways up to 30 days. The retention period of your data is dependent on the available disk space of your device.
> - If your SiteWise Edge gateway has been disconnected from the AWS Cloud for 30 days, the Data Processing Pack is automatically disabled.

Topics

Set up edge capability in SiteWise Edge

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

- **Data collection pack** enables your SiteWise Edge gateway to collect data from multiple OPC UA servers, and then export the data from the edge to the AWS Cloud. It becomes active once you have added data sources to your SiteWise Edge gateway.
- **Data processing pack** enables your SiteWise Edge gateway to process your equipment 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 Model industrial assets.

> **Note**
>
> - The data processing pack is only available on x86 platforms.

To configure edge capabilities

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the SiteWise Edge gateway for which you want to activate edge capabilities.
4. In the **Edge capabilities** section, choose **Edit**

5. In the **Edge capabilities** section, select **Enable data processing pack (incurs additional charges)**.

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

   > **Note**
   >
   > You can also use the Linux or Microsoft Windows credentials to access the SiteWise Edge gateway. For more information, see Access your SiteWise Edge gateway using Linux operating system credentials.

   a. Select **Activated**.

   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 LDAP server.

   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.

7. Choose **Save**.

Now that you've activated edge capabilities on your SiteWise Edge gateway, you need to 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.

For more information about asset properties, see the section called “Define data properties”.

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 “Create an asset model (console)”.

> **Note**
>
> Asset models and dashboards are automatically synced between the AWS Cloud and your SiteWise Edge gateway every 10 minutes. You can also sync manually from the local SiteWise Edge gateway application.

Configure edge data processing for AWS IoT SiteWise models and assets

You can use AWS IoT SiteWise Edge to collect, store, organize and monitor equipment data locally. You can use SiteWise Edge so that you can model your industrial data and SiteWise Monitor to create dashboards for your operational staff to visualize data locally. You can process your data locally and send it to the AWS Cloud, or process it on-premises by using the AWS IoT SiteWise API.

With AWS IoT SiteWise Edge, you can process raw data locally and choose to send only aggregated data to the AWS Cloud to optimize your bandwidth usage and cloud storage costs.

> **Note**
>
> - AWS IoT SiteWise retains your edge data on your SiteWise Edge gateways up to 30 days. The retention period of your data is dependent on the available disk space of your device.
> - If your SiteWise Edge gateway has been disconnected from the AWS Cloud for 30 days, the Set up an OPC UA source in SiteWise Edge is automatically disabled.

Configure an asset model for data processing on SiteWise Edge

You must configure your asset model for the edge before your can process your SiteWise Edge gateway data at the edge. Your asset model edge configuration specifies where your assets properties are computed. You can choose to compute all properties at the edge and send the results to the AWS Cloud, or customize where to compute each asset property separately. For more information, see Configure edge data processing for AWS IoT SiteWise models and assets.

Asset 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 “Define data properties”.

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 “Create an asset model (console)”.

> **Note**
>
> Asset models and dashboards are automatically synced between the AWS Cloud and your SiteWise Edge gateway every 10 minutes. You can also sync manually from the Manage SiteWise Edge gateways.

You can use the AWS IoT SiteWise REST APIs and the AWS Command Line Interface (AWS CLI) to query your SiteWise Edge gateway for data at the edge. Before you query your SiteWise Edge 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 “Manage gateways”.
- The SDK endpoint must point to the IP address of your SiteWise Edge 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 SiteWise Edge gateway certificate must be registered. You can find more information about registering your SiteWise Edge 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 *Query data from AWS IoT SiteWise*.

Add data sources to your AWS IoT SiteWise Edge gateway

After setting up an AWS IoT SiteWise Edge gateway, you can add and configure data sources to ingest data from local industrial equipment to AWS IoT SiteWise. SiteWise Edge supports various protocols, including OPC UA, and many other protocols available through partner data sources. These sources enable your gateway to connect with local servers and retrieve your industrial data. By configuring data sources, you can ingest data from a variety of data sources, and then associate the data streams with asset properties, enabling comprehensive industrial asset modeling and data mapping in AWS IoT SiteWise.

Topics

OPC UA data sources for AWS IoT SiteWise Edge gateways

After you set up an AWS IoT SiteWise Edge gateway, you can configure data sources so that your SiteWise Edge gateway can ingest data from local industrial equipment to AWS IoT SiteWise. Each source represents a local server, such as an OPC UA server, that your SiteWise Edge gateway connects and retrieves industrial data streams. For more information about setting up a SiteWise Edge gateway, see Create a self-hosted SiteWise Edge gateway.

The gateway type, MQTT-enabled, V3 gateways versus Classic stream, V2 gateways, influences how OPC UA data is handled. In Classic stream, V2 gateways, OPC UA data sources are added directly to the gateway IoT SiteWise publisher configuration. Each data source is coupled with the gateway, and data routing is configured individually for each source. In contrast, using MQTT-enabled, V3 gateways, OPC UA data sources are converted to MQTT topics and are managed through centralized destinations. For more information on each type, see MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge and Classic streams, V2 gateways for AWS IoT SiteWise Edge.

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

You can view CloudWatch metrics to verify that a data source is connected to AWS IoT SiteWise. For more information, see AWS IoT Greengrass Version 2 gateway metrics.

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

- OPC UA – A machine-to-machine (M2M) communication protocol for industrial automation.

Support for additional industrial protocols

SiteWise Edge supports a wide range of industrial protocols through integration with data source partners. These partnerships enable connectivity with over 200 different protocols, accommodating various industrial systems and devices.

For a list of available data source partners, see SiteWise Edge gateway partner data source options.

Set up an OPC UA source in SiteWise Edge

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

Topics

Configure an OPC UA source (console)

You can use the console to configure the OPC UA source with the following procedure.

> **Note**
>
> Warning: Duplicate TQVs may result in double charging.

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

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the SiteWise Edge gateway to add an OPC UA source.
4. Choose **Add data source**.
5. Enter a name for the source.
6. 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 this: **`opc.tcp://203.0.113.0:49320`**
7. (Optional) For **Node ID for selection**, add node filters to limit which data streams are ingested to the AWS Cloud. By default, SiteWise Edge gateways use the root node of a server to ingest all data streams. You can use node filters to reduce your SiteWise Edge gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, SiteWise Edge 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 Use OPC UA node filters in SiteWise Edge.
8. **Destinations** vary between MQTT-enabled, V3 gateways and Classic streams, V2 gateways.

   - **Classic steams, V2 gateway destinations** have a 1:1 relationship with the source. Each source sends data to a particular destination.
   - **MQTT-enabled, V3 gateway destinations** are set up separately because the hub and spoke model lets you centralize configuration and management of multiple data sources across different gateways. To set up destinations in a V3 gateway, see Understand AWS IoT SiteWise Edge destinations.

Classic steams, V2 gateway destinations

- **AWS IoT SiteWise real-time –** Choose this to send data directly to AWS IoT SiteWise storage. Ingest and monitor data in real-time at the edge.
- **AWS IoT SiteWise Buffered using Amazon S3 –** Send data in Parquet format to Amazon S3 and then import into AWS IoT SiteWise storage. Choose this option to ingest data in batches, and store historical data in a cost-effective way. You can configure your preferred Amazon S3 bucket location, and the frequency at which you want data to be uploaded to Amazon S3. You can also choose what to do with the data after ingestion into AWS IoT SiteWise. You can choose to have the data available in both AWS IoT SiteWise and Amazon S3 or you can choose to delete it automatically from Amazon S3 after it has been imported into AWS IoT SiteWise.
  - The Amazon S3 bucket is a staging and buffering mechanism and supports files in the Parquet format.
  - If you select the check box **Import data into AWS IoT SiteWise storage**, data is uploaded into Amazon S3 first, and then into AWS IoT SiteWise storage.
    - If you select the check box **Delete data from Amazon S3**, data is deleted from Amazon S3, after it is imported into SiteWise storage.
    - If you clear the check box **Delete data from Amazon S3**, data is stored both in Amazon S3, and in SiteWise storage.
  - If you clear the check box **Import data into AWS IoT SiteWise storage**, data is stored only in Amazon S3. It is not imported into SiteWise storage.

  Visit Manage data storage for details on the various storage options AWS IoT SiteWise provides. To learn more about pricing options, see AWS IoT SiteWise pricing.

- **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.

MQTT-enabled, V3 gateway destinations

1. See MQTT-enabled, V3 Gateways for AWS IoT SiteWise Edge for information on adding your relevant destinations.
2. Return to this procedure after adding your source destinations.

9. In the **Advanced configuration** pane, you can do the following:

   a. Choose a **Message security mode** for connections and data in transit between your source server and your SiteWise Edge gateway. This field is the combination of the OPC UA security policy and message security mode. Choose the same security policy and message security mode that you specified for your OPC UA server.

   b. If your source requires authentication, choose an AWS Secrets Manager secret from the **Authentication configuration** list. The SiteWise Edge gateway uses the authentication credentials in this secret when it connects to this data source. You must attach secrets to your SiteWise Edge gateway's AWS IoT Greengrass component to use them for data source authentication. For more information, see the section called "Configure data source authentication".

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

   c. (Optional) You can activate a data stream prefix by selecting **Activate data stream prefix - *optional***.

      - Enter a **Data stream prefix**. The SiteWise Edge 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.

   d. (Optional) Choose a **Data type conversion** option to convert unsupported OPC UA data types into strings before ingesting them into AWS IoT SiteWise. Convert array values with simple data types to JSON strings and DateTime data types to ISO 8601 strings. For more information, see Converting unsupported data types.

   e. (Optional) For **Property groups**, choose **Add new group**.

i. Enter a **Name** for the property group.

ii. For **Properties**:

   1. For **Node paths**, add OPC UA node filters to limit which OPC UA paths are uploaded to AWS IoT SiteWise. The format is similar to **Node ID for selection**.

iii. For **Group settings**, do the following:

   1. For **Data quality setting**, choose the type of data quality that you want AWS IoT SiteWise Collector to ingest.
   2. For **Scan mode setting**, configure the standard subscription properties using **Scan mode**. You can select **Subscribe** or **Poll**. For more information about scan mode, see the section called "Filter data ingestion ranges".

      Subscribe

      To send every data point

      i. Choose **Subscribe** and set the following:

         A. **Data change trigger** – The condition that initiates a data change alert.

         B. **Subscription queue size** – The depth of the queue on an OPC–UA server for a particular metric where notifications for monitored items are queued.

         C. **Subscription publishing interval** – The interval (in milliseconds) of publishing cycle specified when subscription is created.

         D. **Snapshot interval - *Optional*** – The snapshot frequency timeout setting to ensure that AWS IoT SiteWise Edge ingests a steady stream of data.

         E. **Scan rate** – The rate that you want the SiteWise Edge gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your SiteWise Edge gateway.

         F. **Timestamp** – The timestamp to include with your OPC UA data points. You can use the server timestamp or your device's timestamp.

> **Note**
>
> Use version 2.5.0 or later of the IoT SiteWise OPC UA collector component. If you use the timestamp feature with earlier versions, configuration updates fail. For more information, see Update the version of an AWS IoT SiteWise component.

ii. In **Deadband settings**, configure a **Deadband type**. The deadband type 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 "Filter data ingestion ranges".

- **None** – The associated server sends all data points for this property group.
- **Percentage** – The associated server only sends data that falls outside a specified percentage of the data's range. This range is computed by the server based on the engineering unit minimum and maximum defined for each node. If the server does not support percentage deadbands or lacks defined engineering units, the gateway calculates the range using the minimum and maximum values provided below.
- **Absolute** – The associated server only sends data that falls outside of a specific range.

A. Set the **Deadband value** as the percentage of the data range to deadband.

B. (Optional) Specify a minimum and maximum for the deadband range using **Minimum range - *optional*** and **Maximum range - *optional***.

Poll

To send data points at a specific interval

- Choose **Poll** and set the following:

  A. **Scan rate** – The rate that you want the SiteWise Edge gateway to read your registers. AWS IoT SiteWise automatically calculates the minimum allowable scan rate for your SiteWise Edge gateway.

B. **Timestamp** – The timestamp to include with your OPC UA data points. You can use the server timestamp or your device's timestamp.

> **Note**
>
> Use version 2.5.0 or later of the IoT SiteWise OPC UA collector component. If you use the timestamp feature with earlier versions, configuration updates fail. For more information, see [Update the version of an AWS IoT SiteWise component](#).

> **Note**
>
> **Deadband settings** are applicable when you've selected **Subscribe** in the **Scan mode settings**.

10. Choose **Save**.

Configure an OPC UA source (AWS CLI)

You can define OPC UA data sources for an SiteWise Edge gateway using the AWS CLI. To do this, create an OPC UA capability configuration JSON file and use the [update-gateway-capability-configuration](#) command to update the SiteWise Edge gateway configuration. You must define all of your OPC UA sources in a single capability configuration.

MQTT-enabled, V3 gateway

This capability has the following namespace.

- `iotsitewise:opcuacollector:3`

```
{
  "sources": [
    {
      "name": "string",
      "endpoint": {
        "certificateTrust": {
          "type": "TrustAny" | "X509",
          "certificateBody": "string",
```

```
          "certificateChain": "string",
        },
        "endpointUri": "string",
        "securityPolicy": "NONE" | "BASIC128_RSA15" | "BASIC256" | "BASIC256_SHA256"
 | "AES128_SHA256_RSAOAEP" | "AES256_SHA256_RSAPSS",
        "messageSecurityMode": "NONE" | "SIGN" | "SIGN_AND_ENCRYPT",
        "identityProvider": {
          "type": "Anonymous" | "Username",
          "usernameSecretArn": "string"
        },
        "nodeFilterRules": [
          {
            "action": "INCLUDE",
            "definition": {
              "type": "OpcUaRootPath",
              "rootPath": "string"
            }
          }
        ]
      },
      "measurementDataStreamPrefix": "string",
      "typeConversions": {
        "array": "JsonArray",
        "datetime": "ISO8601String"
        },
      "destination": {
        {
          "type":"MQTT"
        }
      },
      "propertyGroups": [
        {
          "name": "string",
          "nodeFilterRuleDefinitions": [
            {
              "type": "OpcUaRootPath",
              "rootPath": "string"
            }
          ],
          "deadband": {
            "type": "PERCENT" | "ABSOLUTE",
            "value": double,
            "eguMin": double,
            "eguMax": double,
```

```
              "timeoutMilliseconds": integer
            },
            "scanMode": {
              "type": "EXCEPTION" | "POLL",
              "rate": integer,
              "timestampToReturn": "SOURCE_TIME" | "SERVER_TIME"
            },
            "dataQuality": {
              "allowGoodQuality": true | false,
              "allowBadQuality": true | false,
              "allowUncertainQuality": true | false
            },
            "subscription": {
              "dataChangeTrigger": "STATUS" | "STATUS_VALUE" |
 "STATUS_VALUE_TIMESTAMP",
              "queueSize": integer,
              "publishingIntervalMilliseconds": integer,
              "snapshotFrequencyMilliseconds": integer
            }
          }
        ]
      }
    ]
}
```

Classic streams, V2 gateway

This capability has the following namespace.

- `iotsitewise:opcuacollector:2`

Request syntax

```
{
  "sources": [
    {
      "name": "string",
      "endpoint": {
        "certificateTrust": {
          "type": "TrustAny" | "X509",
          "certificateBody": "string",
          "certificateChain": "string",
        },
```

```
        "endpointUri": "string",
        "securityPolicy": "NONE" | "BASIC128_RSA15" | "BASIC256" | "BASIC256_SHA256"
 | "AES128_SHA256_RSAOAEP" | "AES256_SHA256_RSAPSS",
        "messageSecurityMode": "NONE" | "SIGN" | "SIGN_AND_ENCRYPT",
        "identityProvider": {
          "type": "Anonymous" | "Username",
          "usernameSecretArn": "string"
        },
        "nodeFilterRules": [
          {
            "action": "INCLUDE",
            "definition": {
              "type": "OpcUaRootPath",
              "rootPath": "string"
            }
          }
        ]
      },
      "measurementDataStreamPrefix": "string",
      "typeConversions": {
        "array": "JsonArray",
        "datetime": "ISO8601String"
        },
      "destination": {
        "type": "StreamManager",
        "streamName": "string",
        "streamBufferSize": integer,
      },
      "propertyGroups": [
        {
          "name": "string",
          "nodeFilterRuleDefinitions": [
            {
              "type": "OpcUaRootPath",
              "rootPath": "string"
            }
          ],
          "deadband": {
            "type": "PERCENT" | "ABSOLUTE",
            "value": double,
            "eguMin": double,
            "eguMax": double,
            "timeoutMilliseconds": integer
          },
```

```
          "scanMode": {
            "type": "EXCEPTION" | "POLL",
            "rate": integer,
            "timestampToReturn": "SOURCE_TIME" | "SERVER_TIME"
          },
          "dataQuality": {
            "allowGoodQuality": true | false,
            "allowBadQuality": true | false,
            "allowUncertainQuality": true | false
          },
          "subscription": {
            "dataChangeTrigger": "STATUS" | "STATUS_VALUE" |
 "STATUS_VALUE_TIMESTAMP",
            "queueSize": integer,
            "publishingIntervalMilliseconds": integer,
            "snapshotFrequencyMilliseconds": integer
          }
        }
      ]
    }
  ]
}
```

Request body

`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 SiteWise Edge gateway trusts any certificate when it connects to the OPC UA source.
- X509 – The SiteWise Edge 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 SiteWise Edge 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.
- 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 Profile SecurityPolicy – 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 SecurityPolicy – 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 SiteWise Edge gateway. For more information, see Set up OPC UA servers to trust the AWS IoT SiteWise Edge gateway.

`messageSecurityMode`

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

- NONE – The SiteWise Edge 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 SiteWise Edge 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 SiteWise Edge gateway. For more information, see Set up OPC UA servers to trust the AWS IoT SiteWise Edge gateway.

`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 SiteWise Edge gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your SiteWise Edge gateway's IoT SiteWise connector to use them for source authentication. For more information, see Configure data source authentication for SiteWise Edge.

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 SiteWise Edge gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, SiteWise Edge 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 Use OPC UA node filters in SiteWise Edge.

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 SiteWise Edge 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 SiteWise Edge 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 SiteWise Edge 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.

`typeConversions`

The types of conversions available for unsupported OPC UA data types. Each data type is converted to strings. For more information, see [Converting unsupported data types].

`array`

The simple array data type that is converted to strings. You can choose the following option:

- `JsonArray` – Indicates that you choose to convert your simple array data types to strings.

`datetime`

The DateTime data type that is converted to strings. You can choose the following option:

- `ISO8601String` – Indicates that you choose to convert ISO 8601 data types to strings.

`destination`

Configuration for the destination of OPC UA tags. Classic stream, v2 and MQTT-enabled, V3 gateways have differing configurations for destinations.

`type`

The type of the destination.

`streamName` – *only for Classic streams, V2 gateways*

The name of the stream. The stream name should be unique.

`streamBufferSize` – *only for Classic streams, V2 gateways*

The buffer size of the stream. This is important for managing the flow of data from OPC UA sources.

`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` value defines the minimum change in a data point's value that must occur before the data is sent to the cloud. It contains the following information:

`type`

The supported types of deadband. You can choose the following options:

- `ABSOLUTE` – A fixed value that specifies the minimum absolute change required to consider a data point significant enough to be sent to the cloud.
- `PERCENT` – A dynamic value that specifies the minimum change required as a percentage of the last sent data point's value. This type of deadband is useful when the data values vary significantly over time.

`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.

`timestampToReturn`

The source of the timestamp. You can choose the following options:

- `SOURCE_TIME` – Uses the timestamp from your device.
- `SERVER_TIME` – Uses the timestamp from your server.

> **Note**
>
> Use `TimestampToReturn` with version 2.5.0 or later of the IoT SiteWise OPC UA collector component. If you use this feature with earlier versions, configuration updates fail. For more information, see Update the version of an AWS IoT SiteWise component.

`nodeFilterRuleDefinitions`

(Optional) A list of node paths to include in the property group. Property groups can't overlap. If you don't specify a value for this field, the group contains all paths under the root, and you can't create additional property groups. The `nodeFilterRuleDefinitions` structure contains the following information:

`type`

`OpcUaRootPath` is the only supported type. This specifies that the value of `rootPath` is a path relative to the root of the OPC UA browsing space.

`rootPath`

A comma-delimited list that specifies the paths (relative to the root) to include in the property group.

Additional capability configuration examples for Classic streams, V2 gateways (AWS CLI)

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

```
aws iotsitewise update-gateway-capability-configuration \
--capability-namespace "iotsitewise:opcuacollector:2" \
--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.

```
{
    "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.

```
{
    "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": {
                            "type": "OpcUaRootPath",
                            "rootPath": "/Utilities/Tank"
                        }
                    }
                ]
            },
            "measurementDataStreamPrefix": "propertyGroups",
            "propertyGroups": [
                 {
                     "name": "Deadband_Abs_5",
                     "nodeFilterRuleDefinitions": [
                         {
                             "type": "OpcUaRootPath",
                             "rootPath": "/Utilities/Tank/Temperature/TT-001"
                         },
                         {
                             "type": "OpcUaRootPath",
                             "rootPath": "/Utilities/Tank/Temperature/TT-002"
                         }
                     ],
                     "deadband": {
                         "type":"ABSOLUTE",
                         "value": 5.0,
                         "timeoutMilliseconds": 120000
                     }
                 },
                 {
                     "name": "Polling_10s",
                     "nodeFilterRuleDefinitions": [
                         {
                             "type": "OpcUaRootPath",
                             "rootPath": "/Utilities/Tank/Pressure/PT-001"
                         }
```

```
                        ],
                        "scanMode": {
                            "type": "POLL",
                            "rate": 10000
                        }
                    },
                    {
                        "name": "Percent_Deadband_Timeout_90s",
                        "nodeFilterRuleDefinitions": [
                            {
                                "type": "OpcUaRootPath",
                                "rootPath": "/Utilities/Tank/Flow/FT-*"
                            }
                        ],
                        "deadband": {
                            "type":"PERCENT",
                            "value": 5.0,
                            "eguMin": -100,
                            "eguMax": 100,
                            "timeoutMilliseconds": 90000
                        }
                    }
                ]
            }
        ]
}
```

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.

```
{
    "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",
                    "usernameSecretArn":
 "arn:aws:secretsmanager:region:123456789012:secret:greengrass-windfarm2-auth-1ABCDE"
                },
                "nodeFilterRules": [
                  {
                      "action": "INCLUDE",
                      "definition": {
                          "type": "OpcUaRootPath",
                          "rootPath": "/WindFarm/2/WindTurbine/"
                    }
                  }
                ]
            },
            "measurementDataStreamPrefix": "/Washington"
        }
    ]
}
```

Example : OPC UA source configuration with certificate trust

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.

```
{
```

```
    "sources": [
        {
            "name": "Wind Farm #3",
            "endpoint": {
                "certificateTrust": {
                    "type": "X509",
                    "certificateBody": "-----BEGIN CERTIFICATE-----
         MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
         -----END CERTIFICATE-----",
                    "certificateChain": "-----BEGIN CERTIFICATE-----
         MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
         -----END CERTIFICATE-----"
                },
                "endpointUri": "opc.tcp://203.0.113.2:49320",
                "securityPolicy": "BASIC256",
                "messageSecurityMode": "SIGN_AND_ENCRYPT",
```

```
                "identityProvider": {
                    "type": "Anonymous"
                },
                "nodeFilterRules": []
              },
            "measurementDataStreamPrefix": ""

        }
    ]
}
```

Set up OPC UA servers to trust the AWS IoT SiteWise Edge gateway

If you choose a `messageSecurityMode` other than **None** when configuring your OPC UA source, you must enable your source servers to trust the AWS IoT SiteWise Edge gateway. The SiteWise Edge gateway generates a certificate that your source server might require. The process varies depending on your source servers. For more information, see the documentation for your servers.

The following procedure outlines the basic steps.

To enable an OPC UA server to trust the SiteWise Edge gateway

1. Open the interface for configuring your OPC UA server.
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 SiteWise Edge gateway. If this applies to your OPC UA servers, you can use the following procedure to export the OPC UA client certificate from the SiteWise Edge gateway. Then, you can import the certificate on your OPC UA server.

To export the OPC UA client certificate file for a source

1. Run the following command to change to the directory that contains the certificate file. Replace *`sitewise-work`* with the local storage path for the *`aws.iot.SiteWiseEdgeCollectorOpcua`* Greengrass work folder and replace *`source-name`* with the name of the data source.

By default, the Greengrass work folder is */greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua* on Linux and *C:/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua* on Microsoft Windows.

```
cd /sitewise-work/source-name/opcua-certificate-store
```

2. The SiteWise Edge 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`.

```
keytool -exportcert -v -alias aws-iot-opcua-client -keystore aws-iot-opcua-
client.pfx -storepass amazon -storetype PKCS12 -rfc > aws-iot-opcua-client-
certificate.pem
```

3. Transfer the certificate file, `aws-iot-opcua-client-certificate.pem`, from the SiteWise Edge 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 SiteWise Edge 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 using SSH in the *Amazon EC2 User Guide*.

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

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 SiteWise Edge gateway exchange this information.

Collect or filter out data based on quality

You can configure your data quality settings to control what data is collected from the OPC UA source. The data source includes the quality rating as metadata when it sends it. You can select one or all of the following options:

- `Good`
- `Bad`
- `Uncertain`

Handle NaN or null values

SiteWise Edge supports the collection and handling of NaN and null values.

- *NaN (Not a Number):* Represents undefined or unrepresentable numerical results.
- *Null:* Indicates missing data.

The IoT SiteWise OPC UA collector captures NaN and Null values with BAD or UNCERTAIN quality. These special values are written to the local stream, enabling more comprehensive data collection.

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 SiteWise Edge 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 SiteWise Edge gateway receives data.
- Polling mode – Your SiteWise Edge 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 SiteWise Edge 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 SiteWise Edge gateway, see OPC UA data sources for AWS IoT SiteWise Edge gateways.

> **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.

Use OPC UA node filters in SiteWise Edge

When you define OPC UA data sources for an SiteWise Edge gateway, you can define node filters. Node filters let you limit which data stream paths the SiteWise Edge gateway sends to the cloud. You can use node filters to reduce your SiteWise Edge gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, SiteWise Edge 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 SiteWise Edge gateway, see [OPC UA data sources for AWS IoT SiteWise Edge gateways](#).

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

Wildcard	Description
`*`	Matches a single level in a data stream path.
`**`	Matches multiple levels in a data stream path.

> **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 : 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.

- To select all nodes in this example, use `/` or `/**/`. You can include multiple directories or folders with the `**` wildcard characters.
- To select all PLC data streams, use `/*/*/*/PLC*` or `/**/PLC*`.
- To select all counters in this example, use `/**/Counter*` or `/*/*/*/Counter*`.
- To select all counters from `Line 2`, use `/**/Line 2/Counter*`.

Converting unsupported data types

Optionally enable data type conversion in AWS IoT SiteWise for simple arrays and DateTime data types. AWS IoT SiteWise doesn't support all OPC UA data types. When you send unsupported data to your AWS IoT Greengrass data stream, that data is lost. However, by converting the unsupported native data types to strings, you can ingest the data into AWS IoT SiteWise rather than discarding it. AWS IoT SiteWise serializes your converted data so that you can later use your own functions to convert the strings back to their original data type downstream, if needed.

You can update your data type conversion settings for a data source at any time and each data source can have its own settings.

When you add data sources in the AWS IoT SiteWise console, there are two checkboxes under **Data type conversion** in **Advanced Configuration**. You can indicate which data types to convert to strings.

Additionally, the IoT SiteWise OPC UA collector can accept NaN or null values on the edge.

- Convert array values with simple data types to JSON strings
- Convert DateTime values to ISO 8601 strings

Prerequisite

- Use version 2.5.0 or later of the IoT SiteWise OPC UA collector.

Limitations

These are the limitations for OPC UA data type conversion to strings in AWS IoT SiteWise.

- Complex data type conversion is not supported.
- String limits after conversion are 1024 bytes. If the string is longer than 1024 bytes, the string is rejected by AWS IoT SiteWise.

Configure data source authentication for SiteWise Edge

If your OPC UA server requires authentication credentials to connect, you can use AWS Secrets Manager to create and deploy a secret to your SiteWise Edge gateway. AWS Secrets Manager encrypts secrets on the device to keep your user name and password secure until you need to use them. For more information about the AWS IoT Greengrass secret manager component, see Secret manager in the *AWS IoT Greengrass Version 2 Developer Guide*.

For information about managing access to Secrets Manager secrets, see:

- Who has permissions to your AWS Secrets Manager secrets.
- Determining if a request is allowed or denied within an account.

Step 1: Create source authentication secrets

You can use AWS Secrets Manager to create an authentication secret for your data source. In the secret, define **`username`** and **`password`** key-value pairs that contain authentication details for your data source.

To create a secret (console)

1. Navigate to the [AWS Secrets Manager console](https://console.aws.amazon.com/secretsmanager/).
2. Choose **Store a new secret**.
3. Under **Secret type**, choose **Other type of secrets**.
4. Under **Key/value pairs**, do the following:
    1. In the first input box, enter **`username`** and in the second input box enter the username.
    2. Choose **Add row**.
    3. In the first input box, enter **`password`** and in the second input box enter the password.
5. For **Encryption key**, select **aws/secretsmanager**, and then choose **Next**.
6. On the **Store a new secret** page, enter a **Secret name**.
7. (Optional) Enter a **Description** that helps you identify this secret, and then choose **Next**.
8. (Optional) On the **Store a new secret** page, turn on **Automatic rotation**. For more information, see [Rotate secrets] in the *AWS Secrets Manager User Guide*.
9. Specify a rotation schedule.
10. Choose a Lambda function that can rotate this secret, and then choose **Next**.
11. Review your secret configurations, and then choose **Store**.

To authorize your SiteWise Edge gateway to interact with AWS Secrets Manager, the IAM role for your SiteWise Edge gateway must allow the `secretsmanager:GetSecretValue` action. You can use the **Greengrass core device** to search for the IAM policy. For more information about updating an IAM policy, see Editing IAM policies in the *AWS Identity and Access Management User Guide*.

Example policy

Replace *`secret-arn`* with the Amazon Resource Name (ARN) of the secret that you created in the previous step. For more information about how to get the ARN of a secret, see Find secrets in AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

JSON

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

Step 2: Deploy secrets to your SiteWise Edge gateway device

You can use the AWS IoT SiteWise console to deploy secrets to your SiteWise Edge gateway.

To deploy a secret (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Gateways**.
3. From the **Gateways** list, choose the target SiteWise Edge gateway.
4. In the **Gateway configuration** section, choose the **Greengrass core device** link to open the AWS IoT Greengrass core associated with the SiteWise Edge gateway.
5. In the navigation pane, choose **Deployments**.
6. Choose the target deployment, and then choose **Revise**.
7. On the **Specify target** page, choose **Next**.
8. On the **Select components** page, in the **Public components** section, turn off **Show only selected components**.
9. Search for and choose the **aws.greengrass.SecretManager** component, and then choose **Next**.
10. From the **Selected components** list, choose the **aws.greengrass.SecretManager** component, and then choose **Configure component**.

11. In the **Configuration to merge** field, add the following JSON object.

> **Note**
>
> Replace *secret-arn* with the ARN of the secret that you created in the previous step. For more information about how to get the ARN of a secret, see [Find secrets in AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

```
{
"cloudSecrets":[
  {
     "arn":"secret-arn"
  }
]
}
```

12. Choose **Confirm**.
13. Choose **Next**.
14. On the **Configure advanced settings** page, choose **Next**.
15. Review your deployment configurations, and then choose **Deploy**.

Step 3: Add authentication configurations

You can use the AWS IoT SiteWise console to add authentication configurations to your SiteWise Edge gateway.

To add authentication configurations (console)

1. Navigate to the [AWS IoT SiteWise console](#).
2. From the **Gateways** list, choose the target SiteWise Edge gateway.
3. From the **Data sources** list, choose the target data source, and then choose **Edit**.
4. On the **Add a data source** page, choose **Advanced configuration**.
5. For **Authentication configuration**, choose the secret that you deployed in the previous step.
6. Choose **Save**.

Partner data sources on SiteWise Edge gateways

When using an AWS IoT SiteWise Edge gateway you can connect a partner data source to your SiteWise Edge gateway and receive data from the partner in your SiteWise Edge gateway and the AWS cloud. These partner data sources are AWS IoT Greengrass components that are developed in partnership between AWS and the partner. When you add a partner data source, AWS IoT SiteWise will create this component and deploy it on your SiteWise Edge gateway.

> **Note**
>
> You can add one data source for each partner in each gateway.

To add a partner data source, do the following:

1. Add a partner data source in SiteWise Edge
2. Go to the partner's web portal, where applicable, and configure the partner data source so it connects to the SiteWise Edge gateway.

Topics

- Security
- Set up Docker on your SiteWise Edge gateway
- Add a partner data source in SiteWise Edge
- SiteWise Edge gateway partner data source options

Security

As part of the Shared Responsibility Model between AWS, our customers, and our partners the following describes who is responsible for the different aspects of security:

Customer responsibility

- Vetting the partner.
- Configuring the network access given to the partner.
- Monitoring for reasonable usage of the SiteWise Edge gateway machine resources (CPU, memory, and file system).

AWS responsibility

- Isolating the partner from the customer AWS cloud resources except those needed by the partner. In this case, AWS IoT SiteWise ingestion.
- Restricting the partner solution to a reasonable usage of the SiteWise Edge gateway machine resources (CPU and memory).

Partner responsibility

- Using secure defaults.
- Keeping the solution secure over time through patches and other appropriate updates.
- Keeping customer data confidential.

Set up Docker on your SiteWise Edge gateway

AWS IoT SiteWise provides a Docker image that allows you to run the SiteWise Edge application on various platforms and environments. This Docker image encapsulates all the necessary components and dependencies required to collect, process, and send data from your industrial equipment to the AWS Cloud. By using the Docker image, you can deploy and run the SiteWise Edge application on Docker-compatible hosts, such as servers, edge devices, or cloud-based container services.

To add a partner data source, [Docker Engine](https://docs.docker.com/engine/) 1.9.1 or later must be installed on your local device.

> **Note**
>
> Version 20.10 is the latest version that is verified to work with the SiteWise Edge gateway software.

Verify Docker is installed

To verify Docker is installed, run the following command from a terminal connected to your SiteWise Edge gateway:

```
docker info
```

If the command returns a `docker is not recognized` result, or an older version of Docker is installed, [Install Docker Engine](https://docs.docker.com/engine/install/) before continuing.

Set up Docker

The system user that runs a Docker container component must have root or administrator permissions, or you must configure Docker to run it as a non-root or non-admistrator user.

On Linux devices, you must add a `ggc_user` user to the `docker` group to call Docker commands without `sudo`.

To add `ggc_user`, or the non-root user that you use to run Docker container components, to the `docker` group, run the following command:

```
sudo usermod -aG docker ggc_user
```

For more information, see [Linux post-installation steps for DockerEngine](#).

Add a partner data source in SiteWise Edge

To connect a partner data source to your SiteWise Edge gateway, add it as a data source. When you add it as a data source, AWS IoT SiteWise will deploy a private AWS IoT Greengrass component to your SiteWise Edge gateway.

Prerequisites

To add a partner data source, you must do the following:

- For EasyEdge and CloudRail, create an account with the partner, then bind the accounts.
- Set up Docker on your SiteWise Edge gateway

Create a SiteWise Edge gateway with a partner data source

If you want to create a new SiteWise Edge gateway, complete the steps in Create a self-hosted SiteWise Edge gateway. After you've created SiteWise Edge gateway follow the steps in Add a partner data source to an existing SiteWise Edge gateway to add a partner data source.

Add a partner data source to an existing SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Choose the SiteWise Edge gateway you want to connect the partner data source to.

4. Under **Data sources**, choose **Add data source**.
5. On the **Add data source** screen, choose a **Source type**, to select the partner that connects your SiteWise Edge gateway. Each data source has its own configuration options. There are two categories of data sources: AWS sources and Partner sources.

   Using a partner data source, you can select one source per gateway. For a list of data source partner integration options, see SiteWise Edge gateway partner data source options. Note that you can add up to 100 OPC UA data sources (AWS sources). To get started with OPC UA data sources, see OPC UA data sources for AWS IoT SiteWise Edge gateways.
6. Enter a name for the source.
7. Select your data source's tab below and follow the configuration procedure.

   CloudRail

   Much of the CloudRail configuration is done in the CloudRail portal after saving the data source for your SiteWise Edge gateway. However, authorizing the connection is required.

   > **Note**
   >
   > The CloudRail connection is only available on Linux.

   1. Create a CloudRail account to get started with connecting to AWS IoT SiteWise.
   2. Ensure that Docker is installed on your gateway. For more information, see Set up Docker on your SiteWise Edge gateway.
   3. Read the **Authorize access and deployment** agreement, then choose **Authorize**. Checking the box grants the AWS partner access to your data source and allows AWS to deploy on the partner's component.

   > **Note**
   >
   > The **Measurement Prefix – *optional*** is set within your CloudRail portal.

> **Note**
>
> Partner software is developed, maintained, and supported by the AWS partner. AWS is not responsible for the interface, configuration, or software.

For more information, see CloudRail.

EasyEdge

Much of the EasyEdge configuration is done in the EasyEdge portal after saving the data source for your SiteWise Edge gateway. However, authorizing the connection is required.

> **Note**
>
> The EasyEdge connection is only available on Linux.

1. Create an EasyEdge account to get started with connecting to AWS IoT SiteWise.
2. Ensure that Docker is installed on your gateway. For more information, see Set up Docker on your SiteWise Edge gateway.
3. Read the **Authorize access and deployment** agreement, then choose **Authorize**. Checking the box grants the AWS partner access to your data source and allows AWS to deploy on the partner's component.

> **Note**
>
> The **Measurement Prefix – *optional*** is set within your EasyEdge portal.

> **Note**
>
> Partner software is developed, maintained, and supported by the AWS partner. AWS is not responsible for the interface, configuration, or software.

For more information, see EasyEdge.

Litmus Edge

You can activate the Litmus configuration in two ways. Activate Litmus Edge directly through AWS IoT SiteWise using information from the Litmus Edge Manager portal. Or, you can manually activate Litmus Edge for AWS IoT SiteWise through Litmus Edge Manager.

> **Note**
>
> The Litmus Edge connection is only available on Linux.

To activate using a Litmus Edge activation code on AWS IoT SiteWise

Use this procedure when adding a Litmus Edge data source with a Litmus Edge activation code on the AWS IoT SiteWise console.

1. Select **Activate now using a code**. Additional configuration options appear.
2. Enter the Litmus Edge Manager to connect Litmus Edge to your SiteWise Edge gateway. For more information, see Step 3a: Set Data and Device Management Endpoint in the Litmus Edge Manager documentation.
3. Provide the Litmus Edge Manager activation code to activate Litmus Edge on AWS IoT SiteWise
4. Optionally, provide AWS IoT SiteWise with the **Litmus Edge Manager CA certificate**. The certificate prevents Litmus Edge from activating on an unauthorized Litmus Edge Manager.
5. Ensure that Docker is installed on your gateway. For more information, see Set up Docker on your SiteWise Edge gateway.

   > **Note**
   >
   > AWS IoT SiteWise deploys the partner application as a Docker container. The application is deployed with `NET_ADMIN` capability so that the Litmus Edge Docker container can be managed through Litmus Edge Manager. Litmus Edge

> requires this privileged access to run on your devices. For more information about the Litmus Edge Docker requirements, see Docker Installation in the *QuickStart Guide* in the Litmus Edge documentation.

6. Read the **Authorize access and deployment** agreement, then choose **Authorize**. Checking the box grants the AWS partner access to your data source and allows AWS to deploy on the partner's component.

To activate manually through Litmus Edge

1. Select **Activate later on Litmus Edge**.
2. Ensure that Docker is installed on your gateway. For more information, see Set up Docker on your SiteWise Edge gateway.

   > **Note**
   >
   > AWS IoT SiteWise deploys the partner application as a Docker container. The application is deployed with `NET_ADMIN` capability so that the Litmus Edge Docker container can be managed through Litmus Edge Manager. Litmus Edge requires this privileged access to run on your devices. For more information about the Litmus Edge Docker requirements, see Docker Installation in the *QuickStart Guide* in the Litmus Edge documentation.

3. Read the **Authorize access and deployment** agreement, then choose **Authorize**. Checking the box grants the AWS partner access to your data source and allows AWS to deploy on the partner's component.
4. After the deployment is complete, follow the Access the Litmus Edge Web UI instructions in the Litmus Edge *QuickStart Guide* documentation.

> **Note**
>
> Partner software is developed, maintained, and supported by the AWS partner. AWS is not responsible for the interface, configuration, or software.

For more information, see Litmus Edge.

8. Choose **Save**.

SiteWise Edge gateway partner data source options

AWS IoT SiteWise allows you to connect and ingest data from various partner data sources, such as industrial equipment, sensors, and other third-party systems. To connect a partner data source, you need to follow a few steps, including configuring the data source to send data to AWS IoT SiteWise, setting up the necessary permissions and authentication, and mapping the data to your asset models. This process ensures that your partner data is seamlessly integrated into your AWS IoT SiteWise environment, enabling you to monitor and analyze it alongside your other data sources.

This section lists the available partners for third-party data source integration on SiteWise Edge gateways. Use the information below to configure a partner data source.

> **Note**
>
> You can add one data source for each partner in each gateway

CloudRail

Portal:

https://devices.cloudrail.com/

Requirements

For more information on CloudRail requirements, see FAQS on the CloudRail website.

CloudRail documentation:

Edge Computing: SiteWise Edge

EasyEdge

Portal:

https://studio.easyedge.io/

Requirements

EasyEdge requirements – Information about EasyEdge requirements, including endpoints and ports required for configuring the firewall. **Note**: You'll need an EasyEdge account to access this documentation.

EasyEdge documentation:

EasyEdge for AWS

Litmus Edge

Access to Litmus Edge Manager:

To access Litmus Edge, set up a Litmus Edge Manager account.

Requirements

Litmus Edge Requirements – Recommended configurations and system requirements to deploy Litmus Edge.

Litmus documentation:

- Integration to AWS IoT SiteWise
- Litmus Edge Documentation

AWS IoT Greengrass components for AWS IoT SiteWise Edge

SiteWise Edge uses AWS IoT Greengrass components to collect, process, and transmit industrial data at the edge. These components work together to enable local data processing and seamless integration with the AWS IoT SiteWise cloud service.

IoT SiteWise publisher

The IoT SiteWise publisher component (`aws.iot.SiteWiseEdgePublisher`)is responsible for:

- Securely transmitting collected data to the AWS IoT SiteWise cloud service
- Managing data buffering and retries during connectivity issues

For more information on configuring the publisher for SiteWise Edge, see Configure the AWS IoT SiteWise publisher component. And, for more information on the publisher component, see IoT SiteWise publisher in the *AWS IoT Greengrass Version 2 Developer Guide*.

IoT SiteWise processor

The IoT SiteWise processor component (`aws.iot.SiteWiseEdgeProcessor`) performs the following tasks:

- Executing data transformations and calculations at the edge
- Implementing asset property definitions and computations locally
- Reducing data volume by aggregating or filtering data before transmission

For more information about the processor component, see IoT SiteWise processor in the *AWS IoT Greengrass Version 2 Developer Guide*.

IoT SiteWise OPC UA collector

The IoT SiteWise OPC UA collector (`aws.iot.SiteWiseEdgeCollectorOpcua`) component is designed to:

- Connect to OPC UA servers in industrial environments
- Collect data from OPC UA data sources efficiently
- Transform OPC UA data into a format compatible with AWS IoT SiteWise

For more information about OPC UA collector component, see IoT SiteWise OPC UA collector in the *AWS IoT Greengrass Version 2 Developer Guide*.

IoT SiteWise OPC UA data source simulator

The IoT SiteWise OPC UA data source simulator component (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) provides the following functionality:

- Starts a local OPC UA server that generates sample data
- Simulates a data source that can be read by the AWS IoT SiteWise OPC UA collector component on an AWS IoT SiteWise gateway
- Enables exploration of AWS IoT SiteWise features using the generated sample data

This component is particularly useful for testing and development purposes, allowing you to simulate industrial data sources without the need for physical equipment.

For more information about the data source simulation component, see IoT SiteWise OPC UA data source simulator in the *AWS IoT Greengrass Version 2 Developer Guide*.

These AWS IoT Greengrass components work to enable SiteWise Edge functionality. The IoT SiteWise publisher ensures data is reliably sent to the cloud, the IoT SiteWise processor handles local computations and data optimization, and the IoT SiteWise OPC UA collector facilitates integration with common industrial protocols.

> **Note**
>
> To use these components, you must have AWS IoT Greengrass V2 or later installed on your edge devices. Proper configuration of each component is important for optimal performance of SiteWise Edge.

Filter assets on a SiteWise Edge gateway

You can use edge filtering to more efficiently manage your assets by sending only a subset of assets to a specific SiteWise Edge gateway for use in data processing. If your assets are arranged in a tree, or parent-child, structure, you can set up an IAM policy attached to a SiteWise Edge gateway's IAM role that only allows the root of the tree, or parent, and its children to be sent to a specific SiteWise Edge gateway.

> **Note**
>
> If you're arranging existing assets into a tree structure, after you've created the structure, go into each existing asset that you added to the structure and choose **Edit** and then choose **Save** to make sure AWS IoT SiteWise recognizes the new structure.

Set up edge filtering

Set up edge filtering on your SiteWise Edge gateway by adding the following IAM policy to the SiteWise Edge gateway's IAM role, replacing *<root-asset-id>* with the ID of the root asset you want to send to the SiteWise Edge gateway.

JSON

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

```
                "Effect": "Deny",
                "Action": [
                    "iotsitewise:DescribeAsset",
                    "iotsitewise:ListAssociatedAssets"
                ],
                "Resource": "arn:aws:iotsitewise:*:*:asset/*",
                "Condition": {
                    "StringNotLike": {
                        "iotsitewise:assetHierarchyPath": "/<root-asset-id>*"
                    }
                }
            }
        ]
}
```

If there are assets currently on your SiteWise Edge gateway that you'd like to remove, log into your SiteWise Edge gateway and run the following command to force the SiteWise Edge gateway to sync with AWS IoT SiteWise by deleting the cache.

```
sudo rm /greengrass/v2/work/aws.iot.SiteWiseEdgeProcessor/sync-app/
sync_resource_bundles/edge.json
```

Configure proxy support and manage trust stores for AWS IoT SiteWise Edge

In AWS IoT SiteWise Edge, configure and manage trust stores to set up proxy support for your edge devices. First, set up proxy configuration, then configure trust stores. You can configure trust stores either during gateway installation or manually after your gateway is established.

- **Proxies –** Facilitate connectivity between your edge devices and AWS services in various network environments.
- **Trust stores –** Ensure secure connections by managing trusted certificates. Proper configurations help you comply with your network security policies, enable communication in restricted network environments, and optimize data transfer between edge devices and cloud services.

SiteWise Edge utilizes multiple trust stores for different component types, ensuring secure and efficient data flow from your edge devices to the cloud. You can configure trust stores and proxies on an existing gateway or during the installation process when creating a new gateway.

Requirements for trust store and proxy configurations

Before you configure a trust store or install SiteWise Edge with proxy settings, ensure that you meet the prerequisites. There are varied implementation requirements based on your component usage and functionality requirements.

Proxy support requirements

- The URL of your proxy server. The URL should include the user info, the port number for the host. For example, `scheme://[userinfo@]host[:port]`.
  - `scheme` – Must be HTTP or HTTPS
  - (Optional) `userinfo` – User name and password information
  - `host` – The host name or IP address of the proxy server
  - `port` – The port number
- A list of addresses to bypass the proxy.
- (Optional) The proxy CA certificate file if you're using an HTTPS proxy with a self-signed certificate.

Trust store requirements

- For full data processing pack functionality with HTTPS proxy, you should update all three trust stores.
- If you only use the IoT SiteWise OPC UA collector and IoT SiteWise publisher, update the certificates AWS IoT Greengrass Core and Java trust stores to the latest version.

Best practices for trust store and proxy server edge configurations

For ongoing maintenance and to maintain the highest level of security in your edge environment:

- Regularly review and update proxy settings to align with your network security requirements.
- Monitor gateway connectivity and data flow to ensure proper proxy communication
- Maintain and update trust stores according to your organization's certificate management policies
- You can implement and follow our recommended best practices for secure communication in edge environments, such as:
- Document your proxy and trust store configurations for operational visibility

- Follow your organization's security practices for credential management

These practices help maintain secure and reliable operations for your SiteWise Edge gateways while remaining aligned with your broader security policies.

Configure proxy settings during AWS IoT SiteWise Edge gateway installation

You can configure AWS IoT SiteWise Edge to work with a proxy server during gateway installation. The installation script supports both HTTP and HTTPS proxies and can automatically configure trust stores for secure proxy connections.

When you run the installation script with proxy settings, it performs several important tasks:

- Validates the proxy URL format and parameters to ensure they are correctly specified.
- Downloads and installs required dependencies through the configured proxy.
- If a proxy CA certificate is provided, it's appended to the AWS IoT Greengrass root CA certificate and imported into the Java KeyStore.
- Configures AWS IoT Greengrass (which SiteWise Edge uses) to use the proxy for all outbound connections.
- Completes the SiteWise Edge installation with the appropriate proxy and trust store configurations.

To configure proxy settings when installing gateway software

1. Create a SiteWise Edge gateway. For more information, see Create a self-hosted SiteWise Edge gateway and Install the AWS IoT SiteWise Edge gateway software on your local device.
2. Run the installation script with the appropriate proxy settings for your environment. Replace the placeholders with your specific proxy information

   Replace each of the following items:

   - `-p`, `--proxy-url` – The URL of the proxy server. The URL must be either `http` or `https`.
   - `-n`, `--no-proxy` – A comma-separated list of addresses to bypass the proxy.
   - (Optional)`-c`, `--proxy-ca-cert` – Path to the proxy CA certificate file.
   - (Optional)`-j`, `--javastorepass` – The Java KeyStore password. The default password is `changeit`.

Linux

For Linux systems, use the following command structure:

```
sudo ./install.sh -p proxy-url -n no-proxy-addresses [-c proxy-ca-cert-path] [-
j javastorepass]
```

Windows

For Microsoft Windows systems using PowerShell, use this command structure:

```
.\install.ps1 -ProxyUrl proxy-url -NoProxyAddresses no-proxy-addresses [-
ProxyCaCertPath proxy-ca-cert-path] [-JavaStorePass javastorepass]
```

Troubleshooting during proxy-enabled installation

For more information on resolving trust store issues related to a SiteWise Edge gateway, see Proxy-enabled installation issues.

Manually configure trust stores for HTTPS proxy support in AWS IoT SiteWise Edge

When configuring AWS IoT SiteWise Edge components to connect through an HTTPS proxy, add the proxy server's certificate to the appropriate trust stores. SiteWise Edge uses multiple trust stores to secure communications. There are three trust stores and your use of them depends upon the SiteWise Edge component type in your gateway implementation.

Trust stores are automatically updated during the installation process when proxy settings are provided.

- Configure an AWS IoT Greengrass Core component trust store – The AWS IoT Greengrass root CA certificate is included in the trust stores to verify the authenticity of AWS services.

  This trust store helps AWS IoT Greengrass components securely communicate with AWS services through the proxy while verifying the authenticity of those services.
- Configure a Java-based component trust store – The Java KeyStore (JKS) is the main trust store used by Java-based components for SSL/TLS connections.

Java applications rely on the JKS to establish secure connections. For example, if you're using the IoT SiteWise publisher or IoT SiteWise OPC UA collector, which are Java-based, you'll need to configure this trust store. This ensures these components can securely communicate through the HTTPS proxy when sending data to the cloud or collecting data from OPC UA servers.

- System-level component trust store configuration – When using HTTPS proxies, their certificates must be added to the appropriate trust stores to enable secure connections.

  When using HTTPS proxies, their certificates must be added to the appropriate trust stores to enable secure connections. This is necessary because system-level components, often written in languages like Rust or Go, rely on the system's trust store rather than Java's JKS. For example, if you're using system utilities that need to communicate through the proxy (like for software updates or time synchronization), you'll need to configure the system-level trust store. This ensures these components and utilities can establish secure connections through the proxy.

Configure an AWS IoT Greengrass Core component trust store

For AWS IoT Greengrass Core functions that use Amazon's root CA:

1. Locate the certificate file at `/greengrass/v2/AmazonRootCA1.pem`
2. Append the HTTPS proxy root certificate (self-signed) to this file.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
```

```
-----END CERTIFICATE-----
```

Configure HTTPS proxy on an established gateway

You can add proxy support to an established gateway by connecting to port 443 instead of port 8883. For more information on using a proxy server, see Connect on port 443 or through a network proxy in the *AWS IoT Greengrass Version 2 Developer Guide*. If you create a new gateway, you can set the proxy configuration during gateway installation. For more information, see Configure proxy settings during AWS IoT SiteWise Edge gateway installation.

When you use an HTTPS proxy with AWS IoT Greengrass on SiteWise Edge, the software automatically chooses between HTTP and HTTPS for proxy connections based on the provided URL.

> **Important**
>
> Update all required trust stores before attempting to connect through an HTTPS proxy.

Configure a Java-based component trust store

For IoT SiteWise publisher, IoT SiteWise OPC UA collector, and Java services in the data processing pack, the default Java trust store location is `$JAVA_HOME/jre/lib/security/cacerts`

To add a certificate

1. Create a file to store the proxy server's certificate, such as `proxy.crt`.

   > **Note**
   >
   > Create the file ahead of time using the proxy server's certificate.

2. Add the file to Java's trust store using the following command:

   ```
   sudo keytool -import -alias proxyCert -keystore /usr/lib/jvm/java-11-openjdk-amd64/
   lib/security/cacerts -file proxy.crt
   ```

3. When prompted, use the default password: `changeit`

System-level component trust store configuration

For components written in Rust, Go, and other languages that use the system trust store:

Linux

Linux systems: Add certificates to `/etc/ssl/certs/ca-certificates.crt`

Windows

Microsoft Windows systems: To configure the trust store, follow the Certificate Store procedure in the *Microsoft Ignite* documentation.

Windows offers multiple certificate stores, including separate stores for User and Computer scopes, each with several sub-stores. For most SiteWise Edge setups, we recommend adding certificates to the `COMPUTER | Trusted Root Certification Authorities` store. However, depending on your specific configuration and security requirements, you might need to use a different store.

Troubleshooting trust store issues

For more information on resolving trust store issues related to a SiteWise Edge gateway, see Trust store issues.

Use AWS IoT SiteWise APIs on the edge

AWS IoT SiteWise provides a subset of its APIs, along with edge-specific APIs, enabling seamless interaction with asset models and their associated assets deployed at the edge. These asset models must be configured to run on the edge. For more information, see Configure an asset model for data processing on SiteWise Edge for detailed instructions on this setup process.

After you configure these APIs, you can retrieve comprehensive data about your asset models and individual assets. Retrieving asset model, asset, dashboard, portal and project information can help you monitor deployed portals and dashboards, and access asset data collected at the edge level. This provides a central host in your network for interactions with AWS IoT SiteWise without requiring a web API call.

Topics

- Edge-only APIs for use with AWS IoT SiteWise edge devices
- Enable CORS on AWS IoT SiteWise Edge APIs
- Configure session timeouts for AWS IoT SiteWise Edge
- Tutorial: List asset models on an AWS IoT SiteWise Edge gateway

All available AWS IoT SiteWise Edge device APIs

AWS IoT SiteWise provides a variety of APIs to use on edge devices so that you can complete tasks locally on the device. Some of the available edge APIs include retrieving asset models, creating and updating asset properties, and sending data streams to the cloud. By leveraging these APIs, you can build solutions that can operate in environments with intermittent or limited network connectivity.

Available AWS IoT SiteWise APIs

The following AWS IoT SiteWise APIs are available on edge devices:

- ListAssetModels
- DescribeAssetModel
- ListAssets
- DescribeAsset
- DescribeAssetProperty
- ListAssociatedAssets
- GetAssetPropertyAggregates
- GetAssetPropertyValue
- GetAssetPropertyValueHistory
- ListDashboards
- ListPortals
- ListProjectAssets
- ListProjects
- DescribeDashboard
- DescribePortal

- DescribeProject

Available edge-only APIs

The following APIs are used locally on devices on the edge:

- Authenticate – Use this API to get the SigV4 temporary credentials that you'll use to make API calls.

Edge-only APIs for use with AWS IoT SiteWise edge devices

In addition to the AWS IoT SiteWise APIs that are available on the edge, there are edge-specific ones. Those edge-specifc APIs are described below.

Authenticate

Gets the credentials from the SiteWise Edge gateway. You'll need to add local users or connect to your system using LDAP or a Linux user pool. For more information about adding users, see LDAP or Linux user pool.

Request syntax

```
POST /authenticate HTTP/1.1
Content-type: application/json
{
  "username": "string",
  "password": "string",
  "authMechanism": "string"
}
```

URI request Parameters

The request does not use any URI parameters.

Request body

The request accepts the following data in JSON format.

username

The username used to validate the request call.

Type: String

Required: Yes

password

The password of the user requesting credentials.

Type: String

Required: Yes

authMechanism

The authentication method to validate this user in the host.

Type: String

Valid values: `ldap`, `linux`, `winnt`

Required: Yes

Response syntax

```
HTTP/1.1 200
Content-type: application/json
{
  "accessKeyId": "string",
  "secretAccessKey": "string",
  "sessionToken": "string",
  "region": "edge"
}
```

Response elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format.

accessKeyId

The access key ID that identifies the temporary security credentials.

Length Constraints: Minimum length of 16. Maximum length of 128.

Pattern: `[\w]*`

secretAccessKey

The secret access key that can be used to sign requests.

Type: String

sessionToken

The token that users must pass to the service API to use the temporary credentials.

Type: String

region

The region you are targeting for API calls.

Type: CONSTANT - `edge`

Errors

IllegalArgumentException

The request was rejected because the provided body document was malformed. The error message describes the specific error.

HTTP Status Code: 400

AccessDeniedException

The user doesn't have valid credentials based on the current Identity Provider. The error message describes the authentication Mechanism.

HTTP Status Code: 403

TooManyRequestsException

The request has reached it's limit of authentication attempts. The error message contains the quantity of time to wait until new attempts of authentication are made.

HTTP Status Code: 429

Enable CORS on AWS IoT SiteWise Edge APIs

Enabling CORS (Cross-Origin Resource Sharing) on AWS IoT SiteWise Edge APIs allows web applications to directly communicate with the APIs across different domains. This enables seamless integration, real-time data exchange, and cross-domain data access without intermediary servers or workarounds. CORS settings can be configured to specify allowable origins, ensuring controlled cross-origin access.

> **Note**
>
> CORS is available for version 3.3.1 and later of the This feature is available for version 3.3.1 and later of the `aws.iot.SiteWiseEdgeProcessor` component. For more information, see AWS IoT SiteWise processor in the *AWS IoT Greengrass Version 2 Developer Guide*.

To enable CORS on SiteWise Edge APIs

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the SiteWise Edge gateway for which you want to enable CORS. You can enable CORS on the AWS IoT Greengrass V2 deployment type.
4. In the **Gateway configuration** section, choose the associated **Greengrass core device**.
5. In the **Deployments** tab, under **Greengrass devices**, select the appropriate deployment link.
6. Under **Actions** choose **Revise**, then **Revise deployment**.

> **Important**
>
> Creating a revised CORS enabled configuration replaces the device's current configuration.

7. In **Step 1, Specify target**, provide an optional **Name** to identify the deployment.
8. In **Step 2, Select components - optional**, you can leave all current selections as-is and choose **Next**.
9. In **Step 3, Configure components - optional**, select **aws.iot.SiteWiseEdgeProcessor**, and choose **Configure component**.
10. In the Configuration update section, under Configuration to merge, enter the following JSON:

```
{
    "AWS_SITEWISE_EDGE_ACCESS_CONTROL_ALLOW_ORIGIN": "*"
}
```

> **Note**
>
> Using * as the value for `AWS_SITEWISE_EDGE_ACCESS_CONTROL_ALLOW_ORIGIN` allows all origins. For production environments, it's recommended to specify exact origin URLs for better security.

11. Choose **Confirm**.
12. Choose **Next** to proceed through remaining steps until you arrive at **Step5, Review**.
13. Review your configuration changes, then choose **Deploy** to apply the changes to your SiteWise Edge gateway.

> **Note**
>
> Alternatively, you can enable CORS by setting global the environmental variable `AWS_SITEWISE_EDGE_ACCESS_CONTROL_ALLOW_ORIGIN` to * on your AWS IoT SiteWise gateway.

> **Note**
>
> For authenticated proxy, `userinfo` must be included in the `url` field in the proxy configuration rather than as a separated `username` and `password` fields.

After the deployment is complete, CORS is enabled on your SiteWise Edge API, allowing specified origins to make cross-origin requests to the API.

Configure session timeouts for AWS IoT SiteWise Edge

SiteWise Edge allows you to configure session timeouts for the SiteWise Edge API. This feature enhances security by automatically terminating inactive sessions after a specified time-period. This section guides you through the process of configuring the session timeout using the AWS IoT SiteWise console.

> **Note**
>
> Session timeout configuration is available for version 3.4.0 and later of the `aws.iot.SiteWiseEdgeProcessor` component. For more information, see [AWS IoT SiteWise processor](#) in the *AWS IoT Greengrass Version 2 Developer Guide*.

To configure a session timeout for a SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the SiteWise Edge gateway where you want to configure the session timeout.

   > **Note**
   >
   > You can configure the session timeout on the AWS IoT Greengrass V2 deployment type.

4. In the **Gateway configuration** section, choose the associated **Greengrass core device**.
5. In the **Deployments** tab, under **Greengrass devices**, select the appropriate deployment link.
6. Under **Actions** choose **Revise**. Read the warning, and then choose **Revise deployment**.

   > **Important**
   >
   > Creating a revised session timeout configuration replaces the device's current configuration.

7. In **Step 1, Specify target**, provide an optional **Name** to identify the revised deployment, and then choose **Next**.
8. In **Step 2, Select components - optional**, you can leave all current selections as-is and choose **Next**.
9. In **Step 3, Configure components - optional**, select **aws.iot.SiteWiseEdgeProcessor**, and choose **Configure component**.
10. In the **Configuration update** section, under **Configuration to merge**, enter the following JSON:

```
{
    "AWS_SITEWISE_EDGE_SESSION_TIMEOUT_MINUTES": "240"
}
```

11. Set the value for `AWS_SITEWISE_EDGE_SESSION_TIMEOUT_MINUTES` in minutes. Session timeout values can be from 1 minute to 10080 minutes (7 days). The default value is 240 minutes (4 hours).
12. Choose **Confirm**.
13. Choose **Next** to proceed through remaining steps until you arrive at Step 5, **Review**.
14. Review your configuration changes, then choose **Deploy** to apply the changes to your SiteWise Edge gateway.

> **Note**
>
> Alternatively, you can configure the session timeout by setting the global environmental variable **AWS_SITEWISE_EDGE_SESSION_TIMEOUT_MINUTES** to your desired value (in minutes) on your SiteWise Edge gateway.

After the deployment is complete, the new session timeout configuration is applied to your SiteWise Edge API.

Tutorial: List asset models on an AWS IoT SiteWise Edge gateway

You can use a subset of the available AWS IoT SiteWise APIs along with edge-specific APIs to interact with asset models and their assets on the edge. This tutorial will walk you through getting temporary credentials to an AWS IoT SiteWise Edge gateway and getting a list of the asset models on the SiteWise Edge gateway.

Prerequisites

In the steps of this tutorial you can use a variety of tools. To use these tools, make sure you have the corresponding prerequisites installed.

To complete this tutorial, you need the following:

- A deployed and running AWS IoT SiteWise Edge self-hosted gateway requirements
- Access to your SiteWise Edge gateway in the same network over port 443.

- OpenSSL installed
- (AWS OpsHub for AWS IoT SiteWise) The AWS OpsHub for AWS IoT SiteWise application
- (curl) curl installed
- (Python) urllib3 installed
- (Python) Python3 installed
- (Python) Boto3 installed
- (Python) BotoCore installed

Step 1: Get a SiteWise Edge gateway service signed certificate

To establish a TLS connection to the APIs available at the SiteWise Edge gateway, you need a trusted certificate. You can generate this certificate using a OpenSSL or AWS OpsHub for AWS IoT SiteWise.

OpenSSL

> **Note**
>
> You need OpenSSL installed to run this command.

Open a terminal and run the following command to get a signed certificate from the SiteWise Edge gateway. Replace `<sitewise_gateway_ip>` with the IP of the SiteWise Edge gateway.

```
openssl s_client -connect <sitewise_gateway_ip>:443 </dev/null 2>/dev/null | openssl
 x509 -outform PEM > GatewayCert.pem
```

AWS OpsHub for AWS IoT SiteWise

You can use AWS OpsHub for AWS IoT SiteWise. For more information, see Manage SiteWise Edge gateways.

The absolute path to the downloaded SiteWise Edge gateway certificate is used in this tutorial. Run the following command to export the complete path of your certificate, replacing `<absolute_path_to_certificate>` with the path to the certificate:

```
export PATH_TO_CERTIFICATE='<absolute_path_to_certificate>'
```

Step 2: Get your SiteWise Edge gateway hostname

> **Note**
>
> You need OpenSSL installed to run this command.

To complete the tutorial you'll need the hostname of your SiteWise Edge gateway. To get the hostname of your SiteWise Edge gateway, run the following, replacing `<sitewise_gateway_ip>` with the IP of the SiteWise Edge gateway:

```
openssl s_client -connect <sitewise_gateway_ip>:443 </dev/null 2>/dev/null | grep -Po
 'CN = \K.*'| head -1
```

Run the following command to export the hostname for use later, replacing `<your_edge_gateway_hostname>` with the hostname of your SiteWise Edge gateway:

```
export GATEWAY_HOSTNAME='<your_edge_gateway_hostname>'
```

Step 3: Get temporary credentials for your SiteWise Edge gateway

Now that you have the signed certificate and the hostname of your SiteWise Edge gateway, you need to get temporary credentials so you can run APIs on the gateway. You can get these credentials through AWS OpsHub for AWS IoT SiteWise or directly from the SiteWise Edge gateway using APIs.

> **Important**
>
> Credentials expire every 4 hours, so you should get the credentials just before using the APIs on your SiteWise Edge gateway. Don't cache credentials for longer than 4 hours.

Get temporary credentials using AWS OpsHub for AWS IoT SiteWise

> **Note**
>
> You need the AWS OpsHub for AWS IoT SiteWise application installed.

To use AWS OpsHub for AWS IoT SiteWise application to get your temporary credentials do the following:

1. Log into the application.
2. Choose **Settings**.
3. For **Authentication**, choose **Copy credentials**.
4. Expand the option that fits your environment and choose **Copy**.
5. Save the credentials for use later.

Get temporary credentials using the SiteWise Edge gateway API

To use the SiteWise Edge gateway API to get the temporary credentials you can use a Python script or curl, first you'll need to have a user name and password for your SiteWise Edge gateway. The SiteWise Edge gateways use SigV4 authentication and authorization. For more information about adding users, see LDAP or Linux user pool. These credentials will be used in the following steps to get the local credentials on your SiteWise Edge gateway that are needed to use the AWS IoT SiteWise APIs.

Python

> **Note**
>
> You need urllib3 and Python3 installed.

To get the credentials using Python

1. Create a file called **get_credentials.py** and the copy the following code into it.

```
'''
The following demonstrates how to get the credentials from the SiteWise Edge
 gateway. You will need to add local users or connect your system to LDAP/AD
https://docs.aws.amazon.com/iot-sitewise/latest/userguide/manage-gateways-
ggv2.html#create-user-pool

Example usage:
    python3 get_credentials.py -e https://<gateway_hostname> -c
 <path_to_certificate> -u '<gateway_username>' -p '<gateway_password>' -m
 '<method>'
```

```
'''
import urllib3
import json
import urllib.parse
import sys
import os
import getopt

"""
This function retrieves the AWS IoT SiteWise Edge gateway credentials.
"""
def get_credentials(endpoint,certificatePath, user, password, method):
    http = urllib3.PoolManager(cert_reqs='CERT_REQUIRED', ca_certs=
 certificatePath)
    encoded_body = json.dumps({
        "username": user,
        "password": password,
        "authMechanism": method,
    })

    url =  urllib.parse.urljoin(endpoint, "/authenticate")

    response = http.request('POST', url,
        headers={'Content-Type': 'application/json'},
        body=encoded_body)

    if response.status != 200:
        raise Exception(f'Failed to authenticate! Response status
 {response.status}')

    auth_data = json.loads(response.data.decode('utf-8'))

    accessKeyId = auth_data["accessKeyId"]
    secretAccessKey = auth_data["secretAccessKey"]
    sessionToken = auth_data["sessionToken"]
    region = "edge"

    return accessKeyId, secretAccessKey, sessionToken, region

def print_help():
    print('Usage:')
    print(f'{os.path.basename(__file__)} -e <endpoint> -c <path/to/certificate>
 -u <user> -p <password> -m <method> -a <alias>')
    print('')
```

```
    print('-e, --endpoint   edge gateway endpoint. Usually the Edge gateway
 hostname.')
    print('-c, --cert_path path to downloaded gateway certificate')
    print('-u, --user       Edge user')
    print('-p, --password   Edge password')
    print('-m, --method     (Optional) Authentication method (linux, winnt,
 ldap), default is linux')
    sys.exit()


def parse_args(argv):
    endpoint = ""
    certificatePath = None
    user = None
    password = None
    method = "linux"

    try:
        opts, args = getopt.getopt(argv, "he:c:u:p:m:",
 ["endpoint=","cert_path=", "user=", "password=", "method="])
    except getopt.GetoptError:
        print_help()

    for opt, arg in opts:
        if opt == '-h':
            print_help()
        elif opt in ("-e", "--endpoint"):
            endpoint = arg
        elif opt in ("-u", "--user"):
            user = arg
        elif opt in ("-p", "--password"):
            password = arg
        elif opt in ("-m", "--method"):
            method = arg.lower()
        elif opt in ("-c", "--cert_path"):
            certificatePath = arg

    if method not in ['ldap', 'linux', 'winnt']:
        print("not valid method parameter, required are ldap, linux, winnt")
        print_help()

    if (user == None or password == None):
        print("To authenticate against edge user, password have to be passed
 together, and the region has to be set to 'edge'")
```

```
        print_help()

    if(endpoint == ""):
        print("You must provide a valid and reachable gateway hostname")
        print_help()

    return endpoint,certificatePath, user, password, method


def main(argv):
    # get the command line args
    endpoint, certificatePath, user, password, method = parse_args(argv)

    accessKeyId, secretAccessKey, sessionToken, region=get_credentials(endpoint,
 certificatePath, user, password, method)

    print("Copy and paste the following credentials into the shell, they are
 valid for 4 hours:")
    print(f"export AWS_ACCESS_KEY_ID={accessKeyId}")
    print(f"export AWS_SECRET_ACCESS_KEY={secretAccessKey}")
    print(f"export AWS_SESSION_TOKEN={sessionToken}")
    print(f"export AWS_REGION={region}")
    print()




if __name__ == "__main__":
   main(sys.argv[1:])
```

2. Run **get_credentials.py** from the terminal replacing <gateway_username> and <gateway_password> with the credentials you created.

```
python3 get_credentials.py -e https://$GATEWAY_HOSTNAME -c $PATH_TO_CERTIFICATE
 -u '<gateway_username>' -p '<gateway_password>' -m 'linux'
```

curl

> **Note**
>
> You need curl installed.

To get the credentials using curl

1. Run the following command from the terminal replacing <gateway_username> and <gateway_password> with the credentials you created.

   ```
   curl --cacert $PATH_TO_CERTIFICATE --location \
   -X POST https://$GATEWAY_HOSTNAME:443/authenticate \
   --header 'Content-Type: application/json' \
   --data-raw '{
       "username": "<gateway_username>",
       "password": "<gateway_password>",
       "authMechanism": "linux"
   }'
   ```

   The response should look like the following:

   ```
   {
       "username": "sweuser",
       "accessKeyId": "<accessKeyId>",
       "secretAccessKey": "<secretAccessKey>",
       "sessionToken": "<sessionToken>",
       "sessionExpiryTime": "2022-11-17T04:51:40.927095Z",
       "authMechanism": "linux",
       "role": "edge-user"
   }
   ```

2. Run the following command from your terminal.

   ```
   export AWS_ACCESS_KEY_ID=<accessKeyId>
   export AWS_SECRET_ACCESS_KEY=<secretAccessKey>
   export AWS_SESSION_TOKEN=<sessionToken>
   export AWS_REGION=edge
   ```

Step 4: Get a list of the asset models on the SiteWise Edge gateway

Now that you have a signed certificate, your SiteWise Edge gateway hostname, and temporary credentials for your SiteWise Edge gateway, you can use the `ListAssetModels` API to get a list of the asset models on your SiteWise Edge gateway.

Python

> **Note**
>
> You need Python3, Boto3, and BotoCore installed.

To get the the list of asset models using Python

1. Create a file called **list_asset_model.py** and the copy the following code into it.

```
import json
import boto3
import botocore
import os

# create the client using the credentials
client = boto3.client("iotsitewise",
    endpoint_url= "https://"+ os.getenv("GATEWAY_HOSTNAME"),
    region_name=os.getenv("AWS_REGION"),
    aws_access_key_id=os.getenv("AWS_ACCESS_KEY_ID"),
    aws_secret_access_key=os.getenv("AWS_SECRET_ACCESS_KEY"),
    aws_session_token=os.getenv("AWS_SESSION_TOKEN"),
    verify=os.getenv("PATH_TO_CERTIFICATE"),
    config=botocore.config.Config(inject_host_prefix=False))

# call the api using local credentials
response = client.list_asset_models()
print(response)
```

2. Run **list_asset_model.py** from the terminal.

```
python3 list_asset_model.py
```

curl

> **Note**
>
> You need curl installed.

To get the list of asset models using curl

Run the following command from the terminal.

```
curl \
  --request GET https://$GATEWAY_HOSTNAME:443/asset-models \
  --cacert $PATH_TO_CERTIFICATE \
  --aws-sigv4 "aws:amz:edge:iotsitewise" \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  -H "x-amz-security-token:$AWS_SESSION_TOKEN"
```

The response should look like the following:

```
{
    "assetModelSummaries": [
        {
            "arn": "arn:aws:iotsitewise:{region}:{account-id}:asset-model/{asset-
model-id}",
            "creationDate": 1.669245291E9,
            "description": "This is a small example asset model",
            "id": "{asset-model-id}",
            "lastUpdateDate": 1.669249038E9,
            "name": "Some Metrics Model",
            "status": {
                "error": null,
                "state": "ACTIVE"
            }
        },
        .
        .
        .
    ],
    "nextToken": null
}
```

Host a SiteWise Edge gateway on Siemens Industrial Edge

Host your gateway on Siemens Industrial Edge using the AWS IoT SiteWise Edge application. Just as with AWS IoT Greengrass V2, you can optimize manufacturing processes or improve operational workflows using the SiteWise Edge on Siemens Industrial Edge.

You can ingest data from your Siemens Industrial Edge device to your AWS account by running a SiteWise Edge gateway on the device. To do this, request access to the AWS IoT SiteWise Edge application from the SiteWise Edge support team. Then, create a SiteWise Edge gateway resource with a deployment target of **Siemens Industrial Edge device - new**. Next, download the configuration file, and upload it to your application through the Siemens Industrial Edge Management portal. For more information about running applications on Siemens Industrial Edge, including how to set up the required Siemens resources, see What is Industrial Edge? in the Siemens documentation.

Note

Siemens is not a vendor or supplier for SiteWise Edge. The Siemens Industrial Edge Marketplace is an independent marketplace.

Topics

Security

As part of the Shared Responsibility Model between AWS, our customers, and our partners the following describes who is responsible for the different aspects of security:

Customer responsibility

- Vetting the partner.
- Configuring the network access given to the partner.
- Physically securing the device running SiteWise Edge.

AWS responsibility

- Isolating the partner from the customer AWS Cloud resources.

Partner responsibility

- Using secure defaults.
- Keeping the solution secure over time through patches and other appropriate updates.
- Keeping customer data confidential.
- Vetting other applications available in the partner marketplace.

Siemens Secure Storage and the AWS IoT SiteWise Edge application

To protect credentials and secrets required to run the AWS IoT SiteWise Edge application, Siemens Industrial Edge provides mechanisms to securely store the credentials on the device. The AWS IoT SiteWise Edge application won't run on a device if it doesn't have support for securely storing these credentials. Run failures caused by missing Secure Storage support are logged in log files.

The following minimum OS versions are required to install and run the AWS IoT SiteWise Edge application. Upgrade your devices to the latest versions to install the application.

- **For virtual devices:** IEVD version 1.19 or above
- **For physical devices:** IED-OS version 2.2 or above

The AWS IoT SiteWise Edge application on Siemens Industrial Edge will not run until you have upgraded your device.

Destinations for Siemens Industrial Edge devices

When using the AWS IoT SiteWise Edge application on Siemens Industrial Edge, destinations help prepare data before sending it to AWS IoT SiteWise for further analysis and distribution. You can configure data destination settings for buffered data ingestion using Amazon S3 or use real-time data ingestion. Both allow you to subscribe to MQTT topics using path filters on the Siemens Industrial Edge device deployment target.

The Siemens Industrial Edge deployment target on your gateway supports two primary data handling methods:

- **AWS IoT SiteWise real-time settings** - Data is sent directly to AWS IoT SiteWise as it's collected
- **AWS IoT SiteWise buffered using Amazon S3 settings** - Data is collected and stored temporarily in batches before being sent to Amazon S3

For more information about configuring these options, see Add an AWS IoT SiteWise buffered destination using Amazon S3 and Add an AWS IoT SiteWise Edge real-time destination.

Prefixes for path filters

Path filters for gateways using Siemens Industrial Edge deployment targets combine both the topic and data stream name to create a unique identifier for your data. The combined topic with data stream name is called a **prefix** in Siemens Industrial Edge gateways. This differs from self-hosted gateways where path filters are based solely on MQTT topics.

Example Path filter structure for Siemens data streams

A typical path filter for a Siemens data stream includes both the topic path and the data stream name:

```
ie/d/device1/application1/datastream1
```

Where:

- `ie/d/` is the required prefix for Siemens data streams
- `device1/application1` represents the hierarchical path
- `datastream1` is the specific data stream name

> **Note**
>
> When working with Siemens Industrial Edge data streams, ensure that you include both the metadata (`ie/m/`) and data (`ie/d/`) topics in your path filters to receive complete information about your data streams.

Destinations and path filters

View the following topics to learn more about destinations and path filters in MQTT-enabled gateways:

- Understand AWS IoT SiteWise Edge destinations
- Add an AWS IoT SiteWise Edge real-time destination
- Add an AWS IoT SiteWise buffered destination using Amazon S3
- Understand path filters for AWS IoT SiteWise Edge destinations
- Add path filters to AWS IoT SiteWise Edge destinations
- Manage AWS IoT SiteWise Edge destinations

Migrate from the preview application

If you ran SiteWise Edge on Siemens Industrial Edge during the preview phase, you'll need to upgrade from the preview version, version 1.0.1, to the latest version. Do the following to migrate:

1. Create new SiteWise Edge gateways. For more information, see Create a gateway for Siemens Industrial Edge.
2. Create a new Siemens Databus user for each new gateway. For more information, see Create a Siemens Databus user for the application.
3. Uninstall the version 1.0.1 AWS IoT SiteWise Edge gateway application on your IED.

> **Note**
>
> Prepare for interruptions to data flow as you reconfigure the AWS IoT SiteWise assets previously used by the preview version of the AWS IoT SiteWise Edge application. While the data history is preserved, there is potential for data loss while you reinstall the new gateway.

4. Delete the SiteWise Edge gateways you created during the preview in the AWS IoT SiteWise console.
5. Install the AWS IoT SiteWise Edge gateway application on IED using the new gateway configuration file. For more information, see Install the application onto a Siemens device.

> **Important**
>
> Installing the new gateway overwrites the preview version of the SiteWise Edge application. It isn't possible to go back to version 1.0.1 after installing version 2.0.0.

After configuring the new gateway and Siemens Databus user, your data flows to your properties.

You can also upgrade your SiteWise Edge application from version directly. However, a new gateway configuration is still necessary.

Troubleshooting

To troubleshoot the SiteWise Edge gateway on your Siemens Industrial Edge device, see Troubleshooting the AWS IoT SiteWise Edge application on Siemens Industrial Edge.

You can also access AWS re:Post to find answers to your questions.

AWS IoT SiteWise Edge application changelog

The following table describes the changes in each version of the AWS IoT SiteWise Edge application.

Version	Changes

3.0.0	New features
	
	• Adds additional configuration options in the AWS IoT SiteWise console for gateways installed on Siemens Industria l Edge:
	  ◦ Add and configure Amazon S3 destinations.
	  ◦ Remove the AWS IoT SiteWise real-time destination.
	  ◦ Add filters for each destination to control data routing.
	• Adds the option for a global prefix configuration to send all data from the IoT SiteWise publisher in the Siemens IE installer. The prefix is applied after filtering.

Version	Changes
	Bug fixes and improvements • Adds the ability to retry connections to the Siemens Industrial Edge Databus if AWS IoT SiteWise Edge applicati on's initial attempt fails.
2.0.1	Bug fixes and improvements • Fixes an issue where the app would enter failure state and quit if it was unable to retrieve AWS credentials on startup. Adds support to retry until successful credential retrieval.
2.0.0	• The AWS IoT SiteWise Edge application is now generally available. • Application requires Siemens IEVD version 1.19, or Siemens IED-OS version 2.2. • Performance improvements: Reduced memory and CPU usage. • Debugging improvements: You can now upload an optional config file to enable debug logs. • Security enhancements: The application uses SecureStorage API to securely store credentials on the device. • Docker digest value: `sha256:4a960f29234a190ebb5224c1fd0f3e99faafccc4cb3d93ca13fef247b6656d18`
1.0.1	Initial release

Requirements for the AWS IoT SiteWise Edge application

To run AWS IoT SiteWise Edge on Siemens Industrial Edge, you need the following:

- A Siemens Digital Exchange Platform account.
- A Siemens Industrial Edge Hub (iehub) account.
- A Siemens Industrial Edge Management instance.

  - The IE App Configuration Service. To learn more, see [Installing the IE App Configuration Service manually]() in the *Siemens Industrial Edge Management* documentation.
- Access to version 2.0.1 or higher of the AWS IoT SiteWise Edge application. For more information, see [Access the AWS IoT SiteWise Edge application]().
- Either a Siemens Industrial Edge Device (IED) or a Siemens Industrial Edge virtual Device (IEVD).
  - A minimum of 15 GB disk space for hardware requirements.
  - 1 GB of RAM with an additional 1 GB of swap memory.
  - Device configuration to allow outbound traffic on ports 443 and 8883.
  - A x86-64 bit processor.
  - Siemens Industrial Edge Management version 1.13.10 or higher.
  - Device conformance to Siemens Secure Storage requirements.
    - On virtual devices, IEVD version 1.19 or above.
    - On physical devices, IED-OS version 2.2 or above.
  - The latest version of Docker Compose.
  - Docker Engine version 18.091 or higher.
- Required domain access. For more information, see [AWS IoT SiteWise endpoints]().

Create a gateway for Siemens Industrial Edge

After you have the proper Siemens accounts and IEM instances, you can create a SiteWise Edge gateway of deployment type **Siemens Industrial Edge device**.

> **Note**
>
> Ensure that you meet all requirements for running a device on Siemens Industrial Edge Management. For more information, see [Requirements for the AWS IoT SiteWise Edge application]().

To create the configuration file

1. Navigate to the [AWS IoT SiteWise console]().
2. In the navigation pane, choose **Edge gateways**.
3. Choose **Create gateway**.

4. For **Deployment type**, choose **Siemens Industrial Edge device**.
5. Enter a name for your SiteWise Edge gateway or use the name generated by AWS IoT SiteWise.
6. (Optional) Under **advanced configuration**, do the following:

   - Enter a name for your AWS IoT Core Thing or use the name generated by AWS IoT SiteWise.
7. Choose **Create gateway**.
8. In the **Generate SiteWise Edge gateway configuration file** dialog box, choose **Generate and download**. AWS IoT SiteWise automatically generates a configuration file that you will use to configure the AWS IoT SiteWise Edge application.

   > **Important**
   >
   > You use the gateway configuration file to backup and restore your AWS IoT SiteWise Edge application. Save your SiteWise Edge gateway configuration file in AWS Secrets Manager to securely store and manage the file. Secrets Manager securely stores, manages, and retrieves sensitive information.

Create a Siemens Databus user for the application

AWS IoT SiteWise Edge on Siemens Industrial Edge ingests data from the Siemens Databus application. To connect SiteWise Edge to the Siemens Databus, you need a Siemens Databus user that provides access to the data you want to securely transfer to AWS IoT SiteWise. To start, create a Siemens Databus user and then provide the credentials to the SiteWise Edge application.

To create a Siemens Databus user

1. In your Siemens Industrial Edge Management instance, choose **Edge Management** in the **Platform Applications** section.
2. Choose the **Data connections** icon.
3. Select **Databus**. A list of your connected devices appears.
4. Select the device to connect to the AWS IoT SiteWise Edge application.
5. Choose **Launch**. The Databus Configurator for your selected device appears.
6. Create a user for your Edge device under **Users**. For more information on creating a user, see Users in the *Siemens Industrial Edge Management* documentation.

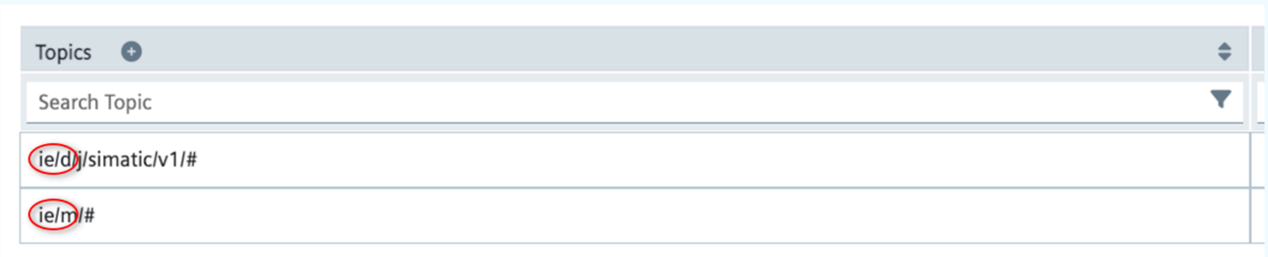
7. Select the topics for which this Siemens Databus should have access. These topics restrict what AWS IoT SiteWise Edge can access.

   Important

   All topics that a Siemens Databus user has access to are published to AWS IoT SiteWise.

   Note

   Siemens Databus users need access to both data and metadata topics. Topics that start with `ie/d` are data topics. And topics that start with `ie/m` are metadata topics. Share topics in pairs so that SiteWise Edge has access to both data and metadata for each respective topic.

8. Set appropriate permissions for your Siemens Databus configuration.

After creating your Siemens Databus configuration, you can install the AWS IoT SiteWise Edge application on your Siemens Industrial Edge Management. For more information, see Install the application onto a Siemens device.

You can also optionally configure destinations and path filters for your Siemens Industrial Edge gateway. For more information, see Destinations and path filters.

Access the AWS IoT SiteWise Edge application

To gain access to the AWS IoT SiteWise Edge application on Siemens Industrial Edge, send an email requesting access to the SiteWise Edge support team.

Include the following information in your email:

- Your name and contact information

- Company name
- Siemens Industrial Edge tenant ID

Install the application onto a Siemens device

After you gain access to the AWS IoT SiteWise Edge application by emailing the SiteWise Edge support team for Siemens Industrial Edge, assign the application to an instance of Siemens Industrial Edge Management. Then, you can install the AWS IoT SiteWise Edge application on your device.

To install the AWS IoT SiteWise Edge application

1. Verify that the Docker digest provided within Siemens Industrial Edge Management matches the latest version listed in the AWS IoT SiteWise Edge application changelog.

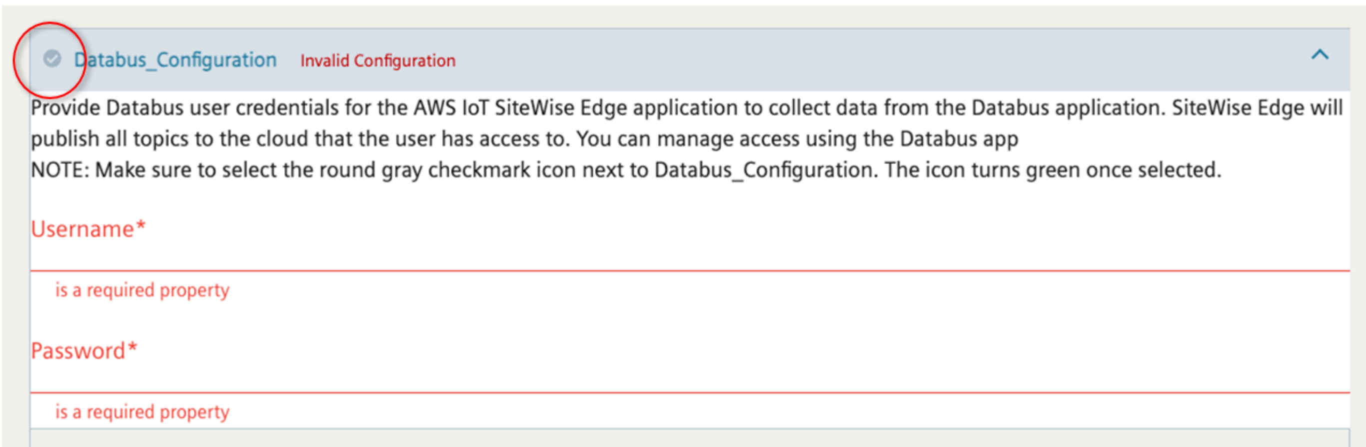
   For more information on locating the Docker digest value for Siemens, see the Managing an app in the *Siemens Industrial Edge Device* of the Siemens documentation.

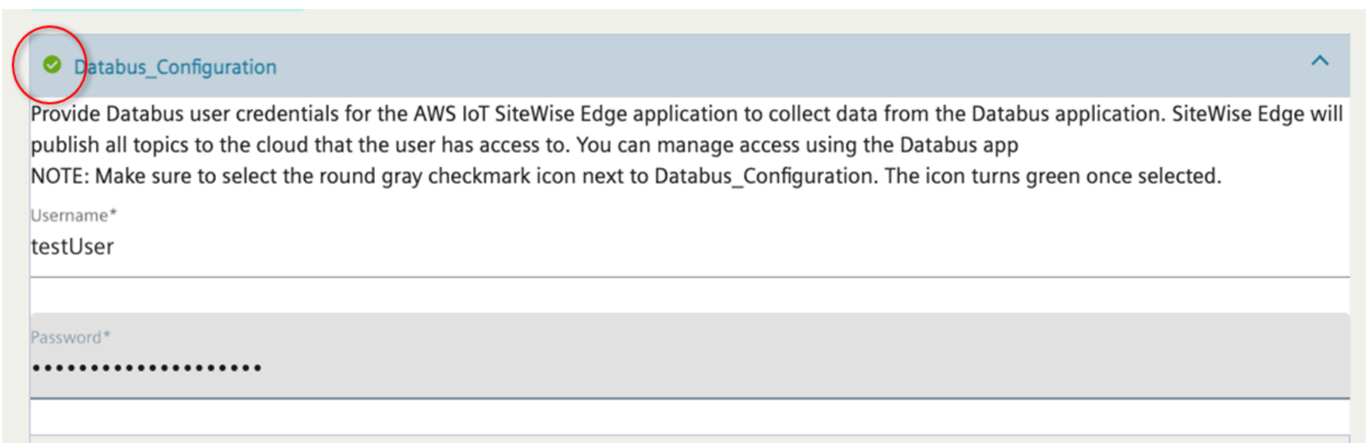
   Siemens Industrial Edge Management supports one version of the AWS IoT SiteWise Edge application at a time. Take this step to ensure that you're using the latest version of the application before installing the AWS IoT SiteWise Edge application on your Siemens Industrial Edge device.
2. Assign the **AWS IoT SiteWise Edge** application to Siemens Industrial Edge Management. For more information, see Managing an app in the *Industrial Edge Management* section of the Siemens documentation.
3. Within **Edge Management**, browse the catalog for the **AWS IoT SiteWise Edge** and choose it.
4. Choose **Install**.

   > **Note**
   >
   > If a **Contact Us** button displays, choose it, and follow the steps to request access to the AWS IoT SiteWise Edge application on Siemens Industrial Edge. For more information, see Access the AWS IoT SiteWise Edge application.
5. Select **Databus_Configuration** in the Schema Configurations options.
6. Enter the **Username** and **Password** for the Databus configuration. For more information on creating a Siemens Databus user, see Create a Siemens Databus user for the application.

7. Choose the small, round gray checkmark icon next to **Databus_Configuration** to turn the icon color green.

> **Note**
>
> The input configurations only apply if the checkmark icon changes from gray to green. Otherwise, the input configuration is ignored.





8. Choose **Next** to move onto **Other Configurations** where you can upload your gateway configuration file.
9. Choose **SiteWise_Edge_Gateway_Config** as the location to upload the gateway configuration file.

> **Note**
>
> Ensure that you choose **SiteWise_Edge_Gateway_Config** rather than **SiteWise_Edge_Support_Config_Optional**.

10. Select the device to install the application.
11. Choose **Install now**.

You can optionally configure the publisher component to export data to the AWS Cloud. For more information, see configure the AWS IoT SiteWise publisher component.

To configure destinations for your Siemens Industrial Edge gateway, see Destinations and path filters.

Update the AWS IoT SiteWise Edge application configuration

There are a few things to consider when updating an AWS IoT SiteWise Edge application configuration on **Siemens Industrial Edge**.

> **Note**
>
> Any change to the AWS IoT SiteWise Edge application configuration requires a restart of the application.

Reasons to restart the AWS IoT SiteWise Edge application

- A new Siemens Databus user for the AWS IoT SiteWise Edge application.
- A change to the gateway configuration file (your **SiteWise_Edge_Gateway_Config** file).
- A proxy configuration update (which also requires a full IEVD reboot)
- To enable debug logs for debugging issues

Restarting the application

1. In your Siemens Industrial Edge Management instance, choose **Edge Management** in the **Platform Applications** section.

2. Choose **My Installed Apps**.
3. Select the AWS IoT SiteWise Edge application.
4. Choose **Restart**.

Destinations and path filters

Destinations in AWS IoT SiteWise Edge provide a flexible and efficient way to manage how your industrial data flows from edge devices to the cloud. This section explains how to configure destinations, use path filters to route specific data streams, and choose the right destination type for your use case.

You can use destinations and path filters on self-hosted MQTT-enabled, V3 gateways and gateways used in conjunction with AWS IoT SiteWise Edge application hosted on Siemens Industrial Edge. Destinations and path filters do not work with Classic Streams, V2 gateways.

Topics

Understand AWS IoT SiteWise Edge destinations

Use AWS IoT SiteWise Edge destinations to determine where to send your source data. You can choose your data destination based on specific characteristics you need, like cost-effectiveness, low latency, or storage requirements. Integrate device data captured by AWS IoT SiteWise, our partners, or custom applications to publish and subscribe to path filters (topics) at the edge. You can then model, transfer, and store your device data in the cloud.

> **Note**
>
> For full use of all destination features on self-hosted gateways, upgrade to the latest versions of the IoT SiteWise publisher and IoT SiteWise OPC UA collector. Stream support is

continued on Classic streams, V2 gateways to maintain compatibility with existing setups. For more information, see Classic streams, V2 gateways for AWS IoT SiteWise Edge

Topics

How SiteWise Edge destinations enhance data management

Export data from the edge to AWS IoT SiteWise in real time, or in batches using Amazon S3.

Destinations enhance flexibility and scalability in your AWS IoT SiteWise environment. Destinations implement a centralized data management model, where sources publish data to a central system. Destinations determine where your data is sent using path filters. Destinations can subscribe to multiple path filters.

MQTT-enabled gateways, whether self-hosted or running on Siemens Industrial Edge, use MQTT for local communication and come with a default real-time destination which has filters set to #. This means that, by default, all messages on all topics are published to the AWS IoT SiteWise real-time destination. For more information, see Understand path filters for AWS IoT SiteWise Edge destinations. You can add one real-time destination in each gateway.

Destination types

When configuring a destination for your gateway, you have two main options: real-time configuration using AWS IoT SiteWise, and a buffered configuration using Amazon S3. Each destination type has its own set of settings and considerations.

AWS IoT SiteWise real-time settings

Choose this to send data directly to AWS IoT SiteWise hot-tier storage to facilitate ingesting and monitoring data in real-time. The real-time settings manage data flow, particularly when a gateway experiences connectivity issues with the cloud. During connection loss, data is temporarily stored locally on the gateway. Once the connection is re-established, the stored data is automatically sent to the cloud.

You can adjust various aspects of the data publishing process, such as the maximum amount of data to be stored locally, the rate at which data is sent to the cloud upon reconnection, and when to delete data after the storage reaches its capacity.

For more information on AWS IoT SiteWise storage tiers, see, Manage data storage in AWS IoT SiteWise.

AWS IoT SiteWise buffered using Amazon S3 settings

This destination type allows you to buffer data locally on the gateway and periodically send it to an Amazon S3 bucket in batches. The data is stored in the efficient Parquet format, which is optimized for analytical workloads. Once the data is in Amazon S3, you can import it into AWS IoT SiteWise for storage, processing, and analysis.

Choose this option to ingest data in batches, and store historical data in a cost-effective way. You can configure your preferred Amazon S3 bucket location, and the frequency at which you want data to be uploaded to Amazon S3. You can also choose what to do with the data after ingestion into AWS IoT SiteWise. You can choose to have the data available in both SiteWise and Amazon S3 or you can choose to delete it automatically from Amazon S3.

Compare destination functionality between gateway versions

The destinations feature in MQTT-enabled gateways streamlines data flow management. Destinations simplify data management through centralized configuration of data routing to various endpoints. This approach eliminates the need for complex individual stream setups, making the overall system more flexible and easier to manage.

By comparison, the Classic streams, V2 gateway, SiteWise Edge transmits data from data sources to publishers via AWS IoT Greengrass streams, configuring data destinations individually for each data source.

With the AWS IoT SiteWise destination feature, the publisher routing configuration is consolidated. Destination configuration allows you to manage destinations and path filters in a centralized manner. You can easily add a destination, manage path filters, delete unnecessary filters or destinations, depending on your needs.

Additionally, the destinations feature utilizes MQTT (Message Queuing Telemetry Transport), an industry-standard protocol widely used in industrial IoT applications. This adoption of MQTT helps AWS IoT SiteWise to facilitate easier integration with various devices and systems.

Destination limitations

Current limitations for destinations on SiteWise Edge gateways include:

- The data processing pack isn't supported on MQTT-enabled gateways.
- Data type support is limited to AWS IoT SiteWise data types. For information on enabling data type conversion, see Converting unsupported data types.

Use cases for SiteWise Edge destinations

SiteWise Edge destinations are utilized in diverse applications. Here are some key examples:

Industrial automation, *Real-time monitoring and predictive maintenance*

In industrial settings, sensors and devices on the factory floor can publish data to SiteWise Edge. Destinations can be configured to filter and route relevant data, enabling real-time monitoring and analysis of machine performance. You can subscribe to relevant MQTT topics using path filters, process the data, and then publish the processed data. In this way, you can selectively route processed data to AWS cloud analytic services or on-premises systems. Manufacturers can then implement predictive maintenance strategies, optimize production processes, and reduce downtime.

Smart buildings, *Energy efficiency and occupancy optimization*

Building automation systems generate data streams to monitor and control various aspects of a building, such as HVAC systems, lighting, and access control. With SiteWise Edge, these data streams can be ingested, processed, and routed to different destinations. Facility managers can configure destinations to filter and forward relevant data, enabling advanced capabilities like energy efficiency measures and occupancy optimization while ensuring data privacy and compliance.

These use cases demonstrate how the Destinations feature in SiteWise Edge can be leveraged across various industries to ingest, process, and route data efficiently. This enables advanced capabilities such as real-time monitoring, predictive maintenance, energy efficiency, and remote diagnostics while ensuring data privacy and compliance.

Understand path filters for AWS IoT SiteWise Edge destinations

Topics

Each destination is configured to route data to AWS IoT SiteWise or Amazon S3. Path filters allow you to select specific data to filter when receiving MQTT messages for a destination. Path filters represent the logical names of your data streams, acting as subscriptions to desired MQTT topics.

In MQTT, data is organized into topics, which are hierarchical strings separated by forward slashes (/). For example, a device might publish temperature data to the topic `home/livingroom/sensor1/temperature`. Here, `home/livingroom/sensor1` represents the path or logical name of the sensor, and `temperature` is the data type being published.

You can use path filters to subscribe to specific topics or a range of topics using wildcards (+ and #). The + wildcard matches a single level in the topic hierarchy. For example, `home/+/sensor1/temperature` would match `home/livingroom/sensor1/temperature` and `home/bedroom/sensor1/temperature`. The # wildcard, when used at the end of a filter, matches multiple levels.

You can also use a variety of characters typically not allowed in the MQTT specification within a path filter name. These characters don't function as wildcards when used within a name. AWS IoT SiteWise converts these characters using encoding to ensure MQTT compliance while preserving your original naming structure. This feature is particularly useful for accommodating existing naming conventions from other systems. For more information, see Special characters in path filter names.

By carefully selecting the appropriate path filters, you can control which data is sent to a specific destination. Tailor the data flow to your IoT system's requirements using path filters.

Path filter requirements

When entering path filters using the AWS IoT SiteWise console, keep the following in mind:

- Path filters are delimited by a new line, with each line representing a separate path filter.
- Individual path filters can have between 1 and 65,535 bytes.

- A path filter can't be blank.
- Null values (U+0000) are not allowed.
- You can enter up to 100 path filters or 65,535 characters at a time, whichever limit is reached first.
- The overall limit is 20,000 path filters for all the destinations on a gateway combined.
- You can use `%`, `#`, `+`, and `$` characters within path filter names, however AWS IoT SiteWise automatically converts them to URI encoding.

Best practices for path filters

When creating path filters for your AWS IoT SiteWise destinations, consider the following strategies to effectively manage your data.

- Structure your filters to mirror your device hierarchy. For example, in a manufacturing setting, `factory/+/machine/#`, captures data from all machines across different production lines.
- Use specific levels for device types, locations, or functions. For example, `factory/assembly-line/robot/temperature`. Or, in smart agriculture, `farm/+/crop/+/moisture`, to monitor moisture levels for various crops across different fields.
- Leverage wildcards strategically: Use + for variations at a single level and # to capture all subsequent levels. For example, `building/+/+/energy-consumption`, tracks energy usage across different zones and floors in a building. This assumes the first + captures all floors and the second + captures all zones.
- Balance specificity and flexibility by creating filters that are specific enough to capture relevant data but flexible enough to accommodate future changes. For example, `site/+/equipment-type/+/measurement` allows for addition of new sites or equipment types without changing the filter structure.

Test your filters thoroughly to ensure they capture the intended data and align with your IoT system's architecture and goals.

Path filters for OPC UA servers

For OPC UA servers, your path filters must correspond to the OPC UA tag names. The final level of your path filter must match the OPC UA tag name exactly. For example, if your OPC UA tag is `Device1.Temperature`, your path filter might be `factory/line1/Device1.Temperature`. You can use wildcards in the preceding levels, such as `factory/+/Device1.Temperature` to capture the tag across multiple production lines. If you have special characters within your path filter names, see Special characters in path filter names for more information.

Special characters in path filter names

AWS IoT SiteWise accommodates characters commonly used in industrial protocols like OPC UA, which are typically not allowed in standard MQTT topic names. This feature facilitates smoother integration of industrial systems with MQTT-based architectures.

> **Note**
> While our special character handling is helpful for integration and migration, it's recommended to align with standard MQTT naming conventions for new implementations when possible to ensure broader compatibility.

When receiving data from industrial sources, AWS IoT SiteWise normalizes topic names using URI encoding for special characters:

- `%` becomes `%25` (encoded first as the escape character)
- # becomes `%23`
- + becomes `%2B`
- $ becomes `%24` (only when at the start of a topic)

This encoding ensures that source data containing these special MQTT characters can be safely used as MQTT topic names while preserving the original industrial naming conventions.

Example : Special characters in path filter names

Here are examples of how industrial topic names might appear in AWS IoT SiteWise path filters:

- `Factory1/Line#2/Sensor+3` becomes `Factory1/Line%232/Sensor%2B3`
- `Plant%A/Unit$1/Temp` becomes `Plant%25A/Unit%241/Temp`
- `Site1/#Section/+Node` becomes `Site1/%23Section/%2BNode`

When creating subscriptions or viewing topic names in AWS IoT SiteWise, you'll see the original, unencoded versions. The encoding is handled automatically to ensure MQTT compliance.

Add an AWS IoT SiteWise Edge real-time destination

The real-time destination type enables you to stream IoT data directly from your devices and gateways into AWS IoT SiteWise storage in real-time. This option is ideal for use cases that require immediate ingestion and processing of data as it is generated, without the need for batching or buffering. You can only have one real-time destination configured in each gateway, as it streams data continuously to AWS IoT SiteWise.

> **Note**
>
> Duplicate TQVs may result in double charging.

To add a real-time destination

Use the AWS IoT SiteWise console or AWS CLI to add a real-time destination to your SiteWise Edge MQTT-enabled gateway.

Console

1. Open the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the gateway to which you want to add a destination.
4. In the **Destinations** section, choose **Add destination**.
5. On the **Add destination** page, enter **Destination details**:

   a. A name for your destination in the **Destination name** field.

   b. Select the **AWS IoT SiteWise real-time** for the **Destination type**.
6. Configure the gateway publishing order by setting the **Publishing order** to either **Publish older data first** or **Publish newest data first**. By default, the gateway publishes the oldest data first.
7. Use **Maximum batch wait time** to set a maximum time for the publisher to wait before sending a batch of data to AWS IoT SiteWise. This setting applies for each alias. The data is stored locally until either:

   - The set time has elapsed, or
   - 10 time-quality-value (TQV) entries are received for the alias

Whichever condition is met first triggers the batch to be sent to the cloud.

8. To compress uploaded data, select the **Activate compression when uploading data** check box. Letting the gateway compress your data prior to uploading it to the cloud reduces bandwidth usage.
9. To filter out expired publisher data, select the **Exclude expired data** check box. This selection only sends active and current data to AWS IoT SiteWise.
10. In the **Cutoff period** field, enter the frequency at which data should be considered expired within your dataset. You can determine if the data is counted in terms of minutes or days. The minimum cutoff period is five minutes. The maximum cutoff period is seven days.
11. Optionally configure the **Local storage settings**:

    a. Set the **Retention period** frequency – The amount of time the gateway locally stores data that is older than the cutoff period. The minimum retention period is one minute.

       The maximum retention period is 30 days and greater than or equal to the rotation period.

    b. Set the **Rotation period** – The time interval to specify when saving data that is older than the cutoff period for a single file. The gateway transfers one batch of data to the following local directory at the end of each rotation period: `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/exports`.

       The retention must be greater than one minute and equal to the retention period.

    c. Provide the **Storage capacity (GB)** value to set the maximum size of data stored locally in GB. If the data exceeds the determined maximum local storage size, the gateway starts deleting the oldest data first. The gateway continues to delete until the size of data stored locally is equal to or less than the quota.

       The storage capacity must be greater than or equal to one GB.

12. Add path filters to your destination. For more information see, Add path filters to AWS IoT SiteWise Edge destinations.

For more information, see Destination types.

AWS CLI

Example : Create a new AWS IoT SiteWise real-time destination

Use the UpdateGatewayCapabilityConfiguration API to configure the publisher.

Set the `capabilityNamespace` parameter to `iotsitewise:publisher:3`.

```
{
    "sources": [
        {
            "type": "MQTT"
        }
    ],
    "destinations": [
        {
            "type": "SITEWISE_REALTIME",
            "name": "your-destination-name",
            "config": {
                "publishingOrder": "TIME_ORDER",
                "enableCompression": true,
                "maxBatchWaitTime": "10s"
            },
            "filters": [
                {
                    "type": "PATH",
                    "config": {
                        "paths": [
                            "#"
                        ]
                    }
                }
            ]
        }
    ]
}
```

To update an existing AWS IoT SiteWise real-time destination, first use the `DescribeGatewayCapabilityConfiguration` API to find the `destinationId`.

Example : Update an AWS IoT SiteWise real-time destination

Use the UpdateGatewayCapabilityConfiguration API to configure the publisher.

Set the `capabilityNamespace` parameter to `iotsitewise:publisher:3`.

```
{
    "sources": [
        {
            "type": "MQTT"
        }
    ],
    "destinations": [
        {
            "id": "your-existing-destination-id",
            "type": "SITEWISE_REALTIME",
            "name": "your-destination-name",
            "config": {
                "publishingOrder": "TIME_ORDER",
                "enableCompression": true,
                "dropPolicy": {
                    "cutoffAge": "7d",
                    "exportPolicy": {
                        "retentionPeriod": "7d",
                        "rotationPeriod": "6h",
                        "exportSizeLimitGB": 10
                    }
                },
                "maxBatchWaitTime": "10s"
            },
            "filters": [
                {
                    "type": "PATH",
                    "config": {
                        "paths": [
                            "#"
                        ]
                    }
                }
            ]
        }
    ]
}
```

The following configuration options are specific to gateways using the `iotsitewise:publisher:3` namespace.

`sources`

Defines data sources to transfer of data from your industrial equipment to AWS IoT SiteWise. For MQTT-enabled gateways, use `MQTT`.

Type: Array of objects

Required: Yes

`destinations`

Defines where to send data. Destinations are either real-time or buffered using Amazon S3. At least one destination object is required, but you can add an empty array. You can have one real-time destination for each gateway. For more information, see [Understand AWS IoT SiteWise Edge destinations](#).

Type: Array of objects

Required: Yes

`id`

The unique identifier for the destination. You can either provide an existing destination ID or leave it blank. If you do not specify an ID then a UUID is generated by default.

Type: String

Required: No

`type`

Type of destination. Options include: `SITEWISE_REALTIME` and `SITEWISE_BUFFERED`.

- `SITEWISE_REALTIME` – Send data directly to AWS IoT SiteWise storage in real-time.
- `SITEWISE_BUFFERED` – Send data to Amazon S3 in batches in Parquet format, and then import into AWS IoT SiteWise storage.

Type: String

Required: Yes

`name`

A unique name for the destination.

Type: String

Required: Yes

`config`

Configuration specific to the destination type in JSON format. The configuration varies between real-time and buffered destinations.

Type: Object

Required: Yes

publishingOrder

Determines the order in which data is published. Data publishes based on its timestamp. Options include `TIME_ORDER` and `RECENT_DATA`.

- `TIME_ORDER` (default) – Publishes older data first.
- `RECENT_DATA` – Publishes newest data first.

Type: String

Required: No

enableCompression

When set to `true`, enables data compression before sending to AWS IoT SiteWise. Letting the gateway compress your data prior to uploading it to the cloud reduces bandwidth usage. The default value is `true`.

Type: Boolean

Required: No

dropPolicy

Defines how to handle older data.

Type: object

Required: No

- `cutoffAge`

The maximum age of data to be published specified in days, hours, and minutes. For example, `7d` or `1d7h16m`. Data older than what you specify is not sent to AWS IoT SiteWise.

Data that is earlier than the cutoff period is not published to the cloud. The cutoff age must be between five minutes and seven days.

You can use `m`, `h`, and `d` when you specify a cutoff age. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

Type: String

Required: Yes

- `exportPolicy`

  Defines how to handle data that exceeds the cutoff age.

  Type: Object

  Required: No

  - `retentionPeriod`

    Your SiteWise Edge gateway deletes any data at the edge that is earlier than the cutoff period from the local storage after it is stored for the specified retention period. The retention period must be between one minute and 30 days, and greater than or equal to the rotation period.

    You can use `m`, `h`, and `d` when you specify a retention period. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

    Type: String

    Required: No

  - `rotationPeriod`

    The time interval over which to batch up and save data that is earlier than the cutoff period to a single file. The SiteWise Edge gateway transfers one batch of data to the following local directory at the end of each rotation period: `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/exports`. The rotation period must be greater than one minute, and equal to or less than the retention period.

You can use `m`, `h`, and `d` when you specify a rotation period. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

Type: String

Required: No

- `exportSizeLimitGB`

  The maximum allowed size of data stored locally, in GB. If this quota is breached, the SiteWise Edge gateway starts deleting the earliest data until the size of data stored locally is equal to or less than the quota. The value of this parameter must be greater than or equal to 1.

  Type: Integer

  Required: No

`maxBatchWaitTime`

Sets a maximum time for the publisher to wait before sending a batch of data to AWS IoT SiteWise. This setting applies for each alias. The data is stored locally until either:

- The set time has elapsed, or
- 10 time-quality-value (TQV) entries are received for the alias

Use `m`, `h`, and `d` to specify a cutoff time. Note that `m` represents minutes, `h` represents hours, and `d` represents days.

Type: String

Required: No

`filters`

Filters to apply to the data. At least one filter is required.

Type: String

Required: Yes

`type`

Type of filter. Use `PATH`.

Type: String

Required: Yes

`config`

Configuration specific to the filter type in JSON format. At least one object is required, but the array can be empty.

Type: Object

Required: Yes

- `paths`

  An array of path filters. For more information, see Understand path filters for AWS IoT SiteWise Edge destinations. The default path is `#`.

  Type: Array of strings

  Required: Yes

Add an AWS IoT SiteWise buffered destination using Amazon S3

The buffered destination type allows you to save on ingestion costs into AWS IoT SiteWise if you don't need the data in real-time. It enables you to temporarily store your IoT data in an Amazon S3 bucket before importing it into AWS IoT SiteWise. Or, you can simply upload your data to S3 for storage, regardless of whether you plan to import it to AWS IoT SiteWise. This is useful for batching and buffering data from your devices and gateways before ingesting it into AWS IoT SiteWise. With this option, data is uploaded to the specified S3 bucket in Parquet format at a configured frequency. You can then import this data into AWS IoT SiteWise storage for further analysis and processing.

To add a destination buffered using Amazon S3

Use the AWS IoT SiteWise console or AWS CLI to add a destination that buffers data using Amazon S3 to your SiteWise Edge gateway.

Console

Use the AWS Management Console to add an AWS IoT SiteWise destination buffered using Amazon S3.

1. Open the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the gateway to which you want to add a destination.
4. In the **Destinations** section, choose **Add destination**.
5. On the **Add destination** page, enter **Destination details**:

    a. A name for your destination in the **Destination name** field.

    b. Select **AWS IoT SiteWise buffered using Amazon S3** for **Destination type**. AWS IoT SiteWise buffered using Amazon S3 sends data to Amazon Simple Storage Service in batches, in Parquet format, and then imports the data into AWS IoT SiteWise storage.
6. Enter the Amazon S3 URL for the location where you want to store your gateway data. You can browse for the path by choosing **Browse S3**. Once a bucket is added, you can also view the bucket by choosing **View**.
7. Specify how often your gateway should upload data to Amazon S3 by entering a time frame and selecting a time increment for **Data upload frequency**. The frequency value should be greater than 0 and less than or equal to 30 days.
8. In **Data storage settings**, determine what to do with your gateway data after importing it to AWS IoT SiteWise. There are two decisions to make regarding data storage:

    - If you want to copy imported data into AWS IoT SiteWise storage, select the **Copy data to storage** check box. This option duplicates the imported data from your configured Amazon S3 bucket into AWS IoT SiteWise storage.
    - If you choose to import your data from your Amazon S3 bucket into AWS IoT SiteWise storage, you can also specify whether the imported data should be deleted after the import is complete. Select the **Delete data from Amazon S3** check box to delete the imported date from the configured Amazon S3 bucket after importing it to AWS IoT SiteWise storage.
9. Add path filters to your destination. For more information see, Add path filters to AWS IoT SiteWise Edge destinations.

AWS CLI

Example : Create a new AWS IoT SiteWise destination buffered using Amazon S3

Use the UpdateGatewayCapabilityConfiguration API to configure the publisher.

Set the `capabilityNamespace` parameter to `iotsitewise:publisher:3`.

```
{
    "sources": [
      {
        "type": "MQTT"
      }
    ],
    "destinations": [
      {
        "type": "SITEWISE_BUFFERED",
        "name": "your-s3-destination-name",
        "config": {
          "targetBucketArn": "arn:aws:s3:::amzn-s3-demo-bucket/Optional/SomeFolder",
          "publishPolicy": {
            "publishFrequency": "15m",
            "localSizeLimitGB": 10
          },
          "siteWiseImportPolicy": {
            "enableSiteWiseStorageImport": true,
            "enableDeleteAfterImport": true,
            "bulkImportJobRoleArn": "arn:aws:iam::123456789012:role/your-role-name"
          }
        },
        "filters": [
          {
            "type": "PATH",
            "config": {
              "paths": [
                "#"
              ]
            }
          }
        ]
      }
    ]
  }
```

Example : Update an AWS IoT SiteWise destination buffered using Amazon S3

To update an existing AWS IoT SiteWise real-time destination, first use the `DescribeGatewayCapabilityConfiguration` API to find the `destinationId`.

The publisher namespace: `iotsitewise:publisher:3`

```
{
    "sources": [
      {
        "type": "MQTT"
      }
    ],
    "destinations": [
      {
        "id": "your-existing-destination-id",
        "type": "SITEWISE_BUFFERED",
        "name": "your-s3-destination-name",
        "config": {
          "targetBucketArn": "arn:aws:s3:::amzn-s3-demo-bucket/Optional/SomeFolder",
          "publishPolicy": {
            "publishFrequency": "15m",
            "localSizeLimitGB": 10
          },
          "siteWiseImportPolicy": {
            "enableSiteWiseStorageImport": true,
            "enableDeleteAfterImport": true,
            "bulkImportJobRoleArn": "arn:aws:iam::123456789012:role/your-role-name"
          }
        },
        "filters": [
          {
            "type": "PATH",
            "config": {
              "paths": [
                "#"
              ]
            }
          }
        ]
      }
```

```
    ]
}
```

The following configuration options are specific to MQTT-enabled gateways using the `iotsitewise:publisher:3` namespace.

`sources`

Defines data sources to transfer of data from your industrial equipment to AWS IoT SiteWise. For MQTT-enabled gateways, use `MQTT`.

Type: Array of objects

Required: Yes

`destinations`

Defines where to send data. Destinations are either real-time or buffered using Amazon S3. At least one destination object is required, but you can add an empty array. You can have one real-time destination for each gateway. For more information, see [Understand AWS IoT SiteWise Edge destinations](#).

Type: Array of objects

Required: Yes

`id`

The unique identifier for the destination. You can either provide an existing destination ID or leave it blank to have a new ID automatically generated for the destination.

Type: String

Required: No

`type`

Type of destination. Options include: `SITEWISE_REALTIME` and `SITEWISE_BUFFERED`. Choose `SITEWISE_BUFFERED`.

- `SITEWISE_REALTIME` (default) – Send data directly to AWS IoT SiteWise storage in real-time. For more information, see [Add an AWS IoT SiteWise Edge real-time destination](#).

- `SITEWISE_BUFFERED` – Send data to Amazon S3 in batches in Parquet format, and then import into AWS IoT SiteWise storage.

  Type: String

  Required: Yes

`name`

  A unique name for the destination.

  Type: String

  Required: Yes

`config`

  Configuration specific to the destination type in JSON format. The configuration varies between real-time and buffered destinations.

  Type: Object

  Required: Yes

  `targetBucketArn`

    The bucket ARN to publish to. Choose the same AWS Region for both AWS IoT SiteWise and Amazon S3. If a prefix is chosen, it must have between 1-255 characters.

    Note

    AWS IoT SiteWise, including the gateway, will have access to the entire specified S3 bucket. We recommend using a dedicated bucket for buffered data ingestion.

    Type: String

    Required: Yes

  `publishPolicy`

    Details of the publishing policy.

    Type: Object

Required: Yes

`publishFrequency`

The frequency with which the SiteWise Edge gateway publishes to the Amazon S3 bucket. Data upload frequency to Amazon S3 must be greater than 0 minutes and less than or equal to 30 days. You can use `m`, `h`, and `d` when you specify a publishing frequency age. Note that `m` represents minutes, `h` represents hours, and `d` represents days. The default value is 15 minutes.

Type: String

Required: Yes

`localSizeLimitGB`

The maximum size of the files written to local disk in GB. If this threshold is breached, the publisher publishes all buffered data to its destination.

Type: Integer

Required: Yes

`siteWiseImportPolicy`

Details of the import policy for importing data to AWS IoT SiteWise.

Type: Object

Required: Yes

`enableSiteWiseStorageImport`

Set this to `true` to import data from an Amazon S3 bucket to AWS IoT SiteWise storage. It initially makes a copy of the data in AWS IoT SiteWise. Then, if you set `enableDeleteAfterImport` to true, the data in S3 deletes after copying to AWS IoT SiteWise. Pricing implications apply. The default value is `true`.

Type: Boolean

Required: Yes

`enableDeleteAfterImport`

Set this to `true` to delete the file in the Amazon S3 bucket after ingestion into the AWS IoT SiteWise storage. The default value is `true`.

Type: Boolean

Required: Yes

`bulkImportJobRoleArn`

The ARN of the IAM role that AWS IoT SiteWise assumes to read buffered data from Amazon S3 during data ingestion. This role is used when an edge device calls on AWS IoT SiteWise APIs to initiate the bulk import process.

> **Note**
>
> If `enableSiteWiseStorageImport` is set to `true`, this parameter is required.

Type: String

Required: No

Add path filters for your destination. For more information, see Add path filters to AWS IoT SiteWise Edge destinations.

Add path filters to AWS IoT SiteWise Edge destinations

Add path filters to a destination. Path filters use MQTT topic syntax, where # is a wildcard character that matches any number of levels, and + is a wildcard character that matches a single level. You can add multiple destinations to a gateway, each with its own set of path filters subscribed to your equipment telemetry.

Siemens Industrial Edge gateways use a prefix for compatibility. For more information, see Prefixes for path filters.

Console

To add path filters

1. Open the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the gateway to which you want to add path filters.

4. In the **Path filters** section under **Add destination**, choose **Add path filter**.
5. Enter the path filter that you want this destination to subscribe to. You can use wildcard characters (# and +) to subscribe to multiple paths.
6. Choose **Add path filter** to add the path filter to the list.
7. Repeat steps to add additional path filters, if needed.
8. Once you have added all of the required path filters, choose **Create**.

AWS CLI for self-hosted gateways

Example : Path filter configuration

```
{
  "destinations": [
    {
      ...
    }
  ],
  "filters": [
    {
      "type": "PATH",
      "config": {
        "paths": [
          "home/+/sensor1/temperature",
          "home/livingroom/sensor1/temperature",
          "home/livingroom/sensor1/temperature",
          "building/#"
        ]
      }
    }
  ]
}
```

AWS CLI for Siemens IEgateways

Example : Prefix configuration for path filters

Capture all data by using both the data (`ie/d`) and metadata (`ie/m`) prefixes for each path filter.

```
{
  "destinations": [
    {
      ...
    }
  ],
  "filters": [
    {
      "type": "PATH",
      "config": {
        "paths": [
          "ie/d/home/+/sensor12/temperature",
          "ie/m/home/livingroom/sensor12/temperature",
          "ie/d/home/livingroom/sensor13/temperature2",
          "ie/m/home/livingroom/sensor13/temperature2",
          "ie/d/building/#",
          "ie/m/building/#"
        ]
      }
    }
  ]
}
```

Note

Copy path filters between destinations by downloading list of path filters. For more information, see Download all path filters in a destination (console).

Upload path filters in bulk

To upload path filters in bulk, use a CSV or text file. AWS IoT SiteWise automatically removes exact duplicates when you upload files. For example, `windfarm/site1/` and `windfarm/site1/` are exact duplicates that AWS IoT SiteWise catches because the string is exactly the same. Partial duplicates are not removed and result in additional charges. For example, `windfarm/#` and `windfarm/site1` are overlapping topics because `windfarm/site1` is already encompassed by `windfarm/#`.

Note

Avoid duplicates to prevent additional charges. The uploaded file must be in either .csv or .txt format. It can't contain any headers and should consist of a single column. In the column, list your path filters, with each filter on a separate line. No other information should be included in the file.

File upload requirements

These are additional path filter requirements.

- You can upload one .csv or .txt file. Other file formats are not supported.
- CSV (.csv) files cannot have headers and should only contain one column.
- You can have one path filter on each line.
- The uploaded files cannot be empty.
- When using # as a wildcard, it must be the last character in the topic filter. For example, `topic/#` or as a standalone character at a particular topic level. However, note that # can also be used as a regular character within a topic level name, such as `factory/machine#1/topic`. For more information see [Special characters in path filter names]
  - You can also use the + character. For example, use `factory/+/temp` to get all temperatures for factories instead of `factory/machine2/temp` and `factory/machine3/temp` individually.

Manage AWS IoT SiteWise Edge destinations

After adding destinations, you can perform various operations to manage them, such as editing destination configurations, deleting destinations, and managing path filters.

Edit a destination

Select the radio button next to the destination in the table and choose the **Edit** button to edit a destination.

Console

To edit a destination using the AWS IoT SiteWise console

1. Open the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/).
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the appropriate gateway.
4. In the **Destinations** section, choose destination you want to edit and then choose **Edit**.
5. Modify the destination and then choose **Save**.

AWS CLI

To edit a destination using AWS CLI

- You can edit a destination by modifying the JSON capability configuration information.

```
aws iotsitewise update-gateway-capability-configuration \
--gateway-id your-gateway-id \
--capability-namespace "iotsitewise:publisher:3" \
--capability-configuration '{
    "sources": [
        {
            "type": "MQTT"
        }
    ],
    "destinations": [
        {
            "id": "your-existing-destination-id",
            "type": "SITEWISE_REALTIME",
            "name": "your-updated-destination-name",
            "config": {
                "publishingOrder": "TIME_ORDER",
                "enableCompression": true,
                "dropPolicy": {
                    "cutoffAge": "10d",
                    "exportPolicy": {
                        "retentionPeriod": "10d",
                        "rotationPeriod": "6h",
                        "exportSizeLimitGB": 10
                    }
                },
```

```
                "maxBatchWaitTime": "15s"
            },
            "filters": [
                {
                  ...
                }
            ]
        }
    ]
}'
```

Note

You can't update the destination `type` or `capability-namespace`. For example, you can't switch from a type of `SITEWISE_REALTIME` to `SITEWISE_BUFFERED`. You can have one real-time destination for each MQTT-enabled gateway.

Delete a destination

If you no longer need a destination, you can delete it from your SiteWise Edge gateway.

Console

To delete a destination using the AWS IoT SiteWise console

1. Open the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the appropriate gateway.
4. In the **Destinations** section, choose destination you want to delete and then choose **Delete**. A confirmation screen appears.
5. To confirm your choice to delete the destination, type "delete" in the confirmation box.

AWS CLI

To delete a destination using AWS CLI

- Delete the gateway capability configuration by specifying the gateway ID and modifying the capability configuration to remove the destination you want to delete.

```
aws iotsitewise update-gateway-capability-configuration \
  --gateway-id your-gateway-id \
  --capability-namespace "iotsitewise:publisher:3" \
  --capability-configuration '{
    "sources": [
        {
            "type": "MQTT"
        }
    ],
    "destinations": []
}'
```

> **Note**
>
> The destinations array can be empty ([]), but the destinations object itself must be included in the capability configuration.

Download all path filters in a destination (console)

Download a CSV file containing all of your path filters in the AWS IoT SiteWise console. You can use a downloaded list of path filters to easily share path filter lists between gateway destinations.

To download a CSV file of all path filters using the AWS IoT SiteWise console

1. Open the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the gateway containing your path filters.
4. Choose either **Add destination** or **Edit destination**.
5. Navigate to the **Path filters** section and choose **Download CSV**.

Note

The CSV file includes all path filters in a particular destination, regardless of which ones you selected from the list of path filters.

Edit a path filter

You can edit individual path filters to refine which data your destination receives.

Console

Using the AWS IoT SiteWise console, you can edit each individual path filter within each respective text box.

To edit a path filter using the AWS IoT SiteWise console

1. Open the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the gateway containing your path filters.
4. Select the appropriate destination.
5. Choose **Edit**.
6. Choose the text box for the row containing the path filter that you want to edit.
7. Update the path filter's text, ensuring that the edited path filter's checkbox is selected.
8. Choose **Save**.

AWS CLI

To edit path filters for a destination using the AWS CLI, first retrieve the current configuration, modify it, and then update it using the `update-gateway-capability-configuration` command.

To edit a path filter using AWS CLI

1. Retrieve the current capability configuration:

```
aws iotsitewise describe-gateway-capability-configuration \
  --gateway-id your-gateway-id \
```

```
  --capability-namespace "iotsitewise:publisher:3" \
  --query "capabilityConfiguration"
```

2. Edit the JSON to modify the path filters as needed.
3. Update the capability configuration with the modified path filters:

```
aws iotsitewise update-gateway-capability-configuration \
  --gateway-id your-gateway-id \
  --capability-namespace "iotsitewise:publisher:3" \
  --capability-configuration json-containing-your-updated-path-filters
```

Delete a path filter

You can delete path filters for a destination to control the data it receives from MQTT sources and data processing pipelines.

Console

To delete a path filter using the AWS IoT SiteWise console

1. Open the AWS IoT SiteWise console.
2. In the left navigation, choose **Edge gateways** in the **Edge** section.
3. Select the gateway containing your path filters.
4. Select the appropriate destination.
5. Choose **Edit**.
6. On the **Edit destination** screen, in the **Path filters** section, select one or more the path filters to delete.
7. Choose **Delete**. A deletion confirmation message appears. If want to proceed with deleting the path filters, choose **Delete** on the confirmation screen.

AWS CLI

To delete a destination using AWS CLI

- Delete a path filter by removing it from the capability configuration.

```
aws iotsitewise update-gateway-capability-configuration \
  --gateway-id your-gateway-id \
```

```
  --capability-namespace "iotsitewise:publisher:3" \
  --capability-configuration '{
    "sources": [
        {
            "type": "MQTT"
        }
    ],
    "destinations": [
        {
            "id": "your-destination-id",
            "type": "SITEWISE_REALTIME",
            "name": "your-destination-name",
            "config": {
                ...
            },
            "filters": [
                {
                    "type": "PATH",
                    "config": {
                        "paths": [
                            "/path1",
                            "/path2",
                            "/delete-a-path-to-remove-it"
                        ]
                    }
                }
            ]
        }
    ]
}
```

Note

The filters array can be empty ([]), but the filters object itself must be included in the capability configuration.

Manage SiteWise Edge gateways

You can use the AWS IoT SiteWise console and API operations to manage AWS IoT SiteWise Edge gateways. You can also use the AWS OpsHub for AWS IoT SiteWise for Windows application to manage some aspects of your SiteWise Edge gateway from your local device.

We highly recommend that you use the AWS OpsHub for AWS IoT SiteWise application to monitor the disk usage on your local device. You can also monitor the `Gateway.AvailableDiskSpace` and `Gateway.UsedPercentageDiskSpace` Amazon CloudWatch metrics and create alarms to get notified when the disk space is getting low. For more information about Amazon CloudWatch alarms, see Create a CloudWatch alarm based on a static threshold.

Make sure that your device has enough space for upcoming data. When you're about to run out of space on your local 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 SiteWise Edge gateway storage from running out of space.

Manage your SiteWise Edge gateway with the AWS IoT SiteWise console

You can use the AWS IoT SiteWise console to configure, update, and monitor all SiteWise Edge gateways in your AWS account.

You can view your SiteWise Edge gateways by navigating to the **Edge Gateways** page in the AWS IoT SiteWise console. To access the **Edge gateway details** page for a specific gateway, choose the name of an Edge gateway.

From the **Overview** tab of the **Edge gateway details** page, you can do the following:

- In the **Data sources** section, update data source configuration and configure additional data sources

- Choose **Open CloudWatch metrics** to view the number of data points ingested per data source in the CloudWatch metrics console
- In the **Edge capabilities** section, add data packs to your SiteWise Edge gateway by clicking **Edit**
- In the **Gateway configuration** section, view the connectivity status of your SiteWise Edge gateways
- In the **Publisher configuration** section, view the SiteWise Edge gateway sync status and configuration of the AWS IoT SiteWise publisher component

From the **Updates** tab of the **Edge gateway details** page, you can see the current component and pack versions that are deployed to the Edge gateway. This is also where you deploy new versions, when they're available.

Manage SiteWise Edge gateways using AWS OpsHub for AWS IoT SiteWise

You use the AWS OpsHub for AWS IoT SiteWise application to manage and monitor your self-hosted SiteWise Edge gateways. This application provides the following monitoring and management options:

- Under **Overview**, you can do the following:
  - View SiteWise Edge gateway details that help you get insights into your SiteWise Edge gateway device data, identify issues, and improve the SiteWise Edge gateway's performance.
  - View SiteWise Monitor portals that monitor the data from local servers and equipment at the edge. For more information, see [What is AWS IoT SiteWise Monitor](#) in the *AWS IoT SiteWise Monitor Application Guide*.
- Under **Health**, there's a dashboard that displays data from your SiteWise Edge gateway. Domain experts, such as process engineers, can use the dashboard to see an overview of SiteWise Edge gateway behavior.
- Under **Assets**, view assets deployed to the local device and the last value collected or computed for asset properties.
- Under **Settings**, you can do the following:
  - If the Data Processing Pack is installed, view the SiteWise Edge gateway configuration information and sync resources with the AWS Cloud.
  - Download the authentication files that you can use to access the SiteWise Edge gateway by using other tools.

- Download logs that you can use to troubleshoot the SiteWise Edge gateway.
- View the AWS IoT SiteWise components deployed to the SiteWise Edge gateway.

> **Important**
>
> The following are required to use AWS OpsHub for AWS IoT SiteWise:
>
> - Your local device and the AWS OpsHub for AWS IoT SiteWise application must be connected to the same network.
> - The data processing pack must be enabled.

To manage SiteWise Edge gateways using AWS OpsHub

1. Download and install the AWS OpsHub for AWS IoT SiteWise for Windows application.
2. Open the application.
3. If you don't have local credentials set up for your gateway, follow the steps under Access your SiteWise Edge gateway using local operating system credentials to set them up.
4. You can sign in to your SiteWise Edge gateway with your Linux or Lightweight Directory Access Protocol (LDAP) credentials. To sign in to your SiteWise Edge gateway, do one of the following:

   Linux

   1. For **Hostname or IP address**, enter the hostname or IP address of your local 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 local 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**.

Access your SiteWise Edge gateway using local operating system credentials

Besides Lightweight Directory Access Protocol (LDAP), you can use the Linux or Windows credentials to access your self-hosted SiteWise Edge gateway.

> **Important**
>
> To access your SiteWise Edge gateway with Linux credentials, you must activate the data processing pack for your SiteWise Edge gateway.

Access your SiteWise Edge gateway using Linux operating system credentials

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.

To create a Linux user pool

1. To create an admin group, run the following command.

```
sudo groupadd --system SWE_ADMIN_GROUP
```

   Users in the `SWE_ADMIN_GROUP` group can allow admin access for the SiteWise Edge gateway.

2. To create a user group, run the following command.

```
sudo groupadd --system SWE_USER_GROUP
```

   Users in the `SWE_USER_GROUP` group can allow read-only access for the SiteWise Edge gateway.

3. To add a user to the admin group, run the following command. Replace *`user-name`* and *`password`* with the user name and password that you want to add.

```
sudo useradd -p $(openssl passwd -1 password) user-name
```

4. To add a user to either `SWE_ADMIN_GROUP` or `SWE_USER_GROUP`, replace *user-name* with the the user name that you added in the previous step.

```
sudo usermod -a -G SWE_ADMIN_GROUP user-name
```

You can now use the user name and password to sign in to the SiteWise Edge gateway on the AWS OpsHub for AWS IoT SiteWise application.

Access your SiteWise Edge gateway using Windows credentials

The following steps assume that you use a device with Windows.

> **Important**
>
> Security is a shared responsibility between AWS and you. Create a strong password policy with at least 12 characters and a combination of uppercase, lowercase, numbers, and symbols. Additionally, set the Windows Firewall rules to allow incoming traffic on port 443 and to block incoming traffic on all other ports.

To create a Windows Server user pool

1. Run PowerShell as the administrator.

   a. On the Windows server where you want to install SiteWise Edge Gateway, log in as administrator.

   b. Enter **PowerShell** in the Windows search bar.

   c. In the search results, right click on the Windows PowerShell app. Choose **Run as Administrator**.

2. To create an admin group, run the following command.

```
net localgroup SWE_ADMIN_GROUP /add
```

   You must be a user in the `SWE_ADMIN_GROUP` group to allow admin access for the SiteWise Edge gateway.

3. To create a user group, run the following command.

```
net localgroup SWE_USER_GROUP /add
```

You must be a user in the SWE_USER_GROUP group to allow ready-only access for the SiteWise Edge gateway.

4. To add user, run the following command. Replace *user-name* and *password* with the user name and the password that you want to create.

```
net user user-name password /add
```

5. To add a user to the admin group, run the following command. Replace *user-name* with the user name that you want to add.

```
net localgroup SWE_ADMIN_GROUP user-name /add
```

You can now use the user name and password to sign in to the SiteWise Edge gateway on the AWS OpsHub for AWS IoT SiteWise application.

Manage the SiteWise Edge gateway certificate

You can use SiteWise Monitor and third-party applications, such as Grafana, on your SiteWise Edge gateway devices. These applications require a TLS connection to the service. SiteWise Edge gateways currently use a self-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*.

Change the version of SiteWise Edge gateway component packs

You can use the AWS IoT SiteWise console to change the version of component packs on your SiteWise Edge gateways.

To change the version of a SiteWise Edge gateway component pack

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Gateways**.
3. Select the SiteWise Edge gateway that you would like to change the pack versions for.
4. Under **Gateway configuration**, choose **View software versions**.
5. On the **Edit software versions** page, for the pack you want to update the version of, select the version you want to deploy and choose **Deploy**.
6. Choose **Done**.

List SiteWise Edge gateways

Console

To list SiteWise Edge gateways

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. View the list of all your SiteWise Edge gateways.

AWS CLI

To list your gateways using AWS CLI, follow these steps:

- Use the list-gateways command to view all your gateways:

```
aws iotsitewise list-gateways
```

This command returns a list of your gateways with their IDs, names, and other information.

You can also specify pagination parameters:

```
aws iotsitewise list-gateways --max-results 10 --next-token your-token
```

For more information, see list-gateways in the AWS CLI Command Reference.

Describe a SiteWise Edge gateway

Console

To view gateway details

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the name of the gateway you want to view details for.
4. View the gateway details on the **Edge gateway details** page.

AWS CLI

To get detailed information about a specific gateway using AWS CLI, follow these steps:

- Use the describe-gateway command with the gateway ID:

```
aws iotsitewise describe-gateway --gateway-id a1b2c3d4-5678-90ab-
cdef-1a1a1EXAMPLE
```

This command returns detailed information about the gateway.

For more information, see describe-gateway in the AWS CLI Command Reference.

Create a SiteWise Edge gateway

Console

To create a SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose **Create gateway**.
4. Enter a name for your gateway.
5. Select the Greengrass group for your gateway.
6. Optionally, add tags to your gateway.
7. Choose **Create**.

AWS CLI

To create a new IoT SiteWise gateway using AWS CLI, follow these steps:

- Use the create-gateway command to create a new gateway:

```
aws iotsitewise create-gateway \
    --gateway-name "NewSiteWiseGateway" \
    --gateway-platform '{
        "greengrass": {
            "groupArn": "arn:aws:greengrass:us-east-1:123456789012:group/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE"
        }
    }' \
    --tags '{
        "Environment": "Production",
        "Location": "Factory1"
    }'
```

This command returns the new gateway's ID and ARN:

```
{
    "gatewayId": "a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE",
    "gatewayArn": "arn:aws:iotsitewise:us-east-1:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE"
```

```
}
```

For more information, see create-gateway in the AWS CLI Command Reference.

Update a SiteWise Edge gateway

Console

To update a SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the gateway you want to update.
4. Choose **Edit**.
5. Update the gateway name or other settings as needed.
6. Choose **Save**.

AWS CLI

To update an existing gateway using AWS CLI, follow these steps:

- Use the update-gateway command to update a gateway's name:

  ```
  aws iotsitewise update-gateway \
      --gateway-id a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE \
      --gateway-name "UpdatedGatewayName"
  ```

  This command produces no output when successful.

For more information, see update-gateway in the AWS CLI Command Reference.

Update gateway capability configuration

Console

To update gateway capability configuration

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the name of the gateway you want to update.
4. In the **Data sources** section, choose **Edit**.
5. Update the data source configuration as needed.
6. Choose **Save**.

AWS CLI

To update a gateway's capability configuration using AWS CLI, follow these steps:

- Use the update-gateway-capability-configuration command to update the capability configuration:

```
aws iotsitewise update-gateway-capability-configuration \
    --gateway-id a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE \
    --capability-namespace "iotsitewise:opcuacollector:1" \
    --capability-configuration '{
        "sources": [
            {
                "name": "OPC-UA Server",
                "endpoint": {
                    "certificateTrust": {
                        "type": "TrustAny"
                    },
                    "endpointUri": "opc.tcp://10.0.0.1:4840",
                    "securityPolicy": "NONE",
                    "messageSecurityMode": "NONE",
                    "identityProvider": {
                        "type": "Anonymous"
                    }
                },
                "measurementDataStreamPrefix": ""
            }
```

```
        ]
    }'
```

This command returns the capability namespace and sync status:

```
{
    "capabilityNamespace": "iotsitewise:opcuacollector:1",
    "capabilitySyncStatus": "CONFIGURING"
}
```

For more information, see update-gateway-capability-configuration in the AWS CLI Command Reference.

Tag gateway resources

Console

To tag a gateway resource

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the name of the gateway you want to tag.
4. Choose the **Tags** tab.
5. Choose **Manage tags**.
6. Choose **Add new tag** and enter a key and value for each tag.
7. Choose **Save**.

AWS CLI

To add tags to a gateway using AWS CLI, follow these steps:

- Use the tag-resource command to add tags to a gateway:

```
aws iotsitewise tag-resource \
    --resource-arn "arn:aws:iotsitewise:us-east-1:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE" \
    --tags '{
```

```
        "Department": "Operations",
        "Project": "FactoryAutomation"
    }'
```

This command produces no output when successful.

For more information, see tag-resource in the AWS CLI Command Reference.

List tags for a gateway

Console

To list tags for a gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the name of the gateway you want to view tags for.
4. Choose the **Tags** tab.
5. View the list of tags associated with the gateway.

AWS CLI

To list the tags associated with a gateway using AWS CLI, follow these steps:

- Use the list-tags-for-resource command to list tags for a gateway:

```
aws iotsitewise list-tags-for-resource \
    --resource-arn "arn:aws:iotsitewise:us-east-1:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE"
```

This command returns the tags associated with the gateway:

```
{
    "tags": {
        "Environment": "Production",
        "Location": "Factory1",
        "Department": "Operations",
        "Project": "FactoryAutomation"
```

```
    }
}
```

For more information, see list-tags-for-resource in the AWS CLI Command Reference.

Remove tags from a gateway

Console

To remove tags from a gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the name of the gateway you want to remove tags from.
4. Choose the **Tags** tab.
5. Choose **Manage tags**.
6. Choose the remove icon (X) next to each tag you want to remove.
7. Choose **Save**.

AWS CLI

To remove tags from a gateway using AWS CLI, follow these steps:

- Use the untag-resource command to remove tags from a gateway:

```
aws iotsitewise untag-resource \
    --resource-arn "arn:aws:iotsitewise:us-east-1:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE" \
    --tag-keys '["Project", "Department"]'
```

  This command produces no output when successful.

For more information, see untag-resource in the AWS CLI Command Reference.

Update the version of an AWS IoT SiteWise component

Update the AWS IoT SiteWise gateway component on your AWS IoT Greengrass core device to ensure your access to the latest features, performance improvements, and security patches.

To update an AWS IoT SiteWise component on AWS IoT Greengrass

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Edge gateways**.
3. Select the gateway to edit and choose **Edit**.
4. In **Edge Capabilities**, under **Software versions**, choose **Software updates available**. The **Edit software versions** page appears.
5. Choose the component version.

> **Note**
>
> It's recommend to select the latest version available. Keeping gateway components up-to-date helps you maintain optimal functionality for industrial data collection and processing.

6. Choose **Deploy**. This starts an AWS IoT Greengrass V2 deployment to update the AWS IoT SiteWise component on the gateway.

Delete a SiteWise Edge gateway

Console

To delete the SiteWise Edge gateway

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Choose the gateway you want to delete.
4. Choose **Delete**.
5. To confirm you want to delete the gateway, type "delete" and then choose **Delete** in the window that appears.

AWS CLI

To delete a gateway using AWS CLI, follow these steps:

1. List your gateways to identify the gateway ID of the gateway you want to delete.

```
aws iotsitewise list-gateways
```

This command returns a list of your gateways with their IDs, names, and other information:

```
{
    "gatewaySummaries": [
        {
            "gatewayId": "a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE",
            "gatewayName": "ExampleCorpGateway",
            "gatewayCapabilitySummaries": [
                {
                    "capabilityNamespace": "iotsitewise:opcuacollector:1",
                    "capabilitySyncStatus": "IN_SYNC"
                }
            ],
            "creationDate": 1588369971.457,
            "lastUpdateDate": 1588369971.457
        }
    ]
}
```

2. Delete the gateway by specifying its ID:

```
aws iotsitewise delete-gateway --gateway-id a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE
```

This command produces no output when successful.

> **Note**
>
> When you delete a gateway, some of the gateway's files remain in your gateway's file system.

3. To verify that the gateway has been deleted, you can list your gateways again:

```
aws iotsitewise list-gateways
```

The deleted gateway should no longer appear in the list.

For more information, see delete-gateway in the AWS CLI Command Reference.

Back up and restore SiteWise Edge gateways

This topic covers how to restore SiteWise Edge gateways and backup your metric data. If you are experiencing issues with a broken SiteWise Edge gateway on the same machine and need to troubleshoot the issue, please read the AWS IoT SiteWise documentation Troubleshooting SiteWise Edge gateway issues.

> **Note**
>
> The guidance covered in this topic is for SiteWise Edge gateways installed on AWS IoT Greengrass V2 version 2.1.0 or higher.

Daily backups of metric data

Creating a back up is important, if you would like to transfer or restore the data on a new machine. Backing up your data greatly reduces the risk of loss of operating data during a transfer or restoration process.

This section applies to gateways that use the data processing pack. For more information on the data processing pack, see Configure an asset model for data processing on SiteWise Edge.

The **influxdb** folder path is as follows:

Linux

`/greengrass/v2/work/aws.iot.SiteWiseEdgeProcessor/influxdb`

Windows

`C:\greengrass\v2\work\aws.iot.SiteWiseEdgeProcessor\influxdb`

We recommend that you backup the whole folder with everything underneath it.

We recommend that you periodically backup your metric data from the 1.0 SiteWise Edge to either an external hard drive or to the AWS cloud.

Restore a SiteWise Edge gateway

Before attempting to restore a SiteWise Edge gateway, ensure that all edge devices connected to the gateway are stopped or disconnected.

Use the following procedure to a restore a SiteWise Edge gateway:

1. Use the installation script downloaded when you create SiteWise Edge gateway to restore the SiteWise Edge gateway on the new machine. Read the Installing the SiteWise Edge gateway software on your local device procedure to setup the SiteWise Edge gateway.

   If you lose or cannot find the installation script, please contact AWS Customer Support.
2. Once the SiteWise Edge gateway has been installed, log into the AWS IoT Greengrass console.
3. To redeploy the components, navigate to **Manage** then under **AWS IoT Greengrass devices** select **Core devices**.
4. In the **AWS IoT Greengrass core devices** table select the core device corresponding to your SiteWise Edge gateway.
5. Once on the device page, open the **Deployments** tab and select your **Deployment ID**, this will open the **Deployments** page with your selected ID.
6. Once you are on the **Deployments** page, in the top right press the **Actions** button, and select the **Revise** option. to initiate a new deployment. Configure the deployment. If you would like to keep the deployment as it is, skip to **Review** and **Deploy**.
7. Wait for the **Deployment Status** to become `Completed`.

   > **Note**
   >
   > It will also take a few minutes for all components on the SiteWise Edge to fully setup and running.

Restore AWS IoT SiteWise data

Use the following procedure to restore data on a new machine.

1. Copy the `influxdb` folder to the new machine.
2. Stop the SiteWise EdgeProcessor component, by running the following command in your terminal:

   Linux

   ```
   sudo /greengrass/v2/bin/greengrass-cli component stop -n
   aws.iot.SiteWiseEdgeProcessor
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli component stop -n
   aws.iot.SiteWiseEdgeProcesso
   ```
3. Locate the path where you backed up your data, and run the following command:

   Linux

   ```
   sudo yes | sudo cp -rf <influxdb_backup_path> /greengrass/v2/work/
   aws.iot.SiteWiseEdgeProcessor/influxdb
   ```

   PowerShell

   ```
   Copy-Item -Recurse -Force <influxdb_backup_path>\* C:\greengrass
   \v2\work\aws.iot.SiteWiseEdgeProcessor\
   ```

   Windows

   ```
   robocopy <influxdb_backup_path> C:\greengrass\v2\work
   \aws.iot.SiteWiseEdgeProcessor\ /E
   ```
4. Restart the SiteWiseEdgeProcessor component:

   Linux

   ```
   sudo /greengrass/v2/bin/greengrass-cli component restart -n
   aws.iot.SiteWiseEdgeProcessor
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli component restart -n
   aws.iot.SiteWiseEdgeProcessor
   ```

Validate successful backups and restorations

Use this procedure validate your backed-up data and SiteWise Edge gateway restorations.

> **Note**
>
> This procedure requires that you have installed AWS OpsHub for AWS IoT SiteWise. For more information see, Managing SiteWise Edge gateways using AWS OpsHub for AWS IoT SiteWise.

1. Open AWS OpsHub for AWS IoT SiteWise.
2. On the SiteWise Edge Gateway **Settings** page, check the status of each component listed in the **Components** table. Verify that the status color is green and the readout displays **RUNNING**.
3. Validate your past data on the portal dashboard to check that the past data and the new data are both properly setup. There will be a downtime between past and new data. You should except to see a duration where no data points are collected.

If you run into issues with backing up or restoring a SiteWise Edge gateway see the following troubleshooting topics Troubleshooting an AWS IoT SiteWise Edge gateway.

Legacy gateways (AWS IoT Greengrass Version 1)

> **Note**
>
> SiteWise Edge gateways running on AWS IoT Greengrass V1 are available only if you started using this feature before July 29, 2021. For more information on running an AWS IoT SiteWise gateway using AWS IoT Greengrass V2, see Self-host an AWS IoT SiteWise Edge gateway with AWS IoT Greengrass V2.

SiteWise Edge gateways now exclusively run on AWS IoT Greengrass V2, providing enhanced functionality and improved performance for your industrial IoT applications. This latest version AWS IoT Greengrass V2 represents an architectural evolution, built on a modern component-based framework that enables modular software deployment. It streamlines installation through a unified installer while offering developers greater flexibility in deploying custom components and conducting local testing. The component-based model allows for more efficient resource management and introduces a simplified configuration approach through component recipes. This design facilitates better dependency handling between components, supports continuous deployment practices, and provides enhanced CLI capabilities for local development. Additionally, AWS IoT Greengrass V2 centralizes configuration management through AWS IoT Core and delivers improved logging and monitoring features, all protected by a more granular security permissions model.

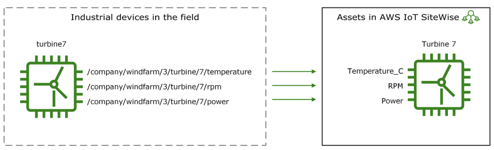
For more information on getting started with SiteWise Edge gateways using AWS IoT Greengrass V2, AWS IoT SiteWise Edge self-hosted gateway requirements. These resources provide step-by-step instructions on setting up your gateways, configuring data sources, and managing your industrial IoT infrastructure.

> **Note**
>
> As AWS continues to innovate and improve its IoT services, it's recommended to stay updated with the latest features and enhancements. Regularly check the AWS IoT SiteWise and AWS IoT Greengrass documentation for new capabilities that can further optimize your industrial IoT solutions.

Model industrial assets

Assets overview

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.

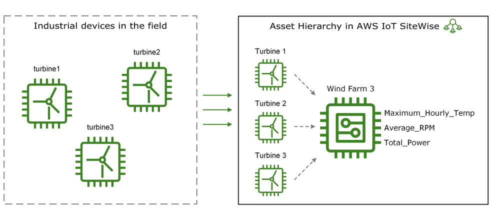


Property aliases identify equipment data streams

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.

Asset hierarchies represent equipment relationships

An asset can also represent a logical grouping of devices, such as an entire wind farm. You can associate assets with other assets to create asset hierarchies that represent complex industrial operations. Assets can access the data within their associated child assets. By doing so, you can use AWS IoT SiteWise expressions to calculate aggregate metrics, such as the net power output of a wind farm.



Asset models standardize equipment representation

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. For example, a manufacturing plant might have an asset model for CNC machines that defines properties such as temperature, downtime, and production rate. In the preceding diagram, you use the same asset model for all three turbines because they share a common set of properties.

Modeling options for industrial equipment

When designing your industrial asset representation, consider these options:

- **Asset models** represent specific types of equipment or processes. You must create each physical asset from an asset model. For example, a chemical processing plant might have separate asset models for reactors, mixers, and storage tanks.
- **Component models** define reusable sub-assemblies that you can include in asset models or other component models. For example, you could include a temperature sensor component model in multiple equipment asset models across a factory.
- **Asset model interfaces** apply standards across different asset models. For example, a "Rotating Equipment" interface could define standard properties for vibration, temperature, and RPM that apply to pumps, turbines, and motors, despite each having its own unique asset model.

Creating and managing assets

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, asset models, and component models. When you update an asset model, every asset based on that asset model reflects any changes that you make to the underlying model. When you update a component model, this applies to every asset based on every asset model that references the component model.

Managing complex asset models

Your asset models may be very complex, for example when modeling a complicated piece of equipment that has many subcomponents. To help keep such asset models organized and maintainable, you can use custom composite models to group related properties or to re-use shared components. For more information, see Custom composite models (components).

Asset and model states

When you create, update, or delete an asset, an asset model, or a component model, the changes take time to propagate. AWS IoT SiteWise resolves these operations asynchronously and updates the status of each resource. Each asset, asset model, and component 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 resource is active. This is the only state in which you can query and interact with assets, asset models, and component models.
- `CREATING` – The resource is being created.
- `UPDATING` – The resource is being updated.
- `DELETING` – The resource is being deleted.
- `PROPAGATING` – (Asset models and component models only) The changes are propagating to all dependent resources (from asset model to assets, or from component model to asset models).
- `FAILED` – The resource failed to validate during a create or update operation, possibly due to a circular reference in an expression. You can delete resources that are in the `FAILED` state.

Some of the create, update, and delete operations in AWS IoT SiteWise place an asset, asset model, or component model in a state other than `ACTIVE` while the operation resolves. To query or interact with a resource after you perform one of these operations, you must wait until the state changes to `ACTIVE`. Otherwise, your requests fail.

Topics

Check the status of an asset

You can use the AWS IoT SiteWise console or API to check the status of an asset.

Topics

- Check the status of an asset (console)
- Check the status of an asset (AWS CLI)

Check the status of an asset (console)

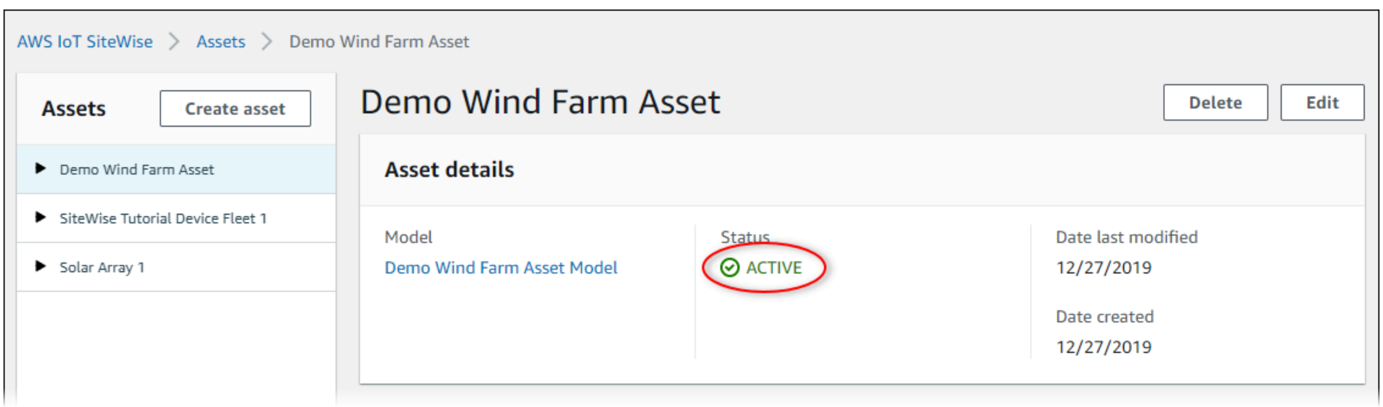
Use the following procedure to check the status of an asset in the AWS IoT SiteWise console.

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.



Check the status of an asset (AWS 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 (AWS CLI)

- Run the following command to describe the asset. Replace *asset-id* with the asset's ID or external ID. The external ID is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

```
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:

```
{
    ...
    "assetStatus": {
      "state": "String",
      "error": {
         "code": "String",
         "message": "String"
      }
    }
  }
```

The asset's state is in `assetStatus.state` in the JSON object.

Check the status of an asset or component model

You can use the AWS IoT SiteWise console or API to check the status of an asset model or component model.

Topics

- Check the status of an asset model or component model (console)
- Check the status of an asset model or component model (AWS CLI)

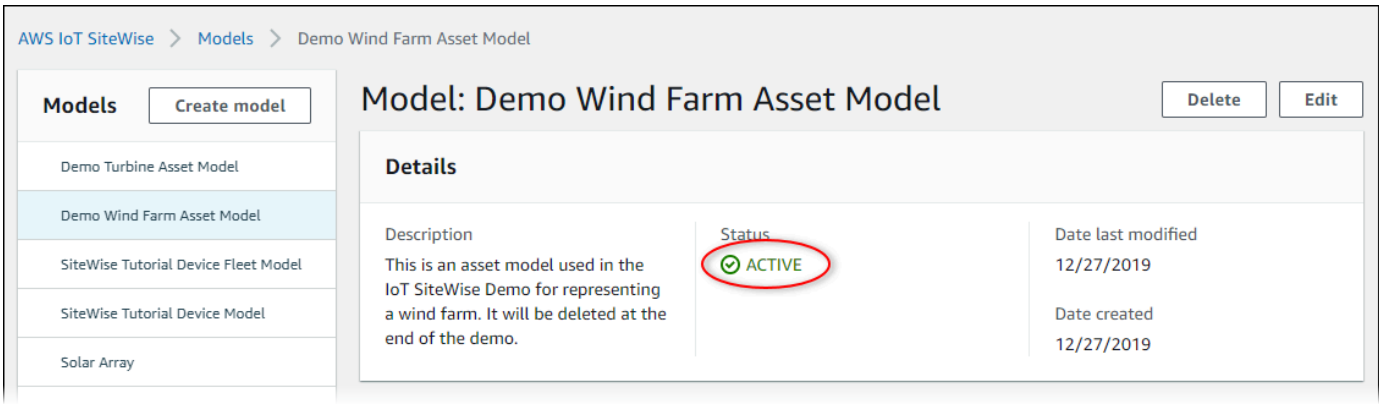
Check the status of an asset model or component model (console)

Use the following procedure to check the status of an asset model or component model in the AWS IoT SiteWise console.

> **Tip**
>
> Asset models and component models are both listed under **Models** in the navigation pane. The **Details** panel of the selected asset model or component model indicates which type it is.

To check the status of an asset model or component model (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the model to check.
4. Find **Status** in the **Details** panel.



Check the status of an asset model or component model (AWS CLI)

You can use the AWS CLI to check the status of an asset model or component model.

To check the status of an asset model or component model, use the DescribeAssetModel operation with the `assetModelId` parameter.

> **Tip**
>
> The AWS CLI defines component models as a type of asset model. Therefore, you use the same DescribeAssetModel operation for both types of model. The `assetModelType` field in the response indicates whether it's an `ASSET_MODEL` or a `COMPONENT_MODEL`.

To check the status of an asset model or component model (AWS CLI)

- Run the following command to describe the model. Replace *asset-model-id* with the ID or the external ID of the asset model or component model. The external ID is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

  ```
  aws iotsitewise describe-asset-model --asset-model-id asset-model-id
  ```

  The operation returns a response that contains the model's details. The response contains an `assetModelStatus` object that has the following structure.

  ```
  {
      ...
      "assetModelStatus": {
        "state": "String",
        "error": {
           "code": "String",
           "message": "String"
        }
      }
    }
  ```

  The model's state is in `assetModelStatus.state` in the JSON object.

Asset model versions

AWS IoT SiteWise supports asynchronous processing of create and update operations on asset models and component models. It also updates the status of the model.

AWS IoT SiteWise propagates a valid model's changes in the create and update requests to its dependent resources (from asset model to assets, or from component model to asset models). It then places the model in `ACTIVE` state.

If the provided model definition is invalid, AWS IoT SiteWise places the model in a `FAILED` state. The changes are not propagated to the dependent resources. The dependent resources refer to the last model definition propagated when the model was in an `ACTIVE` state.

Based on the information above, model definitions have two types of model versions:

1. **Latest version –** The latest definition accepted as part of a create or update request.
2. **Active version –** The latest definition successfully processed, and the model state is `ACTIVE`.

By default, details of the model's latest version is returned when describe APIs are called on an asset model or component model. There are scenarios where the active version of the asset model or component model is needed. See example scenarios below:

- An update operation with an invalid definition places your asset model in a `FAILED` state. You must revert your changes by retrieving the active version of the asset model, and creating another update request referring to this valid definition.
- An application on AWS IoT SiteWise exists where customers can view assets and their corresponding asset models. When a user refers the asset model definition corresponding to a particular asset, and the asset model is in a transitory `UPDATING`, `PROPAGATING`, or `FAILED` state, the latest version returns the asset model definition that is not yet propagated to its assets. In this case, you must retrieve the active version of the asset model to customers.

Topics

Retrieve the active version of an asset model or component model (console)

Follow this procedure to retrieve the active version of an asset model or component model in the AWS IoT SiteWise console.

> **Tip**
>
> Asset models and component models are both listed under **Models** in the navigation pane. The **Details** panel of the selected asset model or component model indicates which type it is.

To retrieve the active version of an asset model or component model (console)

1. Navigate to the AWS IoT SiteWise console.

2. In the navigation pane, choose **Models**.
3. Choose the model to retrieve its active version.
   a. If the model is in an `ACTIVE` state, you are viewing its active version.
   b. If the model is in a transitory `UPDATING`, `PROPAGATING`, or `FAILED` state, find the **See active version** under **Status** in the **Details** panel.

Retrieve the active version of an asset model or component model (AWS CLI)

Use the AWS CLI to retrieve the active version of an asset model or component model.

To retrieve the active version of an asset model or component model, use the DescribeAssetModel operation with the `assetModelVersion` parameter.

> **Tip**
>
> The AWS CLI defines component models as a type of asset model. Therefore, you use the same DescribeAssetModel operation for both types of model. The `assetModelType` field in the response indicates whether it's an `ASSET_MODEL` or a `COMPONENT_MODEL`.

To retrieve the active version of an asset model or component model (AWS CLI)

- Run the following command to describe the model. Replace *`asset-model-id`* with the ID or the external ID of the asset model or component model. The external ID is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id --asset-model-version ACTIVE
```

  The operation returns a response with the model's details. The response contains an `assetModelStatus` object with the following structure.

```
{
    ...
```

```
    "assetModelName": "string",
    "assetModelProperties": [ ... ],
    ...,
    "assetModelVersion": "string"
}
```

Custom composite models (components)

When you're modeling an especially complex industrial asset, such as a complicated piece of machinery that has many parts, it can become a challenge to keep your asset models organized and maintainable.

In such cases, you can add custom composite models, or components if you're using the console, to your existing asset models and component models. These help you stay organized by grouping related properties and re-using subcomponent definitions.

There are two types of custom composite models:

- **Inline** custom composite models define a set of grouped properties that apply to the asset model or component model to which the custom composite model belongs. You use them to group related properties. They consists of a name, a description, and a set of asset model properties. They are not reusable.
- **Component-model-based** custom composite models reference a component model that you want to include in your asset model or component model. You use them to include standard subassemblies in your model. They consist of a name, a description, and the ID of the component model it references. They have no properties of their own; the referenced component model provides its associated properties to any created assets.

The following sections illustrate how to use custom composite models in your designs.

Topics

Inline custom composite models

Inline custom composite models provide a way to organize your asset model by grouping related properties.

For example, suppose you want to model a robot asset. The robot includes a servomotor, a power supply, and a battery. Each of those constituent parts has its own properties that you want to include in the model. You might define an asset model called `robot_model` that has properties such as the following.

- `robot_model`
  - `servo_status` *(integer)*
  - `servo_position` *(double)*
  - `powersupply_status` *(integer)*
  - `powersupply_temperature` *(double)*
  - `battery_status` *(integer)*
  - `battery_charge` *(double)*

However, in some cases, there might be many subassemblies, or the subassemblies themselves might have many properties. In these cases, there might be so many properties that they become cumbersome to reference and maintain in a single flat list at the model root, like in the preceding example.

To deal with such situations, you can use an inline custom composite model to group properties. An inline custom composite model is a custom composite model that defines its own properties. For example, you could model your robot like the following.

- `robot_model`
  - `servo`
    - `status` *(integer)*
    - `position` *(double)*
  - `powersupply`
    - `status` *(integer)*
    - `temperature` *(double)*

- `battery`
  - `status` *(integer)*
  - `charge` *(double)*

In the preceding example, `servo`, `powersupply`, and `battery` are the names of inline custom composite models defined within the `robot_model` asset model. Each of these composite models then defines properties of its own.

> **Note**
>
> In this case, each custom composite model defines its own properties, so that all the properties are part of the asset model itself (`robot_model` in this case). These properties aren't shared with any other asset models or component models. For example, if you created some other asset model that also had an inline custom composite model called `servo`, then making a change to the `servo` within `robot_model` wouldn't affect the other asset model's `servo` definition.
> If you want to implement such sharing (for example, to have only one definition for a servo, which all your asset models can share), you would create a component model for it instead, and then create **component-model-based** composite models that reference it. See the following section for details.

For information about how to create inline custom composite models, see Create custom composite models (components).

Component-model-based custom composite models

You can create a component model in AWS IoT SiteWise to define a standard, reusable sub-assembly. Once you have created a component model, you can add references to it in your other asset models and component models. You do this by adding a **component-model-based custom composite model** to any model where you want to reference the component. You can add references to your component from many models, or multiple times within the same model.

In this way, you can avoid duplicating the same definitions across models. It also simplifies maintaining your models, because any changes you make to a component model will be reflected across all asset models that use it.

For example, suppose that your industrial installation has many types of equipment that all use the same kind of servo motor. Some of them have many servo motors in a single piece of equipment. You create an asset model for each equipment type, but you don't want to duplicate the definition of `servo` every time. You want to model it just once and use it in your various asset models. If you later make a change to the definition of `servo`, it will be updated across all your models and assets.

To model the robot from the previous example in this way, you could define servo motors, power supplies, and batteries as component models, like this.

- `servo_component_model`
  - `status` *(integer)*
  - `position` *(double)*

- `powersupply_component_model`
  - `status` *(integer)*
  - `temperature` *(double)*

- `battery__component_model`
  - `status` *(integer)*
  - `charge` *(double)*

You could then define asset models, such as `robot_model`, that reference these components. Multiple asset models can reference the same component model. You can also reference the same component model multiple times in one asset model, such as if your robot has multiple servomotors in it.

- `robot_model`
  - `servo1` (*reference:* `servo_component_model`)
  - `servo2` (*reference:* `servo_component_model`)
  - `servo3` (*reference:* `servo_component_model`)

- `powersupply` (*reference:* `powersupply_component_model`)
- `battery` (*reference:* `battery_component_model`)

For information about how to create component models, see Create component models.

For information about how to reference your component models in other models, see Create custom composite models (components).

Use paths to reference custom composite model properties

When you create a property on an asset model, component model, or custom composite model, you can reference it from other properties that use its value, such as transforms and metrics.

AWS IoT SiteWise provides different ways for you to reference your property. The simplest way is often to use its property ID. However, if the property you want to reference is on a custom composite model, you may find it more useful to reference it by *path* instead.

A path is an ordered sequence of *path segments* that specifies a property in terms of its position among the nested composite models within an asset model and composite model.

Obtain property paths

You can get a property's path from the `path` field of its AssetModelProperty.

For example, suppose you have an asset model `robot_model` that contains a custom composite model `servo`, which has a property `position`. If you call DescribeAssetModelCompositeModel on `servo`, then the `position` property would list a `path` field that looks like this:

```
"path": [
    {
       "id": "asset model ID",
       "name": "robot_model"
    },
    {
       "id": "composite model ID",
       "name": "servo"
    },
    {
       "id": "property ID",
```

```
        "name": "position"
    }
]
```

Using property paths

You can use a property path when you define a property that references other properties, such as a transform or metric.

A property uses a *variable* to reference another property. For more information about working with variables, see Use variables in formula expressions.

When you define a variable to reference a property, you can use either the property's ID or its path.

To define a variable that uses the path of the referenced property, specify the `propertyPath` field of its value.

For example, to define an asset model that has a metric that references a property by using a path, you could pass a payload like this to CreateAssetModel:

```
{
    ...
    "assetModelProperties": [
        {
            ...
            "type": {
                "metric": {
                    ...
                    "variables": [
                        {
                            "name": "variable name",
                            "value": {
                                "propertyPath": [
                                    path segments
                                ]
                            }
                        }
                    ],
                    ...
                }
            },
            ...
```

```
        },
        ...
    ],
    ...
}
```

Asset model interfaces

AWS IoT SiteWise interfaces set standards across different asset models. They define a common structure that ensures consistency while allowing for variations in implementation.

Interfaces share the same structure as asset models (properties, composite models, and hierarchies) but you cannot create assets directly from them. Instead, interfaces are applied to existing asset models to ensure standardization. Component models are not supported for interfaces.

Using interfaces provides several benefits:

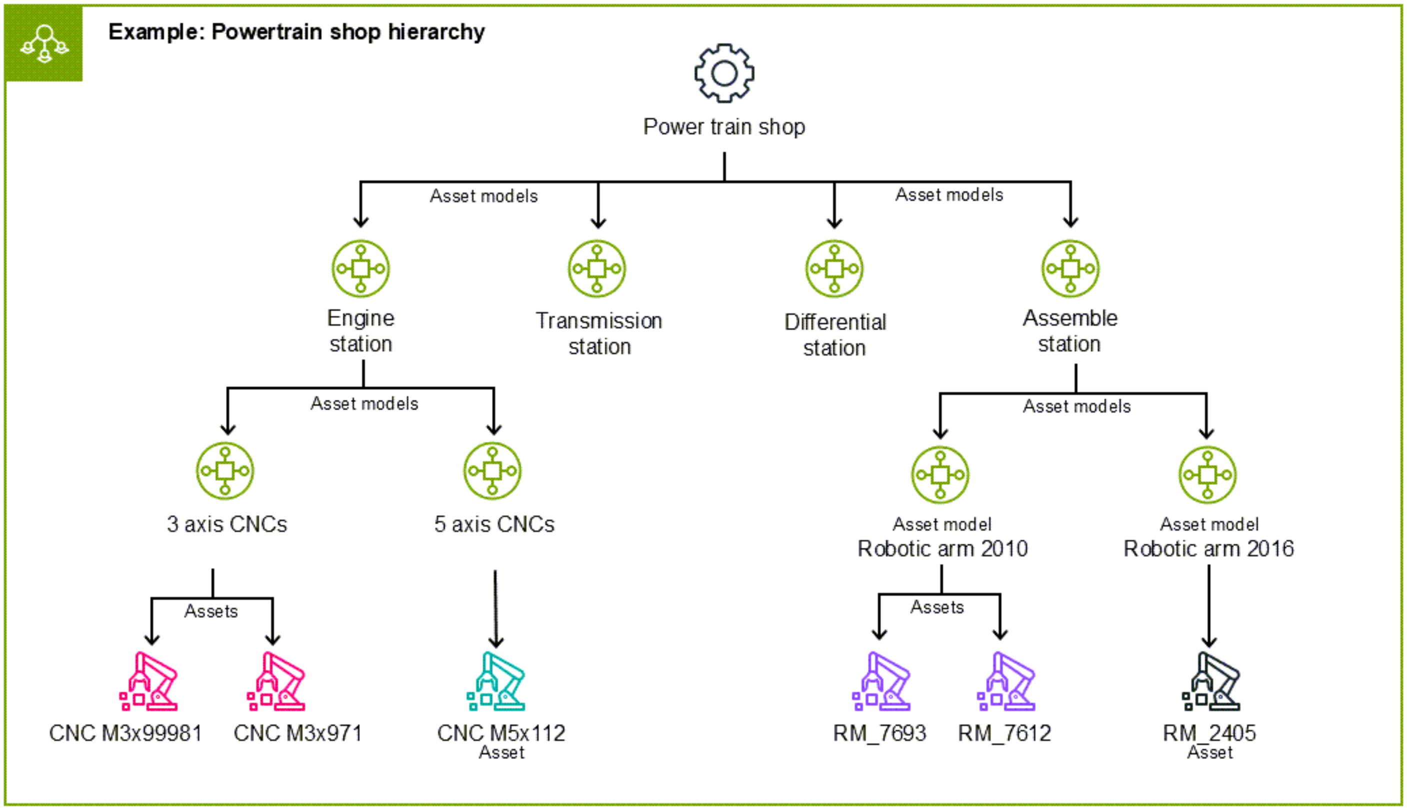
- Standardized properties and metrics across different asset model variations
- Simplified metric definitions at the interface level
- More efficient management of complex asset hierarchies
- Independent property lifecycle management for each asset model variation
- Enhanced cross-team collaboration where operations teams focus on physical asset representation while data teams establish standards across equipment

We recommend creating your asset models first to model your real-world industrial equipment. Every equipment type, with their own set of properties, can be represented by their own asset models.

Asset model standardization use case

Interfaces help standardize properties across different asset models while preserving their unique characteristics.

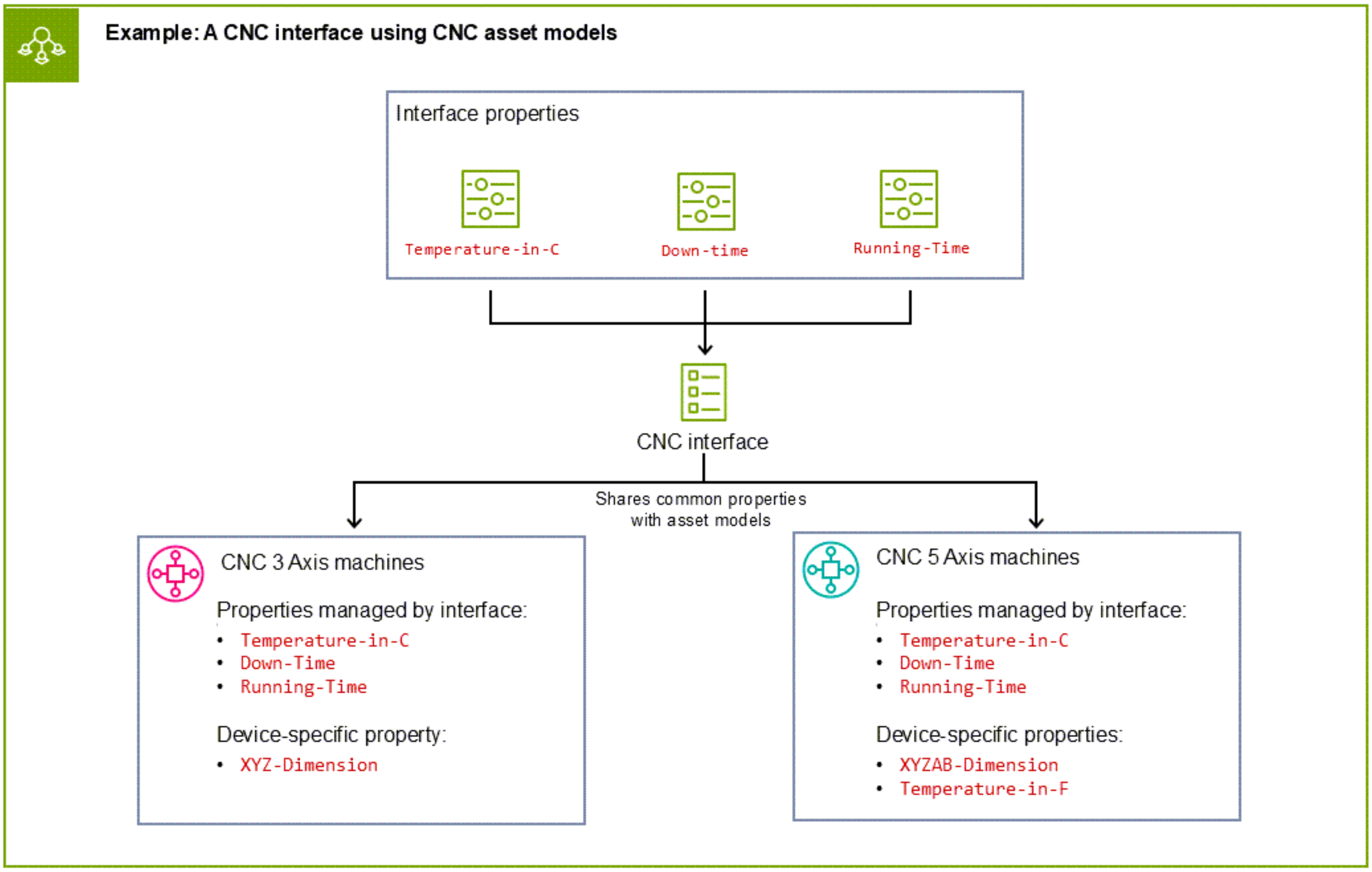
For example, there are four stations in a powertrain shop: engine, transmission, differential, and assembly. Each station contains various equipment types. For example, the engine station includes CNC machines, but they differ in specifications: some are 3-axis, while others are 5-axis.



However, interfaces let you create standards for commonalities seen in the CNC machines. You can use the repeatable properties in an interface rather than create asset models for each property.

For example, you can:

1. Create separate asset models for each category of machine types. In this example, that's the "CNC 3 Axis machines" and the "CNC 5 Axis machines."
2. Define a standard interface with common properties and metrics. In this example, `Temperature-in-C`, `Down-time`, and `Running-time` are all common properties that apply to both CNC machines.
3. Apply this interface to all CNC machine asset models, still allowing for device-specific properties on the individual asset models.



You can also define availability metrics at the interface level. For example, `Avail = avg(Down-time, Running-time)` calculates the availability based on the down time and running time values.

Using interfaces simplifies your asset model management by ensuring consistent property definitions and metrics across applicable equipment while maintaining the unique characteristics of each machine type.

Structure and components

Interfaces include the same property types as asset models: attributes, measurements, transforms, and metrics. When overlayed on an asset model, you map existing properties to their interface counterparts. Unmapped interface properties are automatically created in the asset model.

Interface hierarchies define rollup metrics, while asset model hierarchies enable asset associations. When you use an interface, the service will automatically map asset model hierarchies to interface hierarchies when computing rollup metrics. After applying an interface, rollup metrics are defined through the interface hierarchy rather than the asset model's own hierarchy.

Considerations

When working with interfaces, keep these considerations in mind:

- Asset model and interface properties can be automatically mapped by name or manually mapped. Hierarchies are automatically mapped by the service when computing rollup metrics.
- You cannot define additional metrics in the linked asset model that use interface metrics as inputs.
- An asset model can only be linked to one interface. However, you can have multiple asset models applied to the same interface.
- Alarms are not supported in interfaces.
- Component models are not supported for interfaces.

Topics

Understand the interface-asset model relationship

Interfaces and asset models work together in a complementary relationship:

Interfaces vs. Asset Models

Aspect	Interfaces	Asset Models
Purpose	Define standards and applies consistency	Represent physical or logical assets
Asset Creation	Cannot create assets directly	Used to create assets
Properties	Define standard properties that must be implemented in models	Can have interface-applied and unique properties

Aspect	Interfaces	Asset Models
Metrics	Define standard calculations	Implement interface metrics and can have additional metrics
Hierarchies	Define data computation hierarchi cal relationships for rollup metrics	Define physical hierarchical relations hips for asset associations

When you apply an interface to an asset model:

- The asset model must map all properties defined in the interface.
- Property mappings define how interface properties correspond to asset model properties.
- Mapped asset model properties must remain synchronized with their corresponding interface properties and cannot be modified in a way that would cause inconsistency between the two.
- Unmapped interface properties are automatically created in the asset model.
- The asset model can have additional properties beyond those defined in the interface.
- The asset model implements interface metrics. Changes to interface metrics propagate to all asset models using the interface.
- Interface hierarchies are used for computing rollup metrics. Asset model hierarchies can be defined independently, and the service will automatically map them when computing rollup metrics.

This relationship ensures standardization while allowing for the flexibility needed to represent diverse equipment types.

Standardize existing asset models

While interfaces are valuable when designing new asset models from scratch, they're equally powerful for standardizing existing asset models that may have evolved independently over time.

When working with existing asset models, you can apply interfaces to standardize metrics and properties:

1. Identify common metrics and properties across your existing asset models
2. Create an interface that defines these standard properties and metrics
3. Apply the interface to your existing asset models using property mapping

4. Use rollup metrics to aggregate data across your asset hierarchy

For example, if you have existing CNC machine asset models with different property names but similar data, like `temp_celsius`, `temperature_c`, `machine_temp`), you can:

1. Create a `CNC-INTERFACE` with a standardized `Temperature-in-C` property
2. Apply this interface to each CNC asset model, mapping the existing temperature properties to the interface's `Temperature-in-C` property
3. Define rollup metrics in the interface that calculate statistics across all machines (e.g., average temperature)

This approach allows you to maintain your existing asset models while gaining the benefits of standardization and simplified metrics calculation.

Hierarchy relationships

Interface hierarchy

Defines relationships for calculating and aggregating data across different interfaces. For example, in a factory setting, an interface hierarchy could connect temperature-monitoring interfaces at different levels to calculate average temperatures. For example: machine, production line, and facility. When you define a rollup metric like `AverageTemperature`, the interface hierarchy determines how that metric aggregates data from lower levels to higher levels.

Asset model hierarchy

Represents the actual physical or logical structure of your assets. For instance, a CNC machine asset model might be part of a production line asset model, which in turn belongs to a factory asset model. This hierarchy reflects real-world relationships and helps organize assets in a way that matches their physical arrangement or business structure. When combined with interface hierarchies, asset model hierarchies help the system understand which assets should be included in rollup calculations.

These two hierarchy types work together: interface hierarchies define how to compute aggregated metrics, while asset model hierarchies define which specific assets should be included in those calculations.

Interface metrics and rollup calculations

Interfaces excel at defining standardized metrics that can be applied across different asset models. This is particularly valuable for rollup metrics that aggregate data from multiple assets.

When you define metrics in an interface, they're automatically applied to all asset models that implement the interface. The metrics can reference properties defined in the interface, use aggregation functions to calculate statistics across assets, and ensure consistent calculations across all implementing asset models For example, you can define an availability metric in an interface that calculates the ratio of running time to total time:

```
{
  "name": "Availability",
  "dataType": "DOUBLE",
  "type": {
    "metric": {
      "expression": "Running-time / (Running-time + Down-time) * 100",
      "variables": [
        {
          "name": "Running-time",
          "value": {
            "propertyId": "${Running-time}"
          }
        },
        {
          "name": "Down-time",
          "value": {
            "propertyId": "${Down-time}"
          }
        }
      ],
      "window": {
        "tumbling": {
          "interval": "1h"
        }
      }
    }
  },
  "unit": "Percent"
}
```

When this interface is applied to multiple asset models, the availability metric is calculated consistently for all of them, even if the underlying property names differ (thanks to property mapping).

For more information about defining metrics and using aggregation functions, see [Aggregate data from properties and other assets (metrics)](#).

Rollup metrics with interfaces

Interfaces can also define rollup metrics that aggregate data across assets in a hierarchy. When you define a hierarchy in an interface and apply it to an asset model, you can create metrics that aggregate data from child assets.

For example, you can define a metric that calculates the average temperature across all CNC machines in a factory:

```
{
  "name": "AverageTemperature",
  "dataType": "DOUBLE",
  "type": {
    "metric": {
      "expression": "avg(Temperature-in-C)",
      "variables": [
        {
          "name": "Temperature-in-C",
          "value": {
            "propertyId": "${Temperature-in-C}",
            "hierarchyId": "${CNC-machines}"
          }
        }
      ],
      "window": {
        "tumbling": {
          "interval": "1h"
        }
      }
    }
  },
  "unit": "Celsius"
}
```

This metric uses the `avg()` aggregation function to calculate the average temperature across all CNC machines in the hierarchy. The `hierarchyId` parameter specifies which hierarchy to use for the aggregation.

When this interface is applied to an asset model, the rollup metric automatically aggregates data from all child assets that match the hierarchy mapping.

Create an interface

You can create interfaces using either the AWS IoT SiteWise console or the AWS CLI.

Console

1. Navigate to the AWS IoT SiteWise console and choose **Models** from the navigation pane.
2. Choose **Create interface**.
3. Enter a unique **Name** and optional **Description** for your interface. You can also optionally add an **External ID** of you choosing.
4. Add properties to your interface. You can add attributes, measurements, transforms, and metrics just like with asset models. For more information, see Create an asset model (console).
5. Choose **Create interface** to create the interface.
6. If you have hierarchies to define parent-child relationships between interfaces, choose **Add hierarchy** and enter relevant details.

AWS CLI

To create an interface, use the `CreateAssetModel` operation with the `assetModelType` parameter set to `INTERFACE`:

```
aws iotsitewise create-asset-model \
  --asset-model-name "CNC-INTERFACE" \
  --asset-model-description "Standard interface for CNC machines" \
  --asset-model-type "INTERFACE" \
  --asset-model-properties '[
    {
      "name": "Temperature-in-C",
      "dataType": "DOUBLE",
      "type": {
        "measurement": {}
```

```
      },
      "unit": "Celsius"
    },
    {
      "name": "Down-time",
      "dataType": "DOUBLE",
      "type": {
        "measurement": {}
      },
      "unit": "Minutes"
    },
    {
      "name": "Running-time",
      "dataType": "DOUBLE",
      "type": {
        "measurement": {}
      },
      "unit": "Minutes"
    },
    {
      "name": "Availability",
      "dataType": "DOUBLE",
      "type": {
        "metric": {
          "expression": "Running-time / (Running-time + Down-time) * 100",
          "variables": [
            {
              "name": "Running-time",
              "value": {
                "propertyId": "${Running-time}"
              }
            },
            {
              "name": "Down-time",
              "value": {
                "propertyId": "${Down-time}"
              }
            }
          ],
          "window": {
            "tumbling": {
              "interval": "1h"
            }
          }
```

```
        }
      },
      "unit": "Percent"
    }
  ]'
```

Apply an interface to an asset model

When applying an interface to an asset model, you map asset model properties and hierarchies to their interface counterparts. For unmapped interface properties, corresponding properties are automatically created in the asset model. After linking, the service prevents changes to the asset model that would violate interface standards.

You can add one asset model to an interface at a time. However, multiple asset models can be linked to a single interface.

Console

1. Navigate to the AWS IoT SiteWise console and choose **Models** from the navigation pane.
2. Select the asset model to which you want to apply an interface.
3. Choose **Link asset model** in the **Link asset models** section. This brings up the **Link interface** page.
4. In the **Asset models and interfaces** section, select an asset model from the **Select a model to link** dropdown menu.
5. In the **Property mappings** section, map each interface property to an existing asset model property or create a new property. AWS IoT SiteWise automatically links properties with matching names in the asset model and interface.
6. Review the property mappings and choose **Link interface**.

AWS CLI

To apply an interface to an asset model, use the `PutAssetModelInterfaceRelationship` operation:

```
aws iotsitewise put-asset-model-interface-relationship \
  --asset-model-id "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --interface-asset-model-id "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE" \
```

```
  --property-mapping-configuration '{
    "createMissingProperty": true,
    "matchByPropertyName": true,
    "overrides": [
      {
        "assetModelPropertyId": "a1b2c3d4-5678-90ab-cdef-44444EXAMPLE",
        "interfaceAssetModelPropertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE"

      }
    ]
  }'
```

To retrieve information about an interface relationship, use the `DescribeAssetModelInterfaceRelationship` operation:

```
aws iotsitewise describe-asset-model-interface-relationship \
  --asset-model-id "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --interface-asset-model-id "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE"
```

To list all asset models that have a specific interface applied to them, use the `ListInterfaceRelationships` operation:

```
aws iotsitewise list-interface-relationships \
  --interface-asset-model-id "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE" \
  --max-results 10
```

To delete an interface relationship, use the `DeleteAssetModelInterfaceRelationship` operation:

```
aws iotsitewise delete-asset-model-interface-relationship \
  --asset-model-id "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
  --interface-asset-model-id "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE"
```

Manage interfaces, linked asset models, and properties

After creating interfaces and linking them to asset models, you can manage relationships, edit, and delete interfaces through the console or AWS CLI.

Modify an interface and asset model relationship

To change an interface's relationship to an asset model, do the following in either the AWS IoT SiteWise console or through AWS CLI:

Console

1. Navigate to the AWS IoT SiteWise console and choose **Models** from the navigation pane.
2. Select the interface you want modify.
3. Choose the asset model to modify and edit it.

   You can follow the Apply an interface to an asset model instructions to link a different asset model.
4. Choose **Apply interface** to save your changes.

AWS CLI

To edit an interface and asset model relationship, use the `PutAssetModelInterfaceRelationship` action. Replace *`your-asset-model-id`* and *`your-interface-asset-model-id`* with your own values. For more information, see PutAssetModelInterfaceRelationship in the *AWS IoT SiteWise API Reference*.

```
aws iotsitewise put-asset-model-interface-relationship \
    --asset-model-id your-asset-model-id \
    --interface-asset-model-id your-interface-asset-model-id
```

Modify an interface property mapping

To change an interface's property, do the following in either the AWS IoT SiteWise console or through AWS CLI:

Console

1. Navigate to the AWS IoT SiteWise console and choose **Models** from the navigation pane.
2. Select the interface for which you want modify property mappings. The **Edit property mappings** page appears.
3. In the **Property mappings** section, filter the list to find the appropriate property mappings.

4. Change the properties using the **Model property** column.

AWS CLI

To edit an interface and asset model relationship, use the `PutAssetModelInterfaceRelationship` action. Replace *`your-asset-model-id`* and *`your-interface-asset-model-id`* with your own values. For more information, see PutAssetModelInterfaceRelationship in the *AWS IoT SiteWise API Reference*.

```
aws iotsitewise put-asset-model-interface-relationship \
    --asset-model-id your-asset-model-id \
    --interface-asset-model-id your-interface-asset-model-id \
```

List interfaces linked to an asset model

To get a list of interfaces applied to an asset model, do the following in either the AWS IoT SiteWise console or through AWS CLI:

Console

1. Navigate to the AWS IoT SiteWise console and choose **Models** from the navigation pane.
2. In the **Models** section, choose the appropriate asset model or interface. You can view a list of either applied interfaces or linked asset models on the model's corresponding details page.
   - When viewing a particular interface, see the **Linked asset models** section.
   - When viewing a particular asset model, see the **Applied interfaces** section.

AWS CLI

To list interfaces, you can use the `ListInterfaceRelationships` operation. Replace *`your-interface-asset-model-id`* with your own value. For more information, see ListInterfaceRelationships in the *AWS IoT SiteWise API Reference*.

```
aws iotsitewise list-interface-relationships \
    --interface-asset-model-id your-interface-asset-model-id \
    [--next-token your-next-token] \
    [--max-results 20]
```

View the details of an interface and asset model relationship

To see the details of an interface applied to an asset model, do the following in either the AWS IoT SiteWise console or through AWS CLI:

Console

View the details of applied interfaces and linked asset models.

1. Navigate to the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/) and choose **Models** from the navigation pane.
2. In the **Models** section, search for the appropriate asset model or interface. Select the model or interface's **Name** to open up a page containing more details.

AWS CLI

To view interface details for an interface and asset model relationship, use the `DescribeAssetModelInterfaceRelationship` action. Replace *`your-asset-model-id`* and *`your-interface-asset-model-id`* with your own values. For more information, see [DescribeAssetModelInterfaceRelationship]() in the *AWS IoT SiteWise API Reference*.

```
aws iotsitewise describe-asset-model-interface-relationship \
    --asset-model-id your-asset-model-id \
    --interface-asset-model-id your-interface-asset-model-id
```

Remove an interface applied to an asset model

To remove an interface applied to an asset model, do the following in either the AWS IoT SiteWise console or through AWS CLI:

Console

We recommend removing an interface through the asset model. You can also delete an interface or unlink an interface through a particular interface's page.

1. Navigate to the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/) and choose **Models** from the navigation pane.
2. Select the appropriate asset model from which to remove the interface relationship.
3. Choose **Unlink asset model**.

AWS CLI

To remove an interface relationship from an asset model, you can use the `DeleteAssetModelInterfaceRelationship` action. Replace *`your-interface-asset-model-id`* with your own value. For more information, see DeleteAssetModelInterfaceRelationship in the *AWS IoT SiteWise API Reference*.

```
aws iotsitewise delete-asset-model-interface-relationship \
    --asset-model-id your-asset-model-id \
    --interface-asset-model-id your-interface-asset-model-id
```

Additional interface examples

Here are additional examples of how interfaces can be used in different industrial scenarios:

Energy generation equipment

A power generation company can use interfaces to standardize metrics across different types of generation equipment:

```
{
  "assetModelName": "GENERATOR-INTERFACE",
  "assetModelDescription": "Standard interface for power generators",
  "assetModelType": "INTERFACE",
  "assetModelProperties": [
    {
      "name": "ActivePower",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
      "unit": "MW"
    },
    {
      "name": "ReactivePower",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
      "unit": "MVAR"
    },
    {
      "name": "Frequency",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
```

```
        "unit": "Hz"
      },
      {
        "name": "PowerFactor",
        "dataType": "DOUBLE",
        "type": {
          "metric": {
            "expression": "cos(atan(ReactivePower / ActivePower))",
            "variables": [
              {
                "name": "ActivePower",
                "value": { "propertyId": "${ActivePower}" }
              },
              {
                "name": "ReactivePower",
                "value": { "propertyId": "${ReactivePower}" }
              }
            ],
            "window": { "tumbling": { "interval": "5m" } }
          }
        },
        "unit": "None"
      }
    ]
}
```

This interface can be applied to various generator asset models (gas turbines, steam turbines, wind turbines) to ensure consistent power metrics across the fleet.

Water treatment facilities

A water utility can use interfaces to standardize monitoring across treatment plants:

```
{
  "assetModelName": "WATER-QUALITY-INTERFACE",
  "assetModelDescription": "Standard interface for water quality monitoring",
  "assetModelType": "INTERFACE",
  "assetModelProperties": [
    {
      "name": "pH",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
      "unit": "pH"
```

```
    },
    {
      "name": "Turbidity",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
      "unit": "NTU"
    },
    {
      "name": "DissolvedOxygen",
      "dataType": "DOUBLE",
      "type": { "measurement": {} },
      "unit": "mg/L"
    },
    {
      "name": "QualityIndex",
      "dataType": "DOUBLE",
      "type": {
        "metric": {
          "expression": "(pH >= 6.5 && pH <= 8.5 ? 100 : 50) * (Turbidity < 1 ? 1 :
 0.8) * (DissolvedOxygen > 5 ? 1 : 0.7)",
          "variables": [
            {
              "name": "pH",
              "value": { "propertyId": "${pH}" }
            },
            {
              "name": "Turbidity",
              "value": { "propertyId": "${Turbidity}" }
            },
            {
              "name": "DissolvedOxygen",
              "value": { "propertyId": "${DissolvedOxygen}" }
            }
          ],
          "window": { "tumbling": { "interval": "1h" } }
        }
      },
      "unit": "Score"
    }
  ]
}
```

This interface ensures that water quality is measured consistently across all treatment facilities, regardless of their specific equipment configurations.

Hierarchical interfaces

Interfaces can be organized hierarchically to support aggregate metrics at different levels of your operation:

1. **Equipment-level interface** (for example, `PUMP-INTERFACE`)
   - Properties: Flow rate, pressure, power consumption, vibration
   - Metrics: Efficiency, health score
2. **Process-level interface** (for example, `PUMPING-STATION-INTERFACE`)
   - Properties: Total flow, average pressure, total power
   - Metrics: Station efficiency, operational cost per volume
   - Hierarchy: Contains `PUMP-INTERFACE` equipment
3. **Facility-level interface** (for example, `WATER-FACILITY-INTERFACE`)
   - Properties: Facility throughput, energy usage, chemical usage
   - Metrics: Facility efficiency, cost per unit volume, carbon footprint
   - Hierarchy: Contains `PUMPING-STATION-INTERFACE` processes

This hierarchical approach allows metrics to be calculated at each level while maintaining consistency across your entire operation.

Set up AWS IoT SiteWise object IDs

AWS IoT SiteWise defines various types of persistent objects, such as assets, asset models, properties, and hierarchies. All such objects have unique identifiers that you can use to retrieve, update, and delete them.

AWS IoT SiteWise has different options for customers for ID creation. AWS IoT SiteWise generates one for you by default at object creation time. Users can also provide their own IDs to your objects.

Topics

- Use external IDs

Work with object UUIDs

Every persistent object in AWS IoT SiteWise has a UUID to identify it. For example, asset models have an asset model ID, assets have an asset ID, and so on. This ID is assigned at the time that you create the object, and remains unchanged for the object's lifetime.

When you create a new object, AWS IoT SiteWise generates a unique ID for you by default. You can also provide your own ID at creation time in UUID format.

> **Note**
>
> UUIDs **must** be globally unique within the AWS Region where it's created, and for the same object type. When AWS IoT SiteWise auto-generates an ID for you, it's always unique. If you choose your own ID, make sure that it's unique.

For example, if you create a new asset model by calling CreateAssetModel, you can provide your own UUID in the optional `assetModelId` field of the request.

By contrast, if you omit `assetModelId` from the request, AWS IoT SiteWise generates a UUID for the new asset model.

Use external IDs

To define your own ID in some format other than UUID, you can assign an *external ID*. For example, you can do this if you reuse an ID that you're using in a system that's not AWS, or to be more human-readable. External IDs have a more flexible format. You can use them to reference your objects in AWS IoT SiteWise API operations where you would otherwise use the UUID.

Like the UUIDs, each external ID must be unique within its context. For example, you can't have two asset models with the same external ID. Also, like the UUIDs, an object can only have one external ID in its lifetime, which can't change.

Differences between external IDs and UUIDs

External IDs differ from UUIDs in the following ways:

- Every object has a UUID, but external IDs are optional.
- AWS IoT SiteWise never generates external IDs. You provide these yourself.
- If the object does not already have one, you can assign an external ID at any time.

Format of external IDs

A valid external ID has the following properties:

- Is between 2 and 128 characters long.
- The first and last characters must be alphanumeric (A-Z, a-z, 0-9).
- Characters other than first and last must either be alphanumeric, or else one of the following: `_-.:`

For example, an external ID must conform to the following regular expression:

`[a-zA-Z0-9][a-zA-Z0-9_\-.:]*[a-zA-Z0-9]+`

Reference objects with external IDs

In many places that you could reference an object using its UUID, you can use its external ID instead, if it has one. To do so, append the external ID to the string `externalId:`.

For example, suppose you have an asset model whose UUID (asset model ID) is `a1b2c3d4-5678-90ab-cdef-11111EXAMPLE`, which also has the external ID `myExternalId`. Call DescribeAssetModel to get details about it. You could use either of the following as the value of `assetModelId`:

- With the asset model ID (UUID) itself: `a1b2c3d4-5678-90ab-cdef-11111EXAMPLE`
- With the external ID: `externalId:myExternalId`

```
aws iotsitewise describe-asset-model --asset-model-id a1b2c3d4-5678-90ab-
cdef-11111EXAMPLE
aws iotsitewise describe-asset-model --asset-model-id externalId:myExternalId
```

> **Note**
>
> The `externalId:` prefix is not, itself, part of the external ID. You only need to provide the prefix when you supply an external ID to an API operation that accepts either UUIDs or external IDs. For example, supply the prefix when you query or update an existing object. When you define an external ID for an object, such as when you create an asset model, don't include the prefix.

You can use external IDs in place of UUIDs in this fashion for many API operations in AWS IoT SiteWise, but not all. For example, the GetAssetPropertyValue, **must** use UUIDs; it doesn't support external ID usage.

To determine whether a particular API operation supports this usage, consult the API Reference.

Create asset models, component models, and interfaces for AWS IoT SiteWise

AWS IoT SiteWise asset models, component models, and interfaces drive standardization of your industrial data. Asset models define the overall asset, such as a wind turbine or a manufacturing line. Component models represent the individual components that make up the asset, such as blades, generators, or sensors. Interfaces enforce standards across different asset models. By creating these models, you can organize and structure your asset data in a way that reflects the real-world relationships and hierarchies of your industrial equipment, making it easier to monitor, analyze, and maintain.

An asset model or component model contains a name, description, asset properties, and (optionally) custom composite models that group properties together, or that reference component models for subassemblies.

In AWS IoT SiteWise, you can create asset models, component models, and interfaces to represent the structure and properties of your industrial assets and their components.

- You use an **asset model** to create assets. In addition to the features listed above, an asset model can also contain hierarchy definitions that define relationships among assets.
- A **component model** represents a subassembly within an asset model or another component model. When you create a component model, you can add references to it in asset models and in other component models. However, you can't create assets directly from component models.

- An **interface** enforces standards across different asset models. Interfaces define common properties, metrics, and hierarchies that must be implemented by asset models. You can't create assets directly from interfaces, but they help ensure consistency across similar asset types.

After you create an asset model or component model, you can create custom composite models for it to group properties together or to reference existing component models. You can also link interfaces to asset models to enforce standardization.

For details about how to create asset models, component models, and interfaces, see the following sections.

Topics

Create asset models in AWS IoT SiteWise

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.
> - Asset models can't contain other asset models. If you must define a model that you can reference as a subassembly within another model, you should create a component--> model instead. For more information, see Create component 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

Create 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 auto completion, 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 asset 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 an **External ID** for the model. This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

    c. (Optional) Add **Measurement definitions** for the model. Measurements represent data streams from your equipment. For more information, see Define data streams from equipment (measurements).

    d. (Optional) Add **Transform definitions** for the model. Transforms are formulas that map data from one form to another. For more information, see Transform data (transforms).

    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 Aggregate data from properties and other assets (metrics).

f. (Optional) Add **Hierarchy definitions** for the model. Hierarchies are relationships between assets. For more information, see Define asset model hierarchies.

g. (Optional) Add tags for the asset model. For more information, see Tag your AWS IoT SiteWise resources.

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.

5. (Optional) After you create your asset model, you can configure your asset model for the edge. For more information about SiteWise Edge, see Configure edge capabilities on AWS IoT SiteWise Edge.

   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 Set up an OPC UA source in SiteWise Edge.

   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 Set up an OPC UA source in SiteWise Edge.

   d. Choose **Save**. On the model page, your **Edge configuration** should now be **Configured**.

Create an asset model (AWS 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.

```
{
  "assetModelType": "ASSET_MODEL",
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": Array of AssetModelProperty,
  "assetModelHierarchies": Array of AssetModelHierarchyDefinition
}
```

To create an asset model (AWS CLI)

1. Create a file called `asset-model-payload.json` and then copy the following JSON object into the file.

   ```
   {
     "assetModelType": "ASSET_MODEL",
     "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 asset models and component models in your account in this AWS Region.

b. (Optional) Enter an external ID (`assetModelExternalId`) for the asset model. This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

c. (Optional) Enter a description (`assetModelDescription`) for the asset model, or remove the `assetModelDescription` key-value pair.

d. (Optional) Define asset properties (`assetModelProperties`) for the model. For more information, see Define data properties.

e. (Optional) Define asset hierarchies (`assetModelHierarchies`) for the model. For more information, see Define asset model hierarchies.

f. (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 Monitor data with alarms in AWS IoT SiteWise and Define alarms on asset models in AWS IoT SiteWise.

g. (Optional) Add tags (`tags`) for the asset model. For more information, see Tag your AWS IoT SiteWise resources.

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.

4. (Optional) Create custom composite models for your asset model. With custom composite models, you can group properties within the model, or include a subassembly by referencing a component model. For more information, see Create custom composite models (components).

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

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 Use the AWS IoT SiteWise demo.

```
{
  "assetModelType": "ASSET_MODEL",
  "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"
            }
          }
        }
      }
    },
    {
      "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"
              }
            }
          ],
          "window": {
            "tumbling": {
              "interval": "5m"
            }
          }
        }
      }
    }
  ],
```

```
  "assetModelHierarchies": []
}
```

Wind farm asset model

The following asset model represents a wind farm that comprises multiple wind turbines. This asset model defines a hierarchy to the wind turbine model. This lets the wind farm 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 Use the AWS IoT SiteWise demo.

This asset model depends on the Wind turbine asset model. Replace the `propertyId` and `childAssetModelId` values with those from an existing wind turbine asset model.

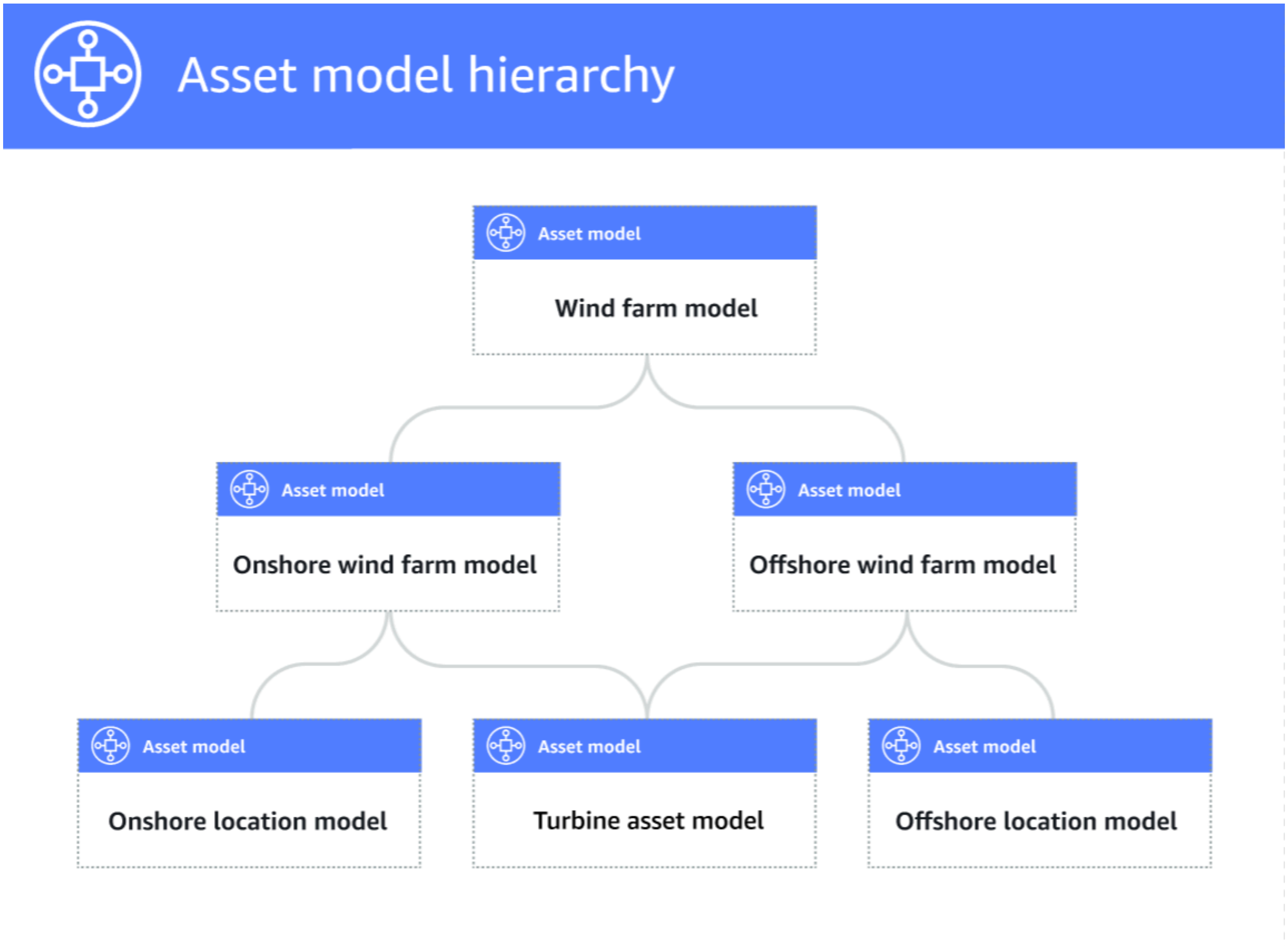
```
{
  "assetModelName": "Wind Farm Asset Model",
  "assetModelDescription": "Represents a wind farm.",
  "assetModelProperties": [
    {
      "name": "Code",
      "dataType": "INTEGER",
      "type": {
        "attribute": {
          "defaultValue": "300"
        }
      }
    },
    {
      "name": "Location",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "Renton"
        }
      }
    },
    {
      "name": "Reliability Manager",
```

```
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "Mary Major"
        }
      }
    },
    {
      "name": "Total Overdrive State Time",
      "dataType": "DOUBLE",
      "unit": "seconds",
      "type": {
        "metric": {
          "expression": "sum(overdrive_state_time)",
          "variables": [
            {
              "name": "overdrive_state_time",
              "value": {
                "propertyId": "ID of Overdrive State Time property in Wind Turbine
 Asset Model",
                "hierarchyId": "Turbine Asset Model"
              }
            }
          ],
          "window": {
            "tumbling": {
              "interval": "5m"
            }
          }
        }
      }
    },
    {
      "name": "Total Average Power",
      "dataType": "DOUBLE",
      "unit": "Watts",
      "type": {
        "metric": {
          "expression": "sum(turbine_avg_power)",
          "variables": [
            {
              "name": "turbine_avg_power",
              "value": {
```

```
                "propertyId": "ID of Average Power property in Wind Turbine Asset
 Model",
                "hierarchyId": "Turbine Asset Model"
              }
            }
          ],
          "window": {
            "tumbling": {
              "interval": "5m"
            }
          }
        }
      }
    }
  ],
  "assetModelHierarchies": [
    {
      "name": "Turbine Asset Model",
      "childAssetModelId": "ID of Wind Turbine Asset Model"
    }
  ]
}
```

Define asset model hierarchies

You can define asset model hierarchies to create logical associations between the asset models in your industrial operation. For example, you can define a wind farm composed of onshore and offshore wind farms. An onshore wind farm contains a turbine and onshore location. An offshore wind farm contains a turbine and offshore location.



When you associate a child asset model to a parent asset model through a hierarchy, the parent asset model's metrics can input data from the child asset model's metrics. You can use asset model hierarchies and metrics to calculate statistics that provide insight to your operation or a subset of your operation. For more information, see [Aggregate data from properties and other assets (metrics)](#).

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 turbines in your wind farms, where all wind turbines are represented by 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.

A parent asset model can be associated with multiple child asset models. For example, if you have an onshore wind farm and an offshore wind farm that are represented by two different asset models, you can associate these asset models with the same parent wind farm asset model.

A child asset model can also be associated with multiple parent asset models. For example, if you have two different types of wind farms, where all wind turbines are represented by the same asset model, you can associate the wind turbine asset model with different wind farm asset models.

> **Note**
>
> When you define an asset model hierarchy, the child asset model must be `ACTIVE` or have a previous `ACTIVE` version. For more information, see Asset and model states.

After you define hierarchical asset models and create assets, you can associate the assets to complete the parent-child relationship. For more information, see Create assets for asset models in AWS IoT SiteWise and Associate and disassociate assets.

Topics

- Define asset model hierarchies (console)
- Define asset hierarchies (AWS CLI)

Define asset model 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.
- **Hierarchy External ID** (Optional) – This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

For more information, see Create an asset model (console).

Define asset hierarchies (AWS 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 or the external 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 model hierarchy that represents a wind farm's relationship to wind turbines. This object is an example of an AssetModelHierarchy. For more information, see Create an asset model (AWS CLI).

```
{
  ...
  "assetModelHierarchies": [
    {
      "name": "Wind Turbines",
      "childAssetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"
    },
  ]
}
```

Create component models

Use AWS IoT SiteWise component models to define subassemblies that you can reference from asset models or other component models. In this way, you can re-use the definition of the component across multiple other models, or multiple times within the same model.

The process of defining a component model is very similar to defining an asset model. Like an asset model, a component model has a name, description, and asset properties. However, component models can't include asset hierarchy definitions, since component models themselves can't be used to create assets directly. Component models also can't define alarms.

For example, you can define a component for a servo motor with motor temperature, encoder temperature, and insulation resistance properties. Then, you can define an asset model for equipment that contains servo motors, such as a CNC machine.

> **Note**
>
> - We recommend that you model your operation starting with the lowest-level nodes. For example, create your servo motor component before you create your CNC machine's asset model. Asset models contain references to existing component models.
> - You can't create an asset directly from a component model. To create an asset that uses your component, you must create an asset model for your asset. Then, you create a custom composite model for it that references your component. For more information about creating asset models, see Create asset models in AWS IoT SiteWise For more information about creating custom composite models, see Create custom composite models (components).

The following sections describe how to use the AWS IoT SiteWise API to create component models.

Topics

- Create a component model (AWS CLI)
- Example component model

Create a component model (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to create a component model.

Use the CreateAssetModel operation to create a component model with properties. This operation expects a payload with the following structure:

```
{
  "assetModelType": "COMPONENT_MODEL",
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": Array of AssetModelProperty,
}
```

To create a component model (AWS CLI)

1. Create a file called `component-model-payload.json` and then copy the following JSON object into the file:

```
{
  "assetModelType": "COMPONENT_MODEL",
  "assetModelName": "",
  "assetModelDescription": "",
  "assetModelProperties": [

  ]
}
```

2. Use your preferred JSON text editor to edit the `component-model-payload.json` file for the following:

   a. Enter a name (`assetModelName`) for the component model, such as **`Servo Motor`** or **`Servo Motor Model`**. This name must be unique across all asset models and component models in your account in this AWS Region.

   b. (Optional) Enter an external ID (`assetModelExternalId`) for the component model. This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

   c. (Optional) Enter a description (`assetModelDescription`) for the asset model, or remove the `assetModelDescription` key-value pair.

   d. (Optional) Define asset properties (`assetModelProperties`) for the component model. For more information, see Define data properties.

   e. (Optional) Add tags (`tags`) for the asset model. For more information, see Tag your AWS IoT SiteWise resources.

3. Run the following command to create a component model from the definition in the JSON file.

```
aws iotsitewise create-asset-model --cli-input-json file://component-model-
payload.json
```

   The operation returns a response that contains the `assetModelId` that you refer to when adding a reference to your component model in an asset model or another component model. The response also contains the state of the model (`assetModelStatus.state`), which is initially `CREATING`. The component model's status is `CREATING` until the changes propagate.

Note

The component model creation process can take up to a few minutes for complex models. To check the current status of your component model, use the DescribeAssetModel operation by specifying the `assetModelId`. After the component model status is `ACTIVE`, you can add references to your component model in asset models or other component models. For more information, see Asset and model states.

4. (Optional) Create custom composite models for your component model. With custom composite models, you can group properties within the model, or to include a subassembly by referencing another component model. For more information, see Create custom composite models (components).

Example component model

This section contains an example component model definition that you can use to create a component model with the AWS CLI and AWS IoT SiteWise SDKs. This component model represents a servo motor that can be used within another piece of equipment, such as a CNC machine.

Topics

- Servo motor component model

Servo motor component model

The following component model represents a servo motor that can be used within equipment such as CNC machines. The servo motor provides various measurements, such as temperatures and electrical resistance. These measurements are available as properties on assets created from asset models that reference the servo motor component model.

```
{
    "assetModelName": "ServoMotor",
    "assetModelType": "COMPONENT_MODEL",
    "assetModelProperties": [
        {
            "dataType": "DOUBLE",
            "name": "Servo Motor Temperature",
```

```
            "type": {
            "measurement": {}
            },
            "unit": "Celsius"
        },
        {
            "dataType": "DOUBLE",
            "name": "Spindle speed",
            "type": {
            "measurement": {}
            },
            "unit": "rpm"
        }
    ]
}
```

Define 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 Define static data (attributes).
- **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 Define data streams from equipment (measurements).
- **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 Transform data (transforms).
- **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 Aggregate data from properties and other assets (metrics).

For more information, see Create asset models in AWS IoT SiteWise.

For an example of how to use measurements, transforms, and metrics to calculate Overall Equipment Effectiveness (OEE), see Calculate OEE in AWS IoT SiteWise.

Topics

- Define static data (attributes)
- Define data streams from equipment (measurements)
- Transform data (transforms)
- Aggregate data from properties and other assets (metrics)
- Use formula expressions

Define 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

- Define attributes (console)
- Define attributes (AWS CLI)

Define 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 Update attribute values.
- **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`.
- **External ID** – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

For more information, see Create an asset model (console).

Define attributes (AWS 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 Update attribute values.
- `dataType` – 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`.
- `externalId` – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

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 Create an asset model (AWS CLI).

```
{
...
"assetModelProperties": [
{
  "name": "Model number",
  "dataType": "STRING",
  "type": {
    "attribute": {
      "defaultValue": "BLT123"
    }
  }
```

```
}
],
...
}
```

Define 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.

Topics

- Define measurements (console)
- Define measurements (AWS CLI)

Define measurements (console)

When you define a measurement for an asset model in the AWS IoT SiteWise console, you specify 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`.
- **External ID** – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

For more information, see Create an asset model (console).

Define measurements (AWS 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 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`.
- `unit` – (Optional) The scientific unit for the property, such as mm or Celsius.
- `externalId` – (Optional) This is a user-defined ID. For more information, see [Reference objects with external IDs] in the *AWS IoT SiteWise User Guide*.

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 [Create an asset model (AWS CLI)].

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 [Manage data streams for AWS IoT SiteWise].

```
{
      ...
      "assetModelProperties": [
      {
          "name": "Temperature C",
          "dataType": "DOUBLE",
          "type": {
              "measurement": {}
          },
          "unit": "Celsius"
      }
  ],
      ...
}
```

Transform 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.

> **Note**
>
> For property updates with the same timestamp, output values may be overwritten by updates from other incoming properties.

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` and `Bad_Count` values. To avoid errors, reset `Good_Count` and `Bad_Count` before the next manufacturing run.
- Use metrics if you want to evaluate expressions only after all variables receive new data points.

Topics

- Define transforms (console)
- Define transforms (AWS CLI)

Define 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.
- **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**.
- **External ID** – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.
- **Formula** – The transform expression. Transform expressions can't use aggregation functions or temporal functions. To open the auto complete feature, start typing or press the down arrow key. For more information, see Use formula expressions.

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 launch transform updates. The same request rate for asset property data API operations applies for transform computation results.
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 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.

For more information, see Create an asset model (console).

Define transforms (AWS 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.
- `unit` – (Optional) The scientific unit for the property, such as mm or Celsius.
- `dataType` – The data type of the transform, which must be `DOUBLE` or `STRING`.
- `externalId` – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.
- `expression` – The transform expression. Transform expressions can't use aggregation functions or temporal functions. For more information, see Use formula expressions.
- `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 launch transform updates. The same request rate for asset property data API operations applies for transform computation results.
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 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.

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 Create an asset model (AWS CLI).

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

```
}
```

Aggregate 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 Define asset model hierarchies.

Metrics can also input data from other properties without aggregating data over each time interval. If you specify an attribute in a formula, AWS IoT SiteWise uses the latest value for that attribute when it computes the formula. If you specify a metric in a formula, AWS IoT SiteWise uses the last value for the time interval over which it computes the formula. This means you can define metrics like `OEE = Availability * Quality * 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 Query asset property aggregates in AWS IoT SiteWise.

Topics

- Define metrics (console)
- Define metrics (AWS CLI)

Define 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.
- **Data type** – The data type of the transform, which can be **Double** or **String**.
- **External ID** – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

- **Formula** – The metric expression. Metric expressions can use aggregation functions to input data from a property for all associated assets in a hierarchy. Start typing or press the down arrow key to open the auto complete feature. For more information, see Use formula expressions.

> **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 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.

- **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
- (UTC03+: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**.

   > **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 is on July 2, 2021, at 06:30:30 AM (PST), and so on.

Define metrics (AWS 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`.
- `externalId` – (Optional) This is a user-defined ID. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

- `expression` – The metric expression. Metric expressions can use aggregation functions to input data from a property for all associated assets in a hierarchy. For more information, see Use formula expressions.
- `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).

```
{
    "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 is 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 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.

- `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 Create an asset model (AWS CLI).

```
{
      ...
      "assetModelProperties": [
      ...
      {
        "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.

```
{
      ...
      "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"
      }
    ],
    ...
}
```

Use 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. Formula expressions combine literals, operators, functions, and variables to process data. For more information about how to define asset properties that use formula expressions, see Transform data (transforms) and Aggregate data from properties and other assets (metrics). Transforms and metrics are formula properties.

Topics

- Use variables in formula expressions
- Use literals in formula expressions
- Use operators in formula expressions
- Use constants in formula expressions
- Use functions in formula expressions
- Formula expression tutorials

Use variables in formula expressions

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), raw data streams (measurements), 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 and Defining metrics.

Use variables to reference properties

A variable's *value* defines the property that it references. AWS IoT SiteWise provides different ways to do this.

- **By property ID:** You can specify the property's unique ID (UUID) to identify it.
- **By name:** If the property is on the same asset model, you can specify its name in the property ID field.
- **By path:** A variable value can refer to a property by its *path.* For more information, see Use paths to reference custom composite model properties.

> **Note**
>
> Variables are not supported by AWS IoT SiteWise console. They are used by AWS IoT SiteWise API, including the AWS Command Line Interface AWS CLI) and AWS SDKs.

A variable that you receive in a response from AWS IoT SiteWise includes full information about the value, including both the ID and the path.

However, when you pass a variable into AWS IoT SiteWise (for example, in a "create" or "update" call), you only need to specify one of these. For example, if you specify the path, you don't need to provide the ID.

Use literals in formula expressions

AWS IoT SiteWise supports the use of literals in expressions and formulas. Literals are fixed values that represent a specific data type. In AWS IoT SiteWise, you can define number and string literals in formula expressions. Literals can be used in various contexts, including data transformations, alarm conditions, and visualization calculations.

- **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""}"`

Use operators in formula expressions

You can use the following common operators in formula expressions.

Operator	Description
`+`	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 + " is three"` evaluates to `"3 is three"`. The concatenated string can have up to 1024 characters. If the string exceeds 1024

Operator	Description
	characters, then AWS IoT SiteWise doesn't output a data point for that computation.
`-`	Subtracts the right operand from the left operand. You can only use this operator with numeric operands.
`/`	Divides the left operand by the right operand. You can only use this operator with numeric operands.
`*`	Multiplies the left and right operands. You can only use this operator with numeric operands.
`^`	Raises the left operand to the power of the right operand (exponentiation). You can only use this operator with numeric operands.
`%`	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.
`x < y`	Returns 1 if `x` is less than `y`, otherwise `0`.
`x > y`	Returns 1 if `x` is greater than `y`, otherwise `0`.

Operator	Description
`x <= y`	Returns 1 if `x` is less than or equal to `y`, otherwise `0`.
`x >= y`	Returns 1 if `x` is greater than or equal to `y`, otherwise `0`.
`x == y`	Returns 1 if `x` is equal to `y`, otherwise `0`.
`x != y`	Returns 1 if `x` is not equal to `y`, otherwise `0`.
`!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`.

Operator	Description
`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. • `x` is evaluated to `None`.

`[]`
 	
`s[index]`
	Returns the character at an index `index` of the string `s`. This is equivalent to the index syntax in Python.
	
	Example Examples
	
	• `"Hello!"[1]` returns `e`.
	• `"Hello!"[-2]` returns `o`.

Operator	Description

`[]`
 	
`s[start:end:step]`
	Returns a slice of the string `s`. This is equivalent to the slice syntax in Python. This operator has the following arguments:
	
	• `start` – (Optional) The inclusive start index of the slice. Defaults to `0`.
	• `end` – (Optional) The exclusive end index of the slice. Defaults to the length of the string.
	• `step` – (Optional) The number to increment for each step in the slice. For example, you can specify 2 to return a slice with every other character, or specify `-1` to reverse the slice. Defaults to `1`.
	
	You can omit the `step` argument to use its default value. For example, `s[1:4:1]` is equivalent to `s[1:4]`.
	
	The arguments must be integers or the none constant. If you specify `none`, AWS IoT SiteWise uses the default value for that argument.
	
	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"`.

Use constants in formula expressions

In AWS IoT SiteWise, you can use constants in your expressions and formulas to represent fixed values or predefined parameters. Constants can be used in various contexts, such as data transformations, alarm conditions, or visualization calculations. By using constants, you can simplify your expressions and make them more readable and maintainable.

You can use the following common mathematical constants in your expressions. All constants are case insensitive.

> **Note**
>
> If you define a variable with the same name as a constant, the variable overrides the constant.

Constant	Description
`pi`	The number pi (π): `3.141592653589793`
`e`	The number e: `2.718281828459045`
`true`	Equivalent to the number 1. In AWS IoT SiteWise, Booleans convert to their number equivalents.
`false`	Equivalent to the number 0. In AWS IoT SiteWise, Booleans convert to their number equivalents.
`none`	Equivalent to no value. You can use this constant to output nothing as the result of a conditional expression.

Use functions in formula expressions

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.

> **Note**
>
> You can include a maximum of 10 functions in a formula expression.

Function type	Transforms	Metrics
Use common functions in formula expressions	Yes	Yes
Use comparison functions in formula expressions	Yes	Yes
Use conditional functions in formula expressions	Yes	Yes
Use string functions in formula expressions	Yes	Yes
Use aggregation functions in formula expressions	No	Yes

Function type	Transforms	Metrics
Use temporal functions in formula expressions	Yes	Yes
Use date and time functions in formula expressions	Yes	Yes

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.

Use common functions in formula expressions

In transforms and metrics, you can use the following functions to calculate common mathematical functions in transforms and metrics.

Function	Description
`abs(x)`	Returns the absolute value of `x`.
`acos(x)`	Returns the arccosine of `x`.
`asin(x)`	Returns the arcsine of `x`.
`atan(x)`	Returns the arctangent of `x`.
`cbrt(x)`	Returns the cubic root of `x`.
`ceil(x)`	Returns the nearest integer greater than `x`.
`cos(x)`	Returns the cosine of `x`.
`cosh(x)`	Returns the hyperbolic cosine of `x`.
`cot(x)`	Returns the cotangent of `x`.
`exp(x)`	Returns `e` to the power of `x`.
`expm1(x)`	Returns `exp(x) - 1`. Use this function to more accurately calculate `exp(x) - 1` for small values of `x`.
`floor(x)`	Returns the nearest integer less than `x`.

Function	Description
`log(x)`	Returns the $\log_e$ (base e) of x.
`log10(x)`	Returns the $\log_{10}$ (base 10) of x.
`log1p(x)`	Returns `log(1 + x)`. Use this function to more accurately calculate `log(1 + x)` for small values of x.
`log2(x)`	Returns the $\log_2$ (base 2) of x.
`pow(x, y)`	Returns x to the power of y. This is equivalent to `x ^ y`.
`signum(x)`	Returns the sign of x (-1 for negative inputs, 0 for zero inputs, +1 for positive inputs).
`sin(x)`	Returns the sine of x.
`sinh(x)`	Returns the hyperbolic sine of x.
`sqrt(x)`	Returns the square root of x.
`tan(x)`	Returns the tangent of x.
`tanh(x)`	Returns the hyperbolic tangent of x.

Use comparison functions in formula expressions

In transforms and metrics, 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.

Function	Description
`gt(x, y)`	Returns 1 if x is greater than y, otherwise 0 (`x > y`).

Function	Description
	This function doesn't return a value if `x` and `y` are incompatible types, such as a number and a string.
`gte(x, y)`	Returns 1 if `x` is greater than or equal to `y`, otherwise `0` (`x ≥ y`). 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. This function doesn't return a value if `x` and `y` are incompatible types, such as a number and a string.
`eq(x, y)`	Returns 1 if `x` is equal to `y`, otherwise `0` (`x == y`). 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. This function doesn't return a value if `x` and `y` are incompatible types, such as a number and a string.
`lt(x, y)`	Returns 1 if `x` is less than `y`, otherwise `0` (`x < y`). This function doesn't return a value if `x` and `y` are incompatible types, such as a number and a string.

Function	Description
`lte(x, y)`	Returns 1 if `x` is less than or equal to `y`, otherwise `0` (`x ≤ y`). AWS IoT SiteWise considers the arguments equal if they are within a relative tolerance of `1E-9`. This behaves similar to the [isclose](https://docs.python.org/3/library/math.html#math.isclose) function in Python. This function doesn't return a value if `x` and `y` are incompatible types, such as a number and a string.
`isnan(x)`	Returns 1 if `x` is equal to `NaN`, otherwise `0`. This function doesn't return a value if `x` is a string.

Use conditional functions in formula expressions

In transforms and metrics, you can use the following function to check a condition and return different results, whether the condition evaluates to true or false.

Function	Description
`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 from this function to discard the output for a particula

Function	Description
	r condition. This means you can filter out data points that don't meet a condition. For more information, see Filter data points. **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`. 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))` . You can also chain additional intermediate conditional functions. For example, you can use `if(condition1, result1_i`

Function	Description
	`f_true).elif(condition2, result2_if_true).elif(condition3, result3_if_true, result3_if_false)` instead of nesting multiple `if` statements, such as `if(condition1, result1_if_true, if(condition2, result2_if_true, if(condition3, result3_if_true result3_if_false)))` . **Important** You must use `elif(condition, result_if_true, result_if_false)` with UFCS.

Use string functions in formula expressions

In transforms and metrics, you can use the following functions to operate on strings. For more information, see Use strings in formulas.

> **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 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.

Function	Description
`len(s)`	Returns the length of the string `s`.

Function	Description
`find(s, substring)`	Returns the index of the string `substring` in the string `s`.
`contains(s, substring)`	Returns `1` if the string `s` contains the string `substring`, otherwise `0`.
`upper(s)`	Returns the string `s` in uppercase form.
`lower(s)`	Returns the string `s` in lowercase form.

Function	Description
`jp(s, json_path)`	Evaluates the string `s` with the [JsonPath](https://github.com/json-path/JsonPath) expression `json_path` and returns the result. 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', '$')` returns 111 as a number. To extract a string value from a JSON structure and return it as a number, you must use multiple nested `jp` functions. The outer `jp` function extracts the string from the JSON structure, and the inner `jp` function converts the string to a number. The string `json_path` must contain a string literal. This means that `json_path` can't be an expression that evaluates to a string. **Example Examples** • `jp('{"status":"active","value":15}', '$.value')` returns 15. • `jp('{"measurement":{"reading":25,"confidence":0.95}}', '$.measurement.reading')` returns 25. • `jp('[2,8,23]', '$[2]')` returns 23. • `jp('{"values":[3,6,7]}', '$.values[1]')` returns 6. • `jp('111', '$')` returns 111.

Function	Description
	• `jp(jp('{"measurement":{"reading":25,"confidence":"0.95"}}', '$.measurement.confidence'), '$')` returns `0.95`.
`join(s0, s1, s2, s3, ...)`	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(expression: "format")` or `format("format", expression)`	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 a string, `101`. • `format("The result is %d", 100+1)` returns a string, `The result is 101`.

Function	Description
`f'expression'`	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 `{}` (curly braces) to interpolate expressions. This behaves similar to the formatted string literals in Python. **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')` .

Use aggregation functions in formula expressions

In metrics 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, number literals, temporal functions, nested expressions, or aggregation functions. The formula `max(latest(x), latest(y), latest(z))` uses an aggregation function as an argument 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.

> **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.

Function	Description
avg(x_0, ..., x_n)	Returns the mean of the given variables' values over the current time interval. This function outputs a data point only if the given variables have at least one data point over the current time interval.
sum(x_0, ..., x_n)	Returns the sum of the given variables' values over the current time interval.

Function	Description
	This function outputs a data point only if the given variables have at least one data point over the current time interval.
`min(x0, ..., xn)`	Returns the minimum of the given variables' values over the current time interval. This function outputs a data point only if the given variables have at least one data point over the current time interval.
`max(x0, ..., xn)`	Returns the maximum of the given variables' values over the current time interval. This function outputs a data point only if the given variables have at least one data point over the current time interval.
`count(x0, ..., xn)`	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 Count data points that match a condition. This function computes a data point for every time interval.
`stdev(x0, ..., xn)`	Returns the standard deviation of the given variables' values over the current time interval. This function outputs a data point only if the given variables have at least one data point over the current time interval.

Use temporal functions in formula expressions

Use temporal functions to return values based on timestamps of data points.

Use temporal functions in metrics

In metrics 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 or nested expressions. This means that you can't use properties from child asset models in 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.

Function	Description
`first(x)`	Returns the given variable's value with the earliest timestamp over the current time interval.
`last(x)`	Returns the given variable's value with the latest timestamp over the current time interval.
`earliest(x)`	Returns the given variable's last value before the start of the current time interval. This function computes a data point for every time interval, if the input property has at least one data point in its history. See time-range-defintion for details.

Function	Description
`latest(x)`	Returns the given variable's last value with the latest timestamp before the end of the current time interval. This function computes a data point for every time interval, if the input property has at least one data point in its history. See time-range-defintion for details.
`statetime(x)`	Returns the amount of time in seconds that the given variables are positive over the current time interval. You can use the comparison functions to create a transform property for the `statetime` function to consume. For example, if you have an `Idle` property that is `0` or `1`, you can calculate idle time per time interval with this expression: `IdleTime = statetime(Idle)` . For more informati on, see the example statetime scenario. This function doesn't support metric propertie s as input variables. This function computes a data point for every time interval, if the input property has at least one data point in its history.

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td><code>TimeWeightedAvg(x, [interpolation])</code></td><td>Returns the average of input data weighted with time intervals between points.
See Time weighted functions parameters for computation and intervals details.

The optional argument <code>interpolaton</code> must be a string constant:

• <code>locf</code> – This is the default. The calculation uses the Last Observed Carry Forward computation algorithm for intervals between data points. In this approach, the data point is computed as the last observed value until the next input data point time stamp.

The value after a good data point is extrapolated as its value until the next data point timestamp.
• <code>linear</code> – The calculation uses the linear interpolation computation algorithm for intervals between data points.

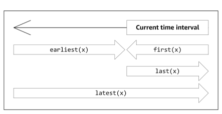
The value between two good data points is extrapolated as linear interpolation between those data point's values.

The value between good and bad data points or the value after the last good data point will be extrapolated as a good data point.</td></tr>
</table>

Function	Description
`TimeWeightedStDev(x, [algo])`	Returns the standard deviation of input data weighted with time intervals between points. See Time weighted functions parameters for computation and intervals details. The calculation uses the Last Observed Carry Forward computation algorithm for intervals between data points. In this approach, the data point is computed as the last observed value until the next input data point time stamp. Weight is computed as time interval in seconds between data points or window boundaries. The optional argument `algo` must be a string constant: • `f` – This is the default. It returns an unbiased weighted sample variance with **Frequency** weights, where TimeWeight is computed in seconds. This algorithm is usually assumed under standard deviation and is known as Bessel's correction of standard deviation for weighted samples. • p – Returns the biased weighted sample variance, also known as **Population** variance. The following formulas are used for computati on where: • S_p = population standard deviation • S_f = frequency standard deviation • X_i = incoming data

Function	Description
	• ω_i = weight that equals time interval in seconds • μ^* = a weighted mean of incoming data Equation for population standard deviation: $$S_p^2 = \frac{\sum_{i=1}^{N} \omega_i (x_i - \mu^*)^2}{\sum_{i=1}^{N} \omega_i}$$ Equation for frequency standard deviation: $$S_f^2 = \frac{\sum_{i=1}^{N} \omega_i (x_i - \mu^*)^2}{\sum_{i=1}^{N} \omega_i - 1}$$

The following diagram shows how AWS IoT SiteWise computes the temporal functions `first`, `last`, `earliest`, and `latest`, relative to the current time interval.



> **Note**
>
> - The time range for `first(x)`, `last(x)` is (current window start, current window end].
> - The time range for `latest(x)` is (beginning of time, current window end].
> - The time range for `earliest(x)` is (beginning of time, previous window end].

Time-weighted functions parameters

Time-weighted functions computed for the aggregate window take into account the following:

- Data points inside the window

- Time intervals between data points
- Last data point before the window
- First data point after the window (for some algorithms)

Terms:

- **Bad data point** – Any data point with non-good quality or non-number value. This is not considered in a window result computation.
- **Bad interval** – The interval after a bad data point. The interval before the first known data point is also considered a bad interval.
- **Good data point** – Any data point with good quality and numeric value.

> **Note**
>
> - AWS IoT SiteWise only consumes `GOOD` quality data when it computes transforms and metrics. It ignores `UNCERTAIN` and `BAD` data points.
> - The interval before the first known data point is considered a **bad interval**. See the section called "Formula expression tutorials" for more information.

The interval after the last known data point continues indefinitely, affecting all following windows. When a new data point arrives, the function recomputes the interval.

Following the rules above, the aggregate window result is computed and limited to window boundaries. By default, the function only sends the window result if the whole window is a **good interval**.

If the window **good interval** is smaller than the window length, the function does not send the window.

When the data points affecting the window result change, the function recalculates the window, even if the data points are outside of the window.

If the input property has at least one data point in its history and a computation has been initiated, the function calculates the time-weighted aggregate functions for every 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 `statetime(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.

Timestamp	2:00:00 PM	2:00:30 PM	2:01:15 PM	2:02:45 PM	2:04:00 PM
Idle	0	1	1	0	0

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.

Timestamp	2:00:00 PM	2:01:00 PM	2:02:00 PM	2:03:00 PM	2:04:00 PM
Idle Time	N/A	30	60	45	0

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.

Example Example TimeWeightedAvg and TimeWeightedStDev scenario

The following tables provide sample inputs and outputs for these one-minute window metrics: `Avg(x), TimeWeightedAvg(x), TimeWeightedAvg(x, "linear"), stDev(x), timeWeightedStDev(x), timeWeightedStDev(x, 'p')`.

Sample input for one-minute aggregate window:

> **Note**
>
> These data points all have `GOOD` quality.

03:00:00	4.0
03:01:00	2.0
03:01:10	8.0
03:01:50	20.0
03:02:00	14.0
03:02:05	10.0

03:02:10	3.0
03:02:30	20.0
03:03:30	0.0

Aggregate results output:

> **Note**
>
> None – Result not produced for this window.

Time	`Avg(x)`	`TimeWeightedAvg(x)`	`TimeWeightedAvg(X, "linear")`	`stDev(X)`	`timeWeightedStDev(x)`	`timeWeightedStDev(x, 'p')`
3:00:00	4	None	None	0	None	None
3:01:00	2	4	3	0	0	0
3:02:00	14	9	13	6	5.430610041581775	5.385164807134504
3:03:00	11	13	12.875	8.5440037 4531753	7.7240544 37220943	7.6594168 62050705
3:04:00	0	10	2.5	0	10.084389681792215	10
3:05:00	None	0	0	None	0	0

Use temporal functions in transforms

In transforms only, you can use the `pretrigger()` function to retrieve the `GOOD` quality value for a variable prior to the property update that initiated 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 `if(pretrigger(current_state) == 1, timestamp(current_state) - timestamp(pretrigger(current_state)), 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 Use conditional functions in formula expressions and the timestamp() 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 + 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 9:05 AM arrive before the data points for 9:00 AM, `z` isn't calculated at 9:05 AM.
- If the value of `x` for 9:05 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 9:05 AM.

	09:00 AM	09:05 AM	09:10 AM	09:15 AM
x	10	20		30
y	1	2	3	

	09:00 AM	09:05 AM	09:10 AM	09:15 AM
`z = x + y + pretrigger(y)`	y doesn't receive any data point before 09:00 AM. Therefore, z isn't calculated at 09:00 AM.	23 = 20 + 2 + 1 `pretrigger(y)` equals 1.	25 = 20 + 3 + 2 x doesn't receive a new data point. `pretrigger(y)` equals 2.	36 = 30 + 3 + 3 y doesn't receive a new data point. Therefore, `pretrigger(y)` equals 3 at 09:15 AM.

Use date and time functions in formula expressions

In transforms and metrics, 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.

Function	Description
`now()`	Returns the current date and time, in seconds, in the Unix epoch format.
`timestamp()`	• In transforms, the function returns the timestamp, in seconds, of the input message in the Unix epoch format. In transforms only, you can do one of the following: • Provide a variable as an argument to the function. The `timestamp(` *`variable-name`* `)` function returns the timestamp, in seconds, of the latest `GOOD` quality value for the specified variable in the Unix epoch format.

Function	Description
	For example, if your asset has a transform property named `Temperature_F` that uses the `9/5 * Temperature_C` formula to convert each temperature data point from Celsius to Fahrenheit, you can use the `timestamp(Temperature_F)` function to get the timestamp of the latest `GOOD` quality value for the `Temperature_F` property. • Use the `pretrigger()` function as an argument to the function. The `timestamp(pretrigger(` *`variable-name`* `))` function returns the timestamp , in seconds, of the `GOOD` quality value for the specified variable prior to the property update that initiated the current transform calculation in the Unix epoch format. For more information, see [Use temporal functions in transforms](#). • In metrics, the function returns the timestamp retrieved at the end of the current window, in seconds, in the Unix epoch format.

Function	Description
`mktime(time_zone, year, month, day_of_month, hour, minute, second)`	Returns the input time in seconds, in the Unix epoch format. The following requirements apply for using this function: • The time zone argument must be a quoted string (`'UTC'`). If not specified, the default time zone is UTC. The time zone argument can be the first or last argument. • The year, month, day of month, hour, minute, and second arguments must be in order. • The year, month, and date arguments are required. 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:

Function	Description
	• `mktime(2020, 2, 29)` • `mktime('UTC+3', 2021, 12, 31, 22)` • `mktime(2022, 10, 13, 2, 55, 13.68, 'PST')`

Function	Description
`localtime(unix_time, time_zone)`	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. The following requirements apply for using this function: • The time zone argument must be a quoted string (`'UTC'`). If not specified, the default time zone is UTC. • 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. Example response: `2007-12-03T10:15:30+01:00[Europe/Paris]` `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. Examples: • `year(localtime('GMT', 1605898608.8113723))` • `now().localtime().year()` • `timestamp().localtime(‘PST’ ).year()`

Function	Description
	• `localtime(1605289736, ‘Europe/London’).year()`
`year(localtime(unix_time, time_zone)`	Returns the year from `localtime (unix_time, time_zone)` .
`mon(localtime(unix_time, time_zone))`	Returns the month from `localtime (unix_time, time_zone)` .
`mday(localtime(unix_time, time_zone))`	Returns the day of the month from `localtime(unix_time, time_zone)` .
`wday(localtime(unix_time, time_zone))`	Returns the day of the week from `localtime (unix_time, time_zone)` .
`yday(localtime(unix_time, time_zone))`	Returns the day of the year from `localtime (unix_time, time_zone)` .
`hour(localtime(unix_time, time_zone))`	Returns the hour from `localtime (unix_time, time_zone)` .
`minute(localtime(unix_time, time_zone))`	Returns the minute from `localtime (unix_time, time_zone)` .
`sec(localtime(unix_time, time_zone))`	Returns the second from `localtime (unix_time, time_zone)` .

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.
- Region based IDs - For example, `'Etc/GMT+12'` and `'Pacific/Pago_Pago'`.

Supported time zone abbreviations

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
- CET - Europe/Paris
- CNT - America/St_Johns
- CST - America/Chicago
- CTT - Asia/Shanghai
- EAT - Africa/Addis_Ababa
- 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

Supported Region-based IDs

The date and time functions support the following Region-based IDs, organized by their relation to UTC+00:00:

- 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/Guadalcanal (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/DumontDUrville (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)
- 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)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

Use 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 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.

AWS IoT SiteWise provides the following formula expression features that you can use to operate on strings:

- String literals
- The index operator (`s[index]`)
- The slice operator (`s[start:end:step]`)
- Comparison functions, which you can use compare strings by lexicographic order
- String functions, which include the `jp` function that can parse serialized JSON objects and convert strings to numbers

Filter data points

You can use the if function 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 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.

Count data points that match a condition

You can use comparison functions and sum() 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 [Query data from AWS IoT SiteWise](#).

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.

Example Examples

Each of the following formula expressions result in a value that AWS IoT SiteWise can't represent as a number. AWS IoT SiteWise doesn't output a data point when it computes these formula expressions.

- `x / 0` is undefined.
- `log(0)` is undefined.
- `sqrt(-1)` is undefined in a real number system.
- `"hello" + " world"` is a string.
- `jp('{"values":[3,6,7]}', '$.values')` is an array.
- `if(gte(temp, 300), temp, none)` is none when `temp` is less than `300`.

Create custom composite models (components)

Custom composite models, or components if you're using the console, provide another level of organization for your asset models and component models. You can use them to structure your models by grouping properties or referencing other models. For more information about working with custom composite models, see Custom composite models (components).

You create a custom composite model within an existing asset model or component model. There are two types of custom composite models. To group related properties within a model, you can create an **inline** custom composite model. To reference a component model within your asset model or component model, you can create a **component-model-based** custom composite model.

The following sections describe how to use the AWS IoT SiteWise API to create custom composite models.

Topics

Create an inline component (console)

You can use the AWS IoT SiteWise console to create an inline component that defines its own properties.

> **Note**
>
> Because this is an *inline* component, these properties only apply to the current asset model and aren't shared anywhere else.
> If you need to produce a reusable model (for example, to share among multiple asset models, or to include multiple instances within one asset model), you should create a component based on a component model instead. See the following section for details.

To create a component (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model to which you want to add a component.
4. On the **Properties** tab, choose **Components**.
5. Choose **Create component**.
6. On the **Create component** page, do the following:

   a. Enter a **Name** for the component, such as **`ServoMotor`** or **`ServoMotor Model`**. This name must be unique across all components in your account in this Region.

   b. (Optional) Add **Attribute definitions** for the model. Attributes represent information that rarely changes. For more information, see Define static data (attributes).

c. (Optional) Add **Measurement definitions** for the model. Measurements represent data streams from your equipment. For more information, see Define data streams from equipment (measurements).

d. (Optional) Add **Transform definitions** for the model. Transforms are formulas that map data from one form to another. For more information, see Transform data (transforms).

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 Aggregate data from properties and other assets (metrics).

f. Choose **Create component**.

Create an inline custom composite model (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to create an inline custom composite model that defines its own properties.

Use the CreateAssetModelCompositeModel operation to create an inline model with properties. This operation expects a payload with the following structure.

> **Note**
>
> Because this is an *inline* composite model, these properties only apply to the current asset model and aren't shared anywhere else. What makes it "inline" is that it doesn't provide a value for the `composedAssetModelId` field.
> If you need to produce a reusable model (for example, to share among multiple asset models, or to include multiple instances within one asset model), you should create a *component-model-based* composite model instead. See the following section for details.

```
{
    "assetModelCompositeModelName": "CNCLathe_ServoMotorA",
    "assetModelCompositeModelType": "CUSTOM",
    "assetModelCompositeModelProperties": [
        {
            "dataType": "DOUBLE",
            "name": "Servo Motor Temperature",
            "type": {
```

```
                "measurement": {}
                },
                "unit": "Celsius"
            },
            {
                "dataType": "DOUBLE",
                "name": "Spindle speed",
                "type": {
                "measurement": {}
                },
                "unit": "rpm"
            }
        ]
}
```

Create a component-model-based component (console)

You can use the AWS IoT SiteWise console to create a component based on a component model.

To create a component-model-based component (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model to which you want to add a component.
4. On the **Properties** tab, choose **Components**.
5. Choose **Create component**.
6. On the **Create component** page, do the following:
    a. Select the component model you want to based the component on.
    b. Enter a **Name** for the component, such as **`ServoMotor`** or **`ServoMotor Model`**. This name must be unique across all components in your account in this Region.
    c. Choose **Create component**.

Create a component-model-based custom composite model (AWS CLI)

You can use the AWS CLI to create a component-model-based custom composite model within your asset model. A component-model-based custom composite model is a reference to a component model that you've already defined elsewhere.

Use the CreateAssetModelCompositeModel operation to create a component-model-based custom composite model. This operation expects a payload with the following structure.

> **Note**
>
> In this example, the value of `composedAssetModelId` is the asset model ID or external ID of an existing component model. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*. For an example of how to create a component model, see Create a component model (AWS CLI).

```
{
    "assetModelCompositeModelName": "CNCLathe_ServoMotorA",
    "assetModelCompositeModelType": "CUSTOM",
    "composedAssetModelId": component model ID
]
```

Since it's just a reference, a component-model-based custom composite model has no properties of its own, other than a name.

If you want to add multiple instances of the same component to your asset model (for example, a CNC machine that has multiple servo motors), you can add multiple component-model-based custom composite models that each have their own name but which all reference the same `composedAssetModelId`.

You can nest components within other components. To do so, you can add a component-model-based composite model, as shown in this example, to one of your component models.

Create assets for asset models in AWS IoT SiteWise

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 Create asset models in AWS IoT SiteWise.

> **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.

Topics

Create 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 Tag your AWS IoT SiteWise resources.

   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.

After you create an asset, see Configure a new asset.

Create an asset (AWS 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`, use the ListAssetModels API to view all of your asset models.

To create an asset from an asset model, use the CreateAsset API with the following parameters:

- `assetName` – The new asset's name. Give your asset a name to help you identify it.
- `assetModelId` – The ID of the asset. This is the actual ID in UUID format, or the `externalId:myExternalId` if it has one. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

To create an asset (AWS 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 or the external ID of the asset model.

```
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.

```
{
  "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.

After you create an asset, see Configure a new asset.

Configure a new asset

After creating an asset in AWS IoT SiteWise, there are several next steps you can take to fully utilize the asset and its data. These steps might include configuring data streams to ingest data from the asset, setting up alarms and notifications to monitor the asset's performance, creating visualizations and dashboards to display the asset's data, and integrating the asset with other AWS services or third-party applications for further analysis or automation.

Finish configuring your asset with the following optional actions:

- Manage data streams for AWS IoT SiteWise if your asset has measurement properties.
- Update attribute values if your asset has unique attribute values.
- Associate and disassociate assets if your asset is a parent asset.

Search assets on AWS IoT SiteWise console

Use the AWS IoT SiteWise console search functionality to find assets based on metadata and real-time property value filters.

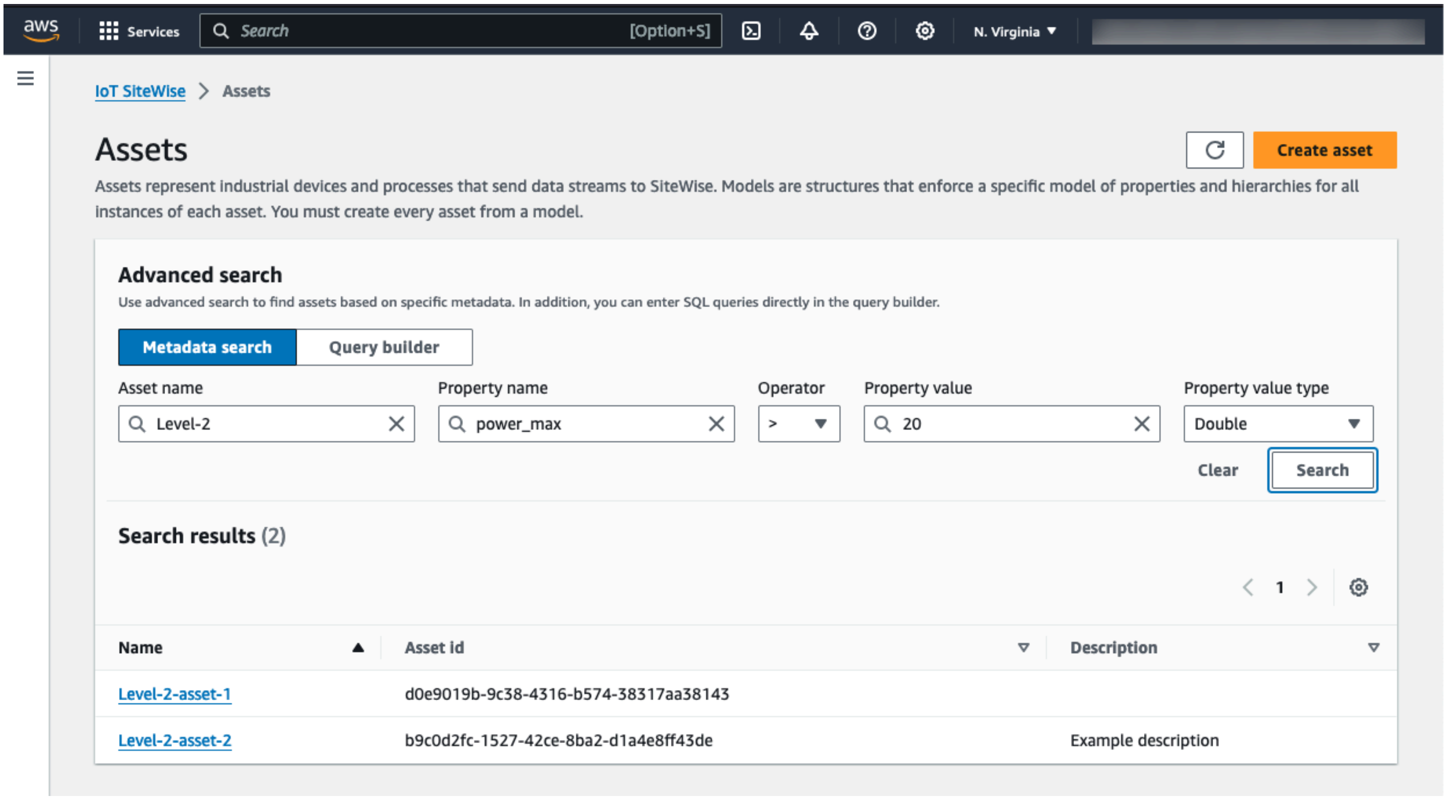
Prerequisites

AWS IoT SiteWise requires permissions to integrate with AWS IoT TwinMaker to better organize, and model industrial data. If you have granted permissions to AWS IoT SiteWise, use the ExecuteQuery API. If you have not granted permissions to AWS IoT SiteWise, and need assistance getting started, see Integrate AWS IoT SiteWise and AWS IoT TwinMaker.

Advanced search on AWS IoT SiteWise console

Metadata search

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Advanced search** under **Assets**.
3. Under **Advanced search** choose the **Metadata search** option.
4. Fill in the parameters. Fill in as many fields as possible for an efficient search.
   a. **Asset name** — Enter a full asset name, or a partial name for a wide search.
   b. **Property name** — Enter a full property name, or a partial name for a wide search.
   c. **Operator** — Choose an operator from:
      - **=**
      - **<**
      - **>**
      - **<=**
      - **>=**
   d. **Property value** — This value is compared with the property's latest value.
   e. **Property value type** — The data type of the property. Choose from the following:
      - **Double**
      - **Integer**
      - **String**
      - **Boolean**
5. Choose **Search**.
6. From the **Search results** table, choose the asset from the **Name** column. This takes you to the detailed asset page for that asset.



Partial search

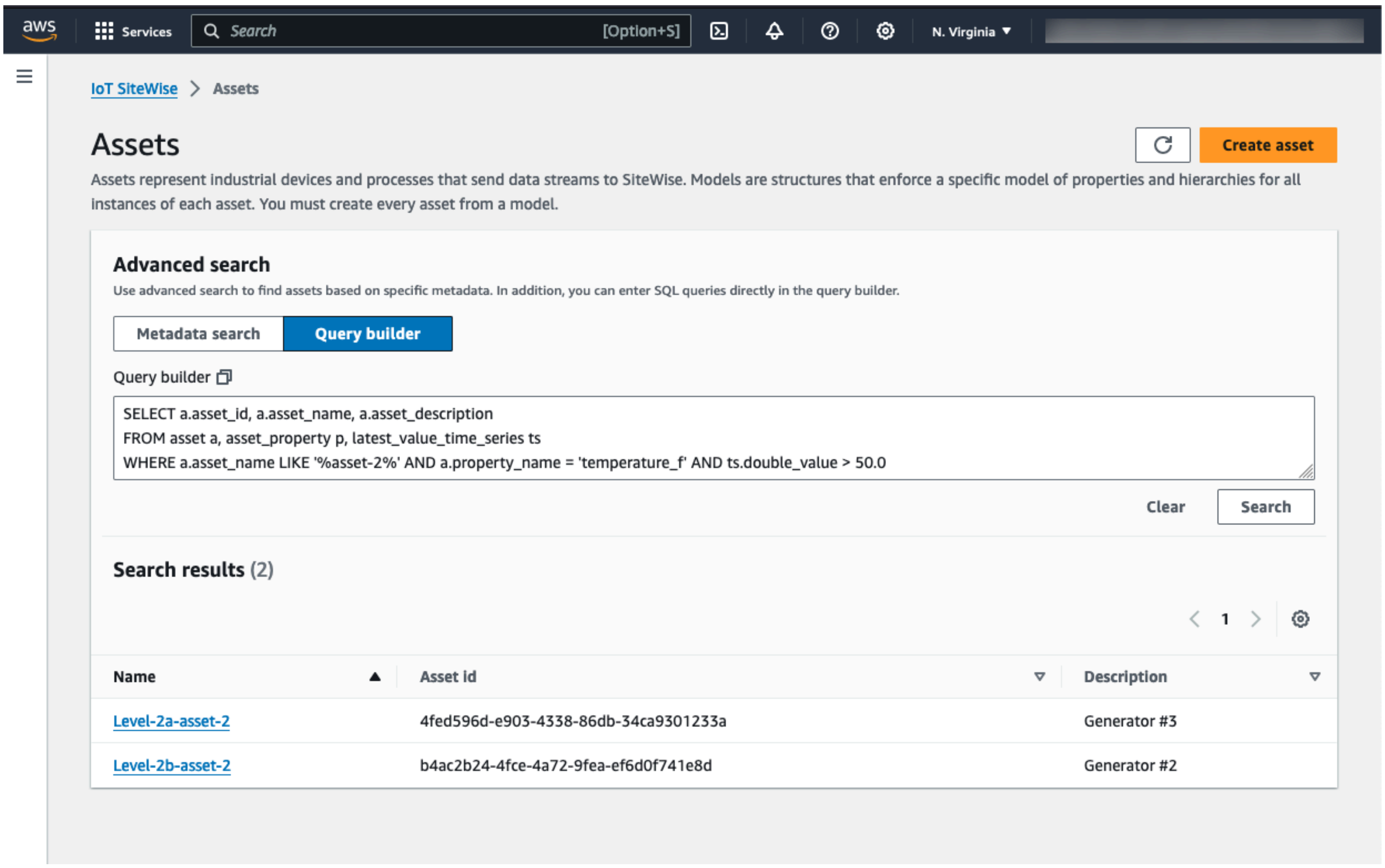
All parameters do not need to be provided for an asset search. Here are some examples of partial searches using the Metadata search option:

- Find assets by their name:
  - Enter a value in the **Asset name** field alone.
  - The **Property name** and **Property value** fields are empty.
- Find assets containing properties with a specific name:
  - Enter a value in the **Property name** field alone.
  - The **Asset name** and **Property value** fields are empty.
- Find assets based on the latest values of their properties:
  - Enter values in the **Property name** and **Property value** fields.
  - Select an **Operator** and **Property value type**.

Query builder search

1. Navigate to the AWS IoT SiteWise console.

2. In the navigation pane, choose **Advanced search** under **Assets**.
3. Under **Advanced search** choose the **Query builder** option.
4. In the **Query builder** pane, write your SQL query to retrieve an `asset_name`, `asset_id` and `asset_description`.
5. Choose **Search**.
6. From the **Search results** table, choose the asset from the **Name** column. This takes you to the detailed asset page for that asset.



> **Note**
>
> - The SELECT clause in the SQL query must include the `asset_name` and `asset_id` fields to ensure a valid asset in the **Search results** table.
> - The **Query builder** only displays the **Name**, **Asset id**, and **Description** in the results table. Adding more fields to the SELECT clause does not add more columns to the results table

Update 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)

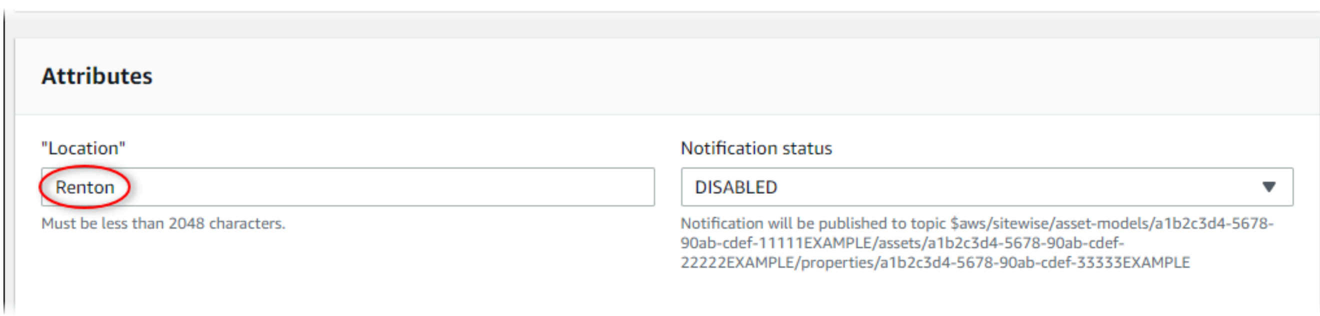
You can use the AWS IoT SiteWise console to update the value of an attribute asset property.

To update an attribute's value (console)

1. Navigate to the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/).
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 (AWS 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. You can also use the external ID. If you created an asset and don't know its `assetId`, use the

ListAssets API to list all the assets for a specific model. 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 (AWS 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.

```
{
  "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`
    - `doubleValue`
    - `integerValue`
    - `stringValue`
    - `nullValue`
  - `timestamp` – A structure that contains the current Unix epoch time in seconds, `timeInSeconds`. AWS IoT SiteWise rejects any data points with timestamps that existed longer 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 Ingest data with AWS IoT SiteWise APIs.

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
```

Associate and disassociate 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 Define asset model hierarchies. If you're using interfaces, hierarchies defined in the interface are enforced on the asset models that implement the interface. For more information about interfaces, see Asset model interfaces.

Topics

- Associate and disassociate assets (console)
- Associate and disassociate assets (AWS CLI)

Associate and disassociate assets (console)

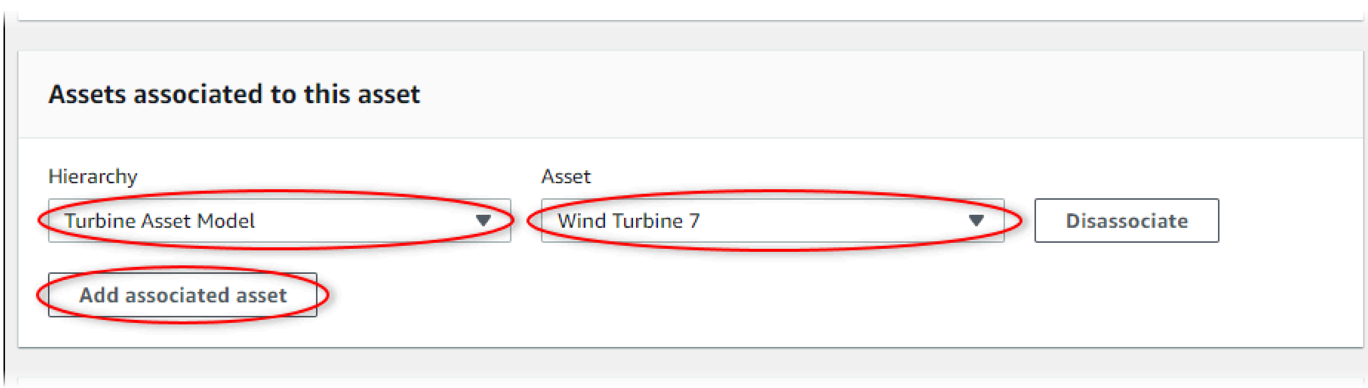
You can use the AWS IoT SiteWise console to associate and disassociate assets.

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**.



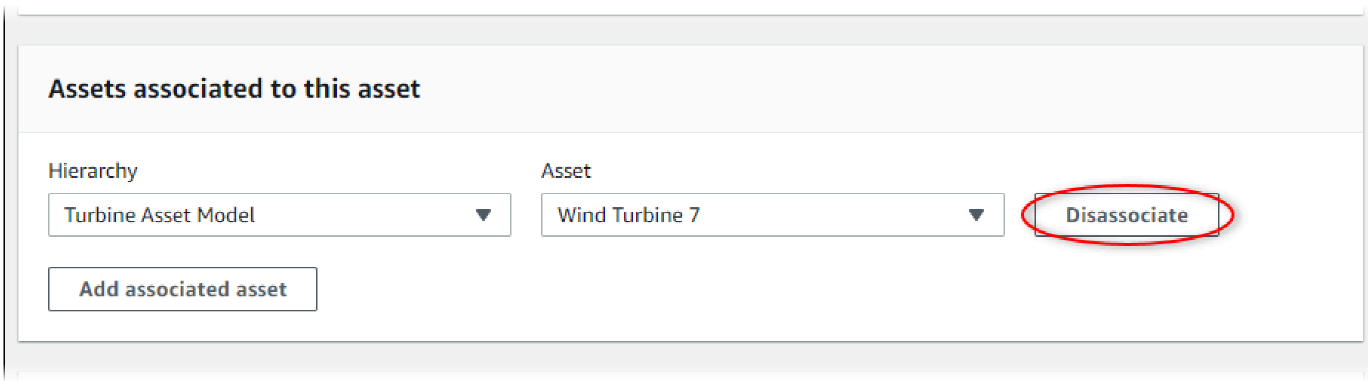
6. For **Hierarchy**, choose the hierarchy that defines the relationship between the parent asset and the child asset.
7. For **Asset**, choose the child asset to associate.
8. Choose **Save**.

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**.

Associate and disassociate assets (AWS 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 or external ID.

```
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:

```
{
  ...
  "assetHierarchies": [
    {
      "id": "String",
      "name": "String"
    }
  ],
  ...
}
```

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 or external ID.
- `hierarchyId` – The hierarchy ID or external ID in the parent asset.
- `childAssetId` – The child asset's ID or external ID.

To associate an asset (AWS 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 (AWS 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
```

Update assets and models

You can update your assets, asset models, component models, and interfaces 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 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

Update assets in AWS IoT SiteWise

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.

Topics

Update 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:
   - Manage data streams for AWS IoT SiteWise
   - Update attribute values
   - Interact with other AWS services
7. Choose **Save**.

Update an asset (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to update an asset's name.

Use the UpdateAsset operation to update an asset. Specify the following parameters:

- `assetId` – The ID of the asset. This is the actual ID in UUID format, or the `externalId:myExternalId` if it has one. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.
- `assetName` – The asset's new name.

To update an asset's name (AWS CLI)

- Run the following command to update an asset's name. Replace *asset-id* with the ID or external ID of the asset. Update the *asset-name* with the new name for the asset.

```
aws iotsitewise update-asset \
  --asset-id asset-id \
  --asset-name asset-name
```

Update asset models, component models, and interfaces

Use the AWS IoT SiteWise console or API to update an asset model, component model, or interface.

You can't change the type or data type of an existing property, or the window of an existing metric. You also can't change the type of the model from asset model to component model or interface, or the other way around.

Important

- If you remove a property from an asset model or component model, AWS IoT SiteWise deletes all previous data for that property. For component models, this affects **all asset models using that component model**, so be especially careful to understand how widely your change may apply.
- 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`.

When you update a component model, every asset model that incorporates that component model reflects the changes. Until the component model changes propagate, each affected asset model has the `UPDATING` state, followed by `PROPAGATING` as it updates its associated assets, as described in the preceding paragraph. You must wait until those asset models return to the `ACTIVE` state before you interact with them. During this time, the updated component model's status will be `PROPAGATING`.

For more information, see Asset and model states.

Topics

Updating an asset model, component model, or interface (console)

You can use the AWS IoT SiteWise console to update an asset model, component model, or interface.

To update an asset model, component model, or interface (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model, component model, or interface 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 Define static data (attributes).
   - Change any of the **Measurement definitions**. You can't change the **Data type** of existing measurements. For more information, see Define data streams from equipment (measurements).
   - Change any of the **Transform definitions**. For more information, see Transform data (transforms).
   - Change any of the **Metric definitions**. You can't change the **Time interval** of existing metrics. For more information, see Aggregate data from properties and other assets (metrics).
   - (Asset models only) Change any of the **Hierarchy definitions**. You can't change the **Hierarchy model** of existing hierarchies. For more information, see Define asset model hierarchies.

6. Choose **Save**.

> **Note**
>
> Update requests made in the console are rejected, if another user successfully updates the asset model since you last opened the **Edit model** page. The console prompts the user to **Refresh** the **Edit model** page, to fetch the updated model. You must make your updates again, and retry your save. See Optimistic locking for asset model writes for more details.

Update an asset model, component model, or interface (AWS CLI)

Use the AWS Command Line Interface (AWS CLI) to update an asset model, component model, or interface.

Use the UpdateAssetModel API to update the name, description, and properties of an asset model, component model, or interface. For asset models only, you can update hierarchies. For interfaces, you can update properties and hierarchies. Specify the following parameters:

- `assetModelId` – The ID of the asset. This is the actual ID in UUID format, or the `externalId:myExternalId` if it has one. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

Specify the updated model in the payload. To learn about the expected format of an asset model or component model, see Create asset models in AWS IoT SiteWise.

> **Warning**
>
> The UpdateAssetModel API overwrites the existing model with the model that you provide in the payload. To avoid deleting your model's properties or hierarchies, you must include their IDs and definitions in the updated model payload. To learn how to query your model's existing structure, see the DescribeAssetModel operation.

Note

The following procedure can only update composite models of type `AWS/ALARM`. If you want to update `CUSTOM` composite models, use UpdateAssetModelCompositeModel instead. For more information, see Update custom composite models (components).

To update an asset model or component model (AWS CLI)

1. Run the following command to retrieve the existing model definition. Replace *asset-model-id* with the ID or the external ID of the asset model or component model to update.

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id
```

   The above command returns the model definition corresponding to model's latest version.

   For an use case where an asset model is in a `FAILED` state, retrieve the valid model definition corresponding to its active version to build your update request. See Asset model versions for details. Run the following command to retrieve the active model definition:

```
aws iotsitewise describe-asset-model --asset-model-id asset-model-id --asset-model-version ACTIVE
```

   The operation returns a response that contains the 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,
    "assetModelCompositeModelSummaries": Array of AssetModelCompositeModelSummary,
    "assetModelCreationDate": "String",
    "assetModelLastUpdateDate": "String",
    "assetModelStatus": {
      "state": "String",
      "error": {
```

```
            "code": "String",
            "message": "String"
          },
        "assetModelType": "String"
        },
        "assetModelVersion": "String",
        "eTag": "String"
}
```

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`
   - `assetModelCompositeModelSummaries`
   - `assetModelCreationDate`
   - `assetModelLastUpdateDate`
   - `assetModelStatus`
   - `assetModelType`
   - `assetModelVersion`
   - `eTag`

   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 Define data properties.
- 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 Define asset model hierarchies.
- Change, add, or remove any of the asset model's composite models of type `AWS/ALARM` (`assetModelCompositeModels`). 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 Monitor data with alarms in AWS IoT SiteWise and Define alarms on asset models in AWS IoT SiteWise.

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
```

> **Important**
>
> When multiple users update an asset model at the same time, an user's changes may be inadvertently overwritten by another user. To prevent this, you must define a conditional update request. See Optimistic locking for asset model writes.

Update custom composite models (components)

You can use the AWS IoT SiteWise API to update a custom composite model, or the AWS IoT SiteWise console to update components.

Topics

Update a component (console)

You can use the AWS IoT SiteWise console to update a component.

To update a component (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model where the component is.
4. On the **Properties** tab, choose **Components**.
5. Choose the component that you want to update.
6. Choose **Edit**.
7. On the **Edit component** 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 Define static data (attributes).
   - Change any of the **Measurement definitions**. You can't change the **Data type** of existing measurements. For more information, see Define data streams from equipment (measurements).
   - Change any of the **Transform definitions**. For more information, see Transform data (transforms).
   - Change any of the **Metric definitions**. You can't change the **Time interval** of existing metrics. For more information, see Aggregate data from properties and other assets (metrics).
8. Choose **Save**.

Update a custom composite model (AWS CLI)

Use the AWS Command Line Interface (AWS CLI) to update a custom composite model.

To update the name or description, use the UpdateAssetModelCompositeModel operation. For inline custom composite models only, you can also update the properties. You can't update the properties of a component-model-based custom composite model, because its referenced component model provides its associated properties.

> **Important**
>
> If you remove a property from a custom composite model, AWS IoT SiteWise deletes all previous data for that property. You can't change the type or data type of an existing property.
>
> To replace an existing composite model property with a new one with the same `name`, do the following:
>
> 1. Submit an `UpdateAssetModelCompositeModel` request with the entire existing property removed.
> 2. Submit a second `UpdateAssetModelCompositeModel` request that includes the new property. The new asset property will have the same `name` as the previous one and AWS IoT SiteWise will generate a new unique `id`.

To update a custom composite model (AWS CLI)

1. To retrieve the existing composite model definition, run the following command. Replace *`composite-model-id`* with the ID or the external ID of the custom composite model to update, and *`asset-model-id`* with the asset model that the custom composite model is associated with. For more information, see the *AWS IoT SiteWise User Guide*.

   a. Run the command below:

   ```
   aws iotsitewise describe-asset-model-composite-model \
   --asset-model-composite-model-id composite-model-id \
   --asset-model-id asset-model-id
   ```

   b. The above command returns the composite model definition corresponding to associated model's latest version. For an use case where an asset model is in a `FAILED` state, retrieve the valid model definition corresponding to its active version to build your update request. See Asset model versions for details.

   c. Run the following command to retrieve the active model definition:

   ```
   aws iotsitewise describe-asset-model-composite-model \
   --asset-model-composite-model-id composite-model-id \
   --asset-model-id asset-model-id \
   --asset-model-version ACTIVE
   ```

d. For more information, see the DescribeAssetModelCompositeModel operation.

2. Create a file called `update-custom-composite-model.json`, and then copy the previous command's response into the file.

3. Remove every key-value pair from the JSON object in `update-custom-composite-model.json` except for the following fields:

   - `assetModelCompositeModelName`
   - `assetModelCompositeModelDescription` (if present)
   - `assetModelCompositeModelProperties` (if present)

4. In `update-custom-composite-model.json`, do any of the following:

   - Change the value of `assetModelCompositeModelName`.
   - Add or remove `assetModelCompositeModelDescription`, or change its value.
   - For inline custom composite models only: Change, add, or remove any of the asset model's properties in `assetModelCompositeModelProperties`.

   For more information about the required format for this file, see the request syntax for UpdateAssetModelCompositeModel.

5. Run the following command to update the custom composite model with the definition stored in `update-custom-composite-model.json`. Replace *`composite-model-id`* with the ID of the composite model, and *`asset-model-id`* with the ID of the asset model it's in.

```
aws iotsitewise update-asset-model-composite-model \
--asset-model-composite-model-id composite-model-id \
--asset-model-id asset-model-id \
--cli-input-json file://update-custom-composite-model.json
```

Important

When multiple users update an asset model at the same time, an user's changes may be inadvertently overwritten by another user. To prevent this, you must define a conditional update request. See Optimistic locking for asset model writes.

Optimistic locking for asset model writes

When updating an asset model, an user does the following:

1. Read the current asset model definition.
2. Edit the asset model definition with required changes.
3. Update asset model with the new definition.

In a scenario with two users updating a model, the following is possible:

- User A reads the asset model X definition.
- User B reads the asset model X definition and commits changes, modifying the definition of X.
- User A commits and overwrites the change made by user B for asset model X, without verifying or incorporating User B's changes.

Optimistic locking is a mechanism used by AWS IoT SiteWise to prevent accidental overwrites like the scenario above. Optimistic locking is a strategy to ensure the current version of asset model being updated or deleted, is the same as its current version in AWS IoT SiteWise. This protects asset model writes from being overwritten by accidental updates.

Follow these steps to perform asset model writes with optimistic locking:

Topics

Performing asset model writes with optimistic lock (console)

The procedure below describes how to perform asset model writes with an optimistic lock on the asset model's active version in the console.

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model or component model to update.
4. Choose **Edit**.

5. Make changes on the **Edit model** page.

6. Choose **Save**.

> **Note**
>
> Sometimes, one or more successful model updates have happened between when the user starts editing the model, and saves the made edits to the model.
> To ensure the user does not accidentally overwrite over new successful updates, the user's write is rejected. The console disables the **Save** button, and prompts the user to refresh the **Edit model** page. The user must update the new active version of the model again. The user must perform the following additional steps:

7. Choose **Refresh**.

8. Follow steps 5 and 6 again.

Performing asset model writes with optimistic lock (AWS CLI)

The procedure below describes how to perform asset model writes with optimistic locking in the AWS CLI.

1. **Fetch the ETag associated with current model definition**

   `ETag` is a unique token generated for each new representation of an asset model. Call DescribeAssetModel API to fetch the current asset model definition, and associated `ETag` from the response.

   During concurrent updates, users perform either successful updates (model in `ACTIVE` state), or unsuccessful updates (model in `FAILED` state). To ensure that an user does not accidentally overwrite a successful update, you must retrieve the active version of the asset model from Asset model versions, and get the `ETag` value.

   Run the following command:

   ```
   aws iotsitewise describe-asset-model --asset-model-id asset-model-id \
   --asset-model-version ACTIVE
   ```

   The response returns the following structure:

```
{
  "assetModelId": "String",
  "assetModelArn": "String",
  "assetModelName": "String",
  ...
  "eTag": "String"
}
```

> **Note**
>
> You must retrieve the latest version of the asset model and its `ETag` in order to not overwrite any updates.

2. **Perform UPDATE and DELETE operations with write conditions**

   The following asset model APIs support optimistic locking:

   - [UpdateAssetModel]
   - [DeleteAssetModel]
   - [CreateAssetModelCompositeModel]
   - [UpdateAssetModelCompositeModel]
   - [DeleteAssetModelCompositeModel]

   > **Note**
   >
   > The below scenarios use `UpdateAssetModel` API as a reference. The conditions apply to all the operations listed above.

   The below scenarios describe the different write conditions depending on concurrency control requirements:

   - Run the following command in order not to overwrite any successful updates. A new active version must not exist, since the last read active version. Replace `e-tag` with the `ETag` returned in the API operation used in the read of the active version.

```
aws iotsitewise update-asset-model \
  --asset-model-id asset-model-id \
  --if-match e-tag \
  --match-for-version-type ACTIVE \
  --cli-input-json file://model-payload.json
```

- When a model creation fails, an active version does not exist for it yet, because it's in a `FAILED` state. It is still possible to overwrite a new active version that is present, before your changes are committed. Run the following command to not overwrite a new active version, when an active version does not exist during your last read.

```
aws iotsitewise update-asset-model \
  --asset-model-id asset-model-id \
  --if-none-match "*" \
  --match-for-version-type ACTIVE \
  --cli-input-json file://model-payload.json
```

- Run the following command to avoid overwriting any successful or unsuccessful updates. This command defines a write condition which ensures that a latest version is not created since your last read latest version. Replace `e-tag` with the `ETag` returned in the API operation used in the read of the active version.

```
aws iotsitewise update-asset-model \
  --asset-model-id asset-model-id \
  --if-match eTag \
  --match-for-version-type LATEST \
  --cli-input-json file://model-payload.json
```

  If the write condition evaluates to `FALSE`, the write request fails with the `PreconditionFailedException`.

Delete assets and models in AWS IoT SiteWise

You can delete your assets, asset models, component models, and interfaces 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

- Delete assets in AWS IoT SiteWise
- Delete asset models, component models, and interfaces in AWS IoT SiteWise

Delete assets in AWS IoT SiteWise

You can use the AWS IoT SiteWise console or API to delete an asset no longer needed in your environment. Deleting an asset model also deletes all associated assets and component models. However, it's important to note that deleting an asset or model is a permanent action, and any data associated with the deleted resources is also be removed. Before deleting assets or models, it's recommended to review any dependencies or integrations that might be impacted and ensure that you have a backup of any important data.

Before you can delete an asset, you must first disassociate its child assets and disassociate it from its parent asset. For more information, see Associate and disassociate assets. If you use the AWS Command Line Interface (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. 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

- Delete an asset (console)
- Delete an asset (AWS CLI)

Delete 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.
5. On the asset's page, choose **Delete**.
6. In the **Delete asset** dialog box, do the following:

   a. Enter **`Delete`** to confirm deletion.

   b. Choose **Delete**.

Delete an asset (AWS 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 ID of the asset. This is the actual ID in UUID format, or the `externalId:myExternalId` if it has one. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

To delete an asset (AWS CLI)

1. Run the following command to list the asset's hierarchies. Replace *`asset-id`* with the ID or the external ID of the asset:

```
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:

```
{
  ...
  "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 or external ID of the asset and *hierarchy-id* with the ID or external ID of the hierarchy.

```
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 or external ID of the asset.

```
aws iotsitewise delete-asset --asset-id asset-id
```

Delete asset models, component models, and interfaces in AWS IoT SiteWise

You can use the AWS IoT SiteWise console or API to delete an asset model, component model, or interface.

Before you can delete an asset model, you must first delete all assets that were created from the asset model. Before you can delete an interface, you must first unlink it from all asset models that implement it.

When you delete an asset model or interface, its status is `DELETING` until the changes propagate. For more information, see Asset and model states. After the asset model or interface is deleted, you can't query that asset model or interface. If you do, the API returns an HTTP 404 response.

Topics

- Delete an asset model, component model, or interface (console)
- Delete an asset model, component model, or interface (AWS CLI)

Delete an asset model, component model, or interface (console)

You can use the AWS IoT SiteWise console to delete an asset model, component model, or interface.

To delete an asset model, component model, or interface (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models**.
3. Choose the asset model, component model, or interface to delete.
4. If deleting an asset model and it 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 Delete an asset (console).
5. On the model's page, choose **Delete**.
6. In the **Delete model** dialog box, do the following:

    a. Enter **`Delete`** to confirm deletion.

    b. Choose **Delete**.

Delete an asset model, component model, or interface (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to delete an asset model, component model, or interface.

Use the DeleteAssetModel operation to delete an asset model, component model, or interface. Specify the following parameter:

- `assetModelId` – The ID of the asset. This is the actual ID in UUID format, or the `externalId:myExternalId` if it has one. For more information, see Reference objects with external IDs in the *AWS IoT SiteWise User Guide*.

To delete an asset model (AWS CLI)

1. Run the following command to list all assets created from the model. Replace *asset-model-id* with the ID or the external 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 Delete an asset (AWS CLI).
3. Run the following command to delete the asset model. Replace *asset-model-id* with the ID or external ID of the asset model.

   ```
   aws iotsitewise delete-asset-model --asset-model-id asset-model-id
   ```

> **Important**
>
> To avoid deleting an asset model that was concurrently updated since the last read operation, you must define a conditional delete request. See Optimistic locking for asset model writes.

Bulk operations with assets and models

To work with a large number of assets or asset models, use bulk operations to bulk import and export resources to a different location. For example, you can create a data file that defines assets or asset models in an Amazon S3 bucket, and use bulk import to create or update them in AWS IoT SiteWise. Alternatively, if you have a large number of assets or asset models in AWS IoT SiteWise, you can export them to Amazon S3.

> **Note**
>
> You perform bulk operations in AWS IoT SiteWise by calling operations in the AWS IoT TwinMaker API. You can do this without setting up AWS IoT TwinMaker or creating an AWS IoT TwinMaker workspace. All you need is an Amazon S3 bucket where you can place your AWS IoT SiteWise content.

Topics

Key concepts and terminology

AWS IoT SiteWise bulk import and export features rely on the following concepts and terminology:

- **Import**: The action of moving assets or asset models from a file in an Amazon S3 bucket to AWS IoT SiteWise.
- **Export**: The action of moving assets or asset models from AWS IoT SiteWise to an Amazon S3 bucket.
- **Source**: The starting location of where you want to move content from.

  For example, an Amazon S3 bucket is an import source, and AWS IoT SiteWise is an export source.
- **Destination**: The desired location of where you want to move your content to.

  For example, an Amazon S3 bucket is an export destination, and AWS IoT SiteWise is an import destination.
- **AWS IoT SiteWise Schema**: This schema is used to import and export metadata from AWS IoT SiteWise.
- **Top-level resource:** An AWS IoT SiteWise resource that you can individually create or update, such as an asset or asset model.
- **Sub-resource:** A nested AWS IoT SiteWise resource within a top-level resource. Examples include properties, hierarchies, and composite models.

- **Metadata**: Key information required to import or export resources successfully. Examples of metadata are definitions of assets and asset models.
- **metadataTransferJob**: The object created when you run `CreateMetadataTransferJob`.

Supported functionality

This topic explains what you can do when you run a bulk operation. Bulk operations support the following functionality:

- **Top-level resource creation:** When you import an asset or asset model that doesn't define an ID, or whose ID doesn't match that of an existing one, then it will be created as a new resource.
- **Top-level resource replacement:** When you import an asset or asset model whose ID matches one that already exists, then it will replace the existing resource.
- **Subresource creation, replacement, or deletion:** When your import replaces a top-level resource such as an asset or asset model, then the new definition replaces all sub-resources, such as properties, hierarchies, or composite models.

  For example, if you update an asset model during a bulk import, and the updated version defines a property that wasn't present on the original, then a new property is created. If it defines a property that already exists, then the existing property will be updated. If the updated asset model omits a property that was present on the original, then the property is deleted.
- **No top-level resource deletion:** Bulk operations don't delete an asset or asset model. Bulk operations only create or update them.

Bulk operation prerequisites

This section explains bulk operation prerequisites, including AWS Identity and Access Management (IAM) permissions for exchanging resources between AWS services and your local machine. Before you start a bulk operation, complete the following prerequisite:

- Create an Amazon S3 bucket to store resources. For more information about using Amazon S3, see [What is Amazon S3?]

IAM permissions

To perform bulk operations, you must create an AWS Identity and Access Management (IAM) policy with permissions that allow the exchange of AWS resources between Amazon S3, AWS IoT SiteWise, and your local machine. For more information about creating IAM policies, see [Creating IAM policies](#).

To perform bulk operations, you need the following policies.

AWS IoT SiteWise policy

This policy allows access to the required AWS IoT SiteWise API actions for bulk operations:

```
{
    "Sid": "SiteWiseApiAccess",
    "Effect": "Allow",
    "Action": [
        "iotsitewise:CreateAsset",
        "iotsitewise:CreateAssetModel",
        "iotsitewise:UpdateAsset",
        "iotsitewise:UpdateAssetModel",
        "iotsitewise:UpdateAssetProperty",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssetModels",
        "iotsitewise:ListAssetProperties",
        "iotsitewise:ListAssetModelProperties",
        "iotsitewise:ListAssociatedAssets",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:AssociateAssets",
        "iotsitewise:DisassociateAssets",
        "iotsitewise:AssociateTimeSeriesToAssetProperty",
        "iotsitewise:DisassociateTimeSeriesFromAssetProperty",
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:BatchGetAssetPropertyValue",
        "iotsitewise:TagResource",
        "iotsitewise:UntagResource",
        "iotsitewise:ListTagsForResource",
        "iotsitewise:CreateAssetModelCompositeModel",
        "iotsitewise:UpdateAssetModelCompositeModel",
        "iotsitewise:DescribeAssetModelCompositeModel",
        "iotsitewise:DeleteAssetModelCompositeModel",
```

```
        "iotsitewise:ListAssetModelCompositeModels",
        "iotsitewise:ListCompositionRelationships",
        "iotsitewise:DescribeAssetCompositeModel"
    ],
    "Resource": "*"
}
```

AWS IoT TwinMaker policy

This policy allows access to the AWS IoT TwinMaker API operations that you use to work with bulk operations:

```
{
    "Sid": "MetadataTransferJobApiAccess",
    "Effect": "Allow",
    "Action": [
        "iottwinmaker:CreateMetadataTransferJob",
        "iottwinmaker:CancelMetadataTransferJob",
        "iottwinmaker:GetMetadataTransferJob",
        "iottwinmaker:ListMetadataTransferJobs"
    ],
    "Resource": "*"
}
```

Amazon S3 policy

This policy provides access to Amazon S3 buckets for transferring metadata for bulk operations.

For a specific Amazon S3 bucket

If you use one specific bucket for working with your bulk operations metadata, this policy provides access to that bucket:

```
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:AbortMultipartUpload",
        "s3:ListBucketMultipartUploads",
```

```
            "s3:ListMultipartUploadParts"
        ],
        "Resource": [
            "arn:aws:s3:::bucket name",
            "arn:aws:s3:::bucket name/*"
        ]
}
```

To allow any Amazon S3 bucket

If you will use many different buckets to work with your bulk operations metadata, this policy provides access to any bucket:

```
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:AbortMultipartUpload",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts"
    ],
    "Resource": "*"
}
```

For information about troubleshooting import and export operations, see Troubleshoot bulk import and export.

Run a bulk import job

Bulk import is the action of moving metadata into an AWS IoT SiteWise workspace. For example, bulk import can move metadata from a local file, or a file in an Amazon S3 bucket, to an AWS IoT SiteWise workspace.

Step 1: Prepare the file to import

Download the AWS IoT SiteWise native format file to import assets and the asset models. See AWS IoT SiteWise metadata transfer job schema for more details.

Step 2: Upload the prepared file to Amazon S3

Upload the file to Amazon S3. See Uploading a file to Amazon S3 in the *Amazon Simple Storage Service User Guide* for details.

Import metadata (console)

You can use the AWS IoT SiteWise console to bulk import metadata. Follow Step 1: Prepare the file to import and Step 2: Upload the prepared file to Amazon S3 to prepare a file that is ready to be imported.

Import data from Amazon S3 to AWS IoT SiteWise console

1. Navigate to the AWS IoT SiteWise console.
2. Choose **Bulk operations New** from the navigation pane.
3. Choose **New import** to start the import process.
4. On the **Import metadata** page:
   - Choose **Browse Amazon S3** to view the Amazon S3 bucket and files.
   - Navigate to the Amazon S3 bucket that contains the prepared import file.
   - Select the file to import.
   - Review the selected file, and choose **Import**.
5. The **Bulk operations on SiteWise metadata** page of the AWS IoT SiteWise console displays the newly created import job in the **Jobs progress** table.

Import metadata (AWS CLI)

To perform an import action, use the following procedure:

Import data from Amazon S3 to AWS CLI

1. Create a metadata file that specifies the resources you want to import, following the AWS IoT SiteWise metadata transfer job schema. Store this file in your Amazon S3 bucket.

   For examples of metadata files to import, see Import metadata examples.
2. Now create a JSON file with the request body. The request body specifies the source and destination for the transfer job. This file is separate from the file from the previous step. Make sure to specify your Amazon S3 bucket as a source and `iotsitewise` as the destination.

The following example shows the request body:

```
{
      "metadataTransferJobId": "your-transfer-job-Id",
      "sources": [{
          "type": "s3",
          "s3Configuration": {
              "location": "arn:aws:s3:::amzn-s3-demo-bucket/
your_import_metadata.json"
          }
      }],
      "destination": {
          "type": "iotsitewise"
      }
  }
```

3. Invoke the `CreateMetadataTransferJob` by running the following AWS CLI command. In this example, the request body file from the previous step is named `createMetadataTransferJobExport.json`.

```
aws iottwinmaker create-metadata-transfer-job --region us-east-1 \
  --cli-input-json file://createMetadataTransferJobImport.json
```

This will create a metadata transfer job, and begin the process of the transferring your selected resources.

Run a bulk export job

Bulk export is the action of moving metadata from an AWS IoT SiteWise workspace to an Amazon S3 bucket.

When you perform a bulk export of your AWS IoT SiteWise content to Amazon S3, you can specify filters to limit which specific asset models and assets you'd like to export.

The filters must be specified in an `iotSiteWiseConfiguration` section within the sources section of your JSON request.

> **Note**
>
> You can include multiple filters in your request. The bulk operation will export asset models and assets that match any of the filters.
> If you don't provide any filters, the bulk operation exports all of your asset models and assets.

Example request body with filters

```
{
      "metadataTransferJobId": "your-transfer-job-id",
      "sources": [
       {
        "type": "iotsitewise",
        "iotSiteWiseConfiguration": {
          "filters": [
           {
              "filterByAssetModel": {
                  "assetModelId": "asset model ID"
              }
            },
            {
              "filterByAssetModel": {
                  "assetModelId": "asset model ID",
                  "includeAssets": true
              }
            },
            {
              "filterByAssetModel": {
                  "assetModelId": "asset model ID",
                  "includeOffspring": true
               }
             }
           ]
          }
        }
       ],
       "destination": {
          "type": "s3",
          "s3Configuration": {
```

```
                "location": "arn:aws:s3:::amzn-s3-demo-bucket"
            }
        }
}
```

Export metadata (console)

The following procedure explains the console export action:

Create an export job in the AWS IoT SiteWise console

1. Navigate to the AWS IoT SiteWise console.
2. Choose **Bulk operations New** from the navigation pane.
3. Choose **New export** to start the export process.
4. On the **Export metadata** page:
   - Enter a name for the export job. This is the name used for the exported file in your Amazon S3 bucket.
   - Choose your resources to export, which sets the filters for the job:
     - Export all assets and asset models. Use filters on assets and asset models.
     - Export assets. Filter on your assets.
       - Select the asset to use for the export filter.
       - (Optional) Add the offspring or the associated asset model.
     - Export asset models. Filter on your asset models.
       - Select the asset model to use for the export filter.
       - (Optional) Add the offspring, or the associated asset or both.
     - Choose **Next**.
   - Navigate to the Amazon S3 bucket:
     - Choose **Browse Amazon S3** to view the Amazon S3 bucket and files.
     - Navigate to the Amazon S3 bucket where the file must be placed.
     - Choose **Next**.
   - Review the export job and choose **Export**.
5. The **Bulk operations on SiteWise metadata** page of the AWS IoT SiteWise console displays the newly created import job in the **Jobs progress** table.

For the different ways to use filters when exporting metadata, see Export metadata examples.

Export metadata (AWS CLI)

The following procedure explains the AWS CLI export action:

Export data from AWS IoT SiteWise to Amazon S3

1. Create a JSON file with your request body. The request body specifies the source and destination for the transfer job. The following example shows an example request body:

```
{
    "metadataTransferJobId": "your-transfer-job-Id",
    "sources": [{
        "type": "iotsitewise"
    }],
    "destination": {
        "type": "s3",
        "s3Configuration": {
            "location": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    }
}
```

   Make sure to specify your Amazon S3 bucket as the destination of the metadata transfer job.

   > **Note**
   >
   > This example will export all of your asset models and assets. To limit the export to specific asset models or assets, you can include filters in your request body. For more information about applying export filters, see Export metadata examples.

2. Save your request body file to use in the next step. In this example, the file is named `createMetadataTransferJobExport.json`.

3. Invoke the `CreateMetadataTransferJob` by running the following AWS CLI command:

```
aws iottwinmaker create-metadata-transfer-job --region us-east-1 \
         --cli-input-json file://createMetadataTransferJobExport.json
```

Replace the input JSON file `createMetadataTransferJobExport.json` with your own transfer file name.

Jobs progress tracking and error handling

A bulk process job takes time to process. Each job is processed in the order of AWS IoT SiteWise receiving the request. It is processed one-at-a-time for each account. When a job completes, the next in queue automatically starts processing. AWS IoT SiteWise resolves the jobs asynchronously and updates the status of each as it progresses. Each job has a status field that contains the state of the resource and an error message, if applicable.

The state can be one of the following values:

- `VALIDATING` – Validating the job including the submitted file format, and its contents.
- `PENDING` – The job is in a queue. You can cancel jobs in this state from the AWS IoT SiteWise console, but all other states will continue until the end.
- `RUNNING` – Processing the job. It is creating and updating resources as defined by the import file, or exporting resources based on the chosen export job filters. If canceled, any resource imported by this job is not deleted. See Review job progress and details (console) for more information.
- `CANCELLING` – The job is actively being cancelled.
- `ERROR` – One or more resources failed to process. Check the detailed job report for more information. See Inspect error details (console) for more information.
- `COMPLETED` – Job completed without errors.
- `CANCELLED` – The job is cancelled and not queued. If you cancelled a `RUNNING` job, resources already imported by this job at the time of cancellation is not deleted from AWS IoT SiteWise.

Topics

Jobs progress tracking

Review job progress and details (console)

See Import metadata (console) or Export metadata (console) to start a bulk job.

Job progress overview in the AWS IoT SiteWise console:

1. Navigate to the AWS IoT SiteWise console.
2. Choose **Bulk operations New** from the navigation pane.
3. The **Jobs progress** table in the AWS IoT SiteWise console, displays the list of bulk operation jobs.
4. The **Job type** column describes if it's an export or import job. The **Date imported** columns display the date that the job started.
5. The **Status** column displays the status of the job. You can select a job to see details about the job.
6. The selected job shows **Success** upon being successful, or a list of failure if the job failed. An error description is also displayed with each resource type.

Job details overview in the AWS IoT SiteWise console:

The **Jobs progress** table in the AWS IoT SiteWise console, displays the list of bulk operation jobs.

1. Choose a job to see more details.
2. For an **import** job, the `Data source ARN` represents the Amazon S3 location of the import file.
3. For an **export** job, the `Data destination ARN` represents the Amazon S3 location of the file after the export.
4. The `Status` and `Status reason`, provide additional details on the current job. See Jobs progress tracking and error handling for more details.
5. The `Queued position` represents the position of the job in the process queue. The jobs are processed one at a time. A queued position of 1, indicates that the job will be processed next.
6. The jobs details page also displays the job progress counts.

   - The job progress count types are:

     i. `Total resources` – Indicates the total count of assets in the transfer process.

     ii. `Succeeded` – Indicates the count of assets successfully transferred during the process.

     iii. `Failed` – Indicates the count assets that failed during the process.

     iv. `Skipped` – Indicates the count of assets that were skipped during the process.

7. A job status of PENDING or VALIDATING, displays all the jobs progress counts as –. This indicates that the jobs progress counts are being evaluated.
8. A job status of RUNNING displays the Total resources count, the job submitted for processing. The detailed counts (Succeeded, Failed, and Skipped), apply to the processed resources. The sum of the detailed counts is lesser than the Total resources count, until the job's status is COMPLETED or ERROR.
9. If a job's status is COMPLETED or ERROR, the Total resources count equals the sum of the detailed counts (Succeeded, Failed, and Skipped).
10. If a job's status is ERROR, check the **Job failures** table for details about the specific errors and failures. See [Inspect error details (console)](#) for more details.

Review job progress and details (AWS CLI)

After starting a bulk operation, you can check or update its status using the following API actions:

- To retrieve information on a specific job, use the [GetMetadataTransferJob](#) API action.

  Retrieve information with the `GetMetadataTransferJob` API:

  1. Create and run a transfer job. Call the GetMetadataTransferJob API.

     Example AWS CLI command:

     ```
     aws iottwinmaker get-metadata-transfer-job \
             --metadata-transfer-job-id your_metadata_transfer_job_id \
             --region your_region
     ```

  2. The GetMetadataTransferJob API returns a MetadataTransferJobProgress object with the following parameters:

     - **succeededCount** – Indicates the count of assets successfully transferred in the process.
     - **failedCount** – Indicates the count of assets that failed during the process.
     - **skippedCount** – Indicates the count of assets that were skipped during the process.
     - **totalCount** – Indicates the total count of assets in the transfer process.

     These parameters indicate the job progress status. If the status is RUNNING, they help track the number of resources still to be processed.

If you encounter schema validation errors, or if **failedCount** is greater than or equal to 1, the job progress state turns to ERROR. A full error report for the job is placed in your Amazon S3 bucket. See Inspect errors for AWS IoT SiteWise for more details.

- To list current jobs, use the ListMetadataTransferJobs API action.

  Use a JSON file to filter the returned jobs based on their current state. See the following procedure:

  1. To specify the filters you want to use, create an AWS CLI input JSON file. want to use:

     ```
     {
         "sourceType": "s3",
         "destinationType": "iottwinmaker",
         "filters": [{
             "state": "COMPLETED"
         }]
     }
     ```

     For a list of valid `state` values, see ListMetadataTransferJobsFilter in the *AWS IoT TwinMaker API Reference Guide*.

  2. Use the JSON file as an argument in the following AWS CLI example command:

     ```
     aws iottwinmaker list-metadata-transfer-job --region your_region \
             --cli-input-json file://ListMetadataTransferJobsExample.json
     ```

- To cancel a job, use the CancelMetadataTransferJob API action. This API cancels the specific metadata transfer job, without affecting any resources already exported or imported:

  ```
  aws iottwinmaker cancel-metadata-transfer-job \
          --region your_region \
          --metadata-transfer-job-id job-to-cancel-id
  ```

Inspect errors for AWS IoT SiteWise

Inspect error details (console)

Error details in the AWS IoT SiteWise console:

1. Navigate to the AWS IoT SiteWise console.
2. See the **Jobs progress** table in AWS IoT SiteWise console for a list of bulk operation jobs.
3. Select a job to view the job details.
4. If a job's status is `COMPLETED` or `ERROR`, the `Total resources` count equals the sum of the detailed counts (`Succeeded`, `Failed`, and `Skipped`).
5. If a job's status is `ERROR`, check the **Job failures** table for details about the specific errors and failures.
6. The **Job failures** table displays the content from the job report. The `Resource type` field indicates the location of the error or failures, such as the following:
   - For example, a validation error in the `Bulk operations template` in the `Resource type` field indicates that the import template and metadata schema file format don't match. See AWS IoT SiteWise metadata transfer job schema for more information.
   - A failed `Asset` in the `Resource type` field indicates that the asset is not created because of a conflict with another asset. See Common errors for information on AWS IoT SiteWise resource errors and conflicts.

Inspect error details (AWS CLI)

To handle and diagnose errors produced during a transfer job, see the following procedure about using the `GetMetadataTransferJob` API action:

1. After creating and running a transfer job, call GetMetadataTransferJob:

   ```
   aws iottwinmaker get-metadata-transfer-job \
           --metadata-transfer-job-id your_metadata_transfer_job_id \
           --region us-east-1
   ```

2. Once you see the state of the job turn to `COMPLETED`, you can start verifying the results of the job.
3. When you call `GetMetadataTransferJob`, it returns an object called `MetadataTransferJobProgress`.

The MetadataTransferJobProgress object contains the following parameters:

- **failedCount:** Indicates the count of assets that failed during the transfer process.
- **skippedCount:** Indicates the count of assets that were skipped during the transfer process.
- **succeededCount:** Indicates the count of assets that succeeded during the transfer process.
- **totalCount:** Indicates the total count of assets involved in the transfer process.

4. Additionally, the API call returns an element `reportUrl`, which contains a presigned URL. If your transfer job has any issues that you need to investigate further, visit this url.

Import metadata examples

This section shows how to create metadata files to import asset models and assets with a single bulk import operation.

Example of a bulk import

You can import many asset models and assets with a single bulk import operation. The following example shows how to create a metadata file to do this.

In this example scenario, you have various work sites that contain industrial robots in work cells.

The example defines two asset models:

- `RobotModel1`: This asset model represents a particular type of robot that you have in your work sites. The robot has a measurement property, `Temperature`.
- `WorkCell`: This asset model represents a collection of robots within one of your work sites. The asset model defines a hierarchy, `robotHierarchyOEM1`, to represent the relationship that a work cell contains robots.

The example also defines some assets:

- `WorkCell1`: a work cell within your Boston site
- `RobotArm123456`: a robot within that work cell
- `RobotArm987654`: another robot within that work cell

The following JSON metadata file defines these asset models and assets. Running a bulk import with this metadata creates the asset models and assets within AWS IoT SiteWise, including their hierarchical relationships.

Metadata file for import

```
{
    "assetModels": [
        {
            "assetModelExternalId": "Robot.OEM1.3536",
            "assetModelName": "RobotModel1",
            "assetModelProperties": [
                {
                    "dataType": "DOUBLE",
                    "externalId": "Temperature",
                    "name": "Temperature",
                    "type": {
                        "measurement": {
                            "processingConfig": {
                                "forwardingConfig": {
                                    "state": "ENABLED"
                                }
                            }
                        }
                    },
                    "unit": "fahrenheit"
                }
            ]
        },
        {
            "assetModelExternalId": "ISA95.WorkCell",
            "assetModelName": "WorkCell",
            "assetModelProperties": [],
            "assetModelHierarchies": [
                {
                    "externalId": "workCellHierarchyWithOEM1Robot",
                    "name": "robotHierarchyOEM1",
                    "childAssetModelExternalId": "Robot.OEM1.3536"
                }
            ]
        }
    ],
    "assets": [
```

```
        {
            "assetExternalId": "Robot.OEM1.3536.123456",
            "assetName": "RobotArm123456",
            "assetModelExternalId": "Robot.OEM1.3536"
        },
        {
            "assetExternalId": "Robot.OEM1.3536.987654",
            "assetName": "RobotArm987654",
            "assetModelExternalId": "Robot.OEM1.3536"
        },
        {
            "assetExternalId": "BostonSite.Area1.Line1.WorkCell1",
            "assetName": "WorkCell1",
            "assetModelExternalId": "ISA95.WorkCell",
            "assetHierarchies": [
                {
                    "externalId": "workCellHierarchyWithOEM1Robot",
                    "childAssetExternalId": "Robot.OEM1.3536.123456"
                },
                {
                    "externalId": "workCellHierarchyWithOEM1Robot",
                    "childAssetExternalId": "Robot.OEM1.3536.987654"
                }
            ]
        }
    ]
}
```

Example of initial on-boarding of models and assets

In this example scenario, you have various work sites that contain industrial robots in a company.

The example defines multiple asset models:

- `Sample_Enterprise` – This asset model represents the company that the sites are part of. The asset model defines a hierarchy, `Enterprise to Site`, to represent the relationship of the sites to the enterprise.
- `Sample_Site` – This asset model represents the manufacturing sites within the company. The asset model defines a hierarchy, `Site to Line`, to represent the relationship of the lines to the site.

- `Sample_Welding Line` – This asset model represents an assembly line within work sites. The asset model defines a hierarchy, `Line to Robot`, to represent the relationship of the robots to the line.
- `Sample_Welding Robot` – This asset model represents a particular type of robot in your work sites.

The example also defines assets based on the asset models.

- `Sample_AnyCompany Motor` – This asset is created from `Sample_Enterprise` asset model.
- `Sample_Chicago` – This asset is created from `Sample_Site` asset model.
- `Sample_Welding Line 1` – This asset is created from `Sample_Welding Line` asset model.
- `Sample_Welding Robot 1` – This asset is created from `Sample_Welding Robot` asset model.
- `Sample_Welding Robot 2` – This asset is created from `Sample_Welding Robot` asset model.

The following JSON metadata file defines these asset models and assets. Running a bulk import with this metadata creates the asset models and assets within AWS IoT SiteWise, including their hierarchical relationships.

JSON file to onboard assets and models for import

```
{
    "assetModels": [
        {
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetModelName": "Sample_Welding Robot",
            "assetModelProperties": [
                {
                    "dataType": "STRING",
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "name": "Serial Number",
                    "type": {
                        "attribute": {
                            "defaultValue": "-"
                        }
                    },
                    "unit": "-"
                },
                {
```

```
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "name": "CycleCount",
                    "type": {
                        "measurement": {}
                    },
                    "unit": "EA"
                },
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "name": "Joint 1 Current",
                    "type": {
                        "measurement": {}
                    },
                    "unit": "Amps"
                },
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Joint_1_Max_Current",
                    "name": "Max Joint 1 Current",
                    "type": {
                        "metric": {
                            "expression": "max(joint1current)",
                            "variables": [
                                {
                                    "name": "joint1current",
                                    "value": {
                                        "propertyExternalId":
 "External_Id_Welding_Robot_Joint_1_Current"
                                    }
                                }
                            ],
                            "window": {
                                "tumbling": {
                                    "interval": "5m"
                                }
                            }
                        }
                    },
                    "unit": "Amps"
                }
            ]
        },
```

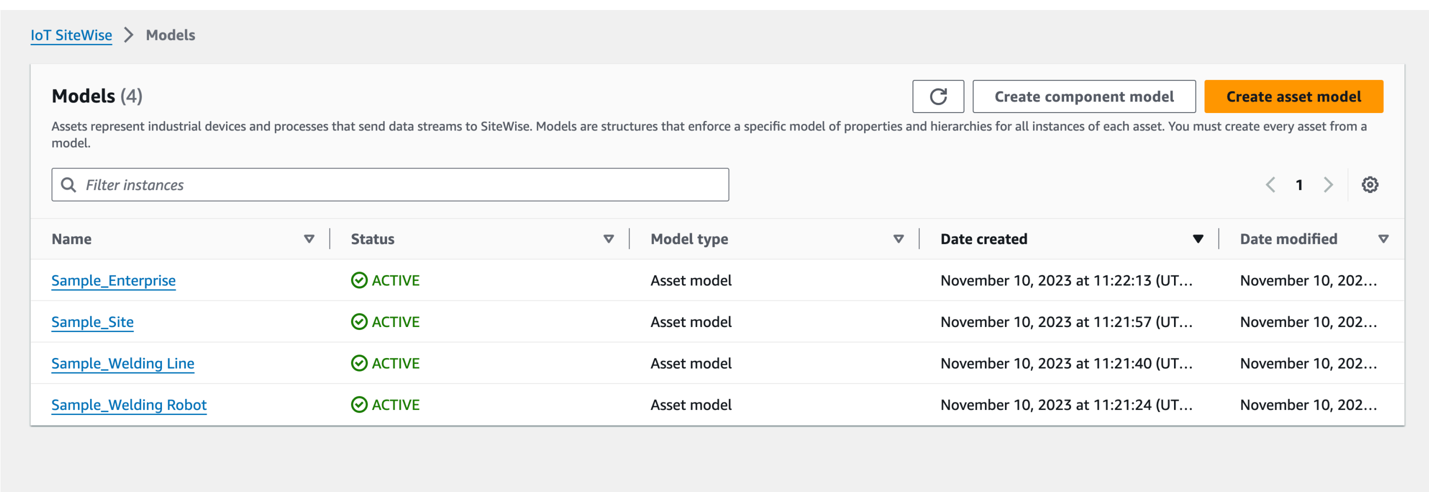
```
        {
            "assetModelExternalId": "External_Id_Welding_Line",
            "assetModelName": "Sample_Welding Line",
            "assetModelProperties": [
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Line_Availability",
                    "name": "Availability",
                    "type": {
                        "measurement": {}
                    },
                    "unit": "%"
                }
            ],
            "assetModelHierarchies": [
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "name": "Line to Robot",
                    "childAssetModelExternalId": "External_Id_Welding_Robot"
                }
            ]
        },
        {
            "assetModelExternalId": "External_Id_Site",
            "assetModelName": "Sample_Site",
            "assetModelProperties": [
                {
                    "dataType": "STRING",
                    "externalId": "External_Id_Site_Street_Address",
                    "name": "Street Address",
                    "type": {
                        "attribute": {
                            "defaultValue": "-"
                        }
                    },
                    "unit": "-"
                }
            ],
            "assetModelHierarchies": [
                {
                    "externalId": "External_Id_Site_TO_Line",
                    "name": "Site to Line",
                    "childAssetModelExternalId": "External_Id_Welding_Line"
                }
```

```
            ]
        },
        {
            "assetModelExternalId": "External_Id_Enterprise",
            "assetModelName": "Sample_Enterprise",
            "assetModelProperties": [
                {
                    "dataType": "STRING",
                    "name": "Company Name",
                    "externalId": "External_Id_Enterprise_Company_Name",
                    "type": {
                        "attribute": {
                            "defaultValue": "-"
                        }
                    },
                    "unit": "-"
                }
            ],
            "assetModelHierarchies": [
                {
                    "externalId": "External_Id_Enterprise_TO_Site",
                    "name": "Enterprise to Site",
                    "childAssetModelExternalId": "External_Id_Site"
                }
            ]
        }
    ],
    "assets": [
        {
            "assetExternalId": "External_Id_Welding_Robot_1",
            "assetName": "Sample_Welding Robot 1",
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetProperties": [
                {
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "attributeValue": "S1000"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "alias": "AnyCompany/Chicago/Welding Line/S1000/Count"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "alias": "AnyCompany/Chicago/Welding Line/S1000/1/Current"
```

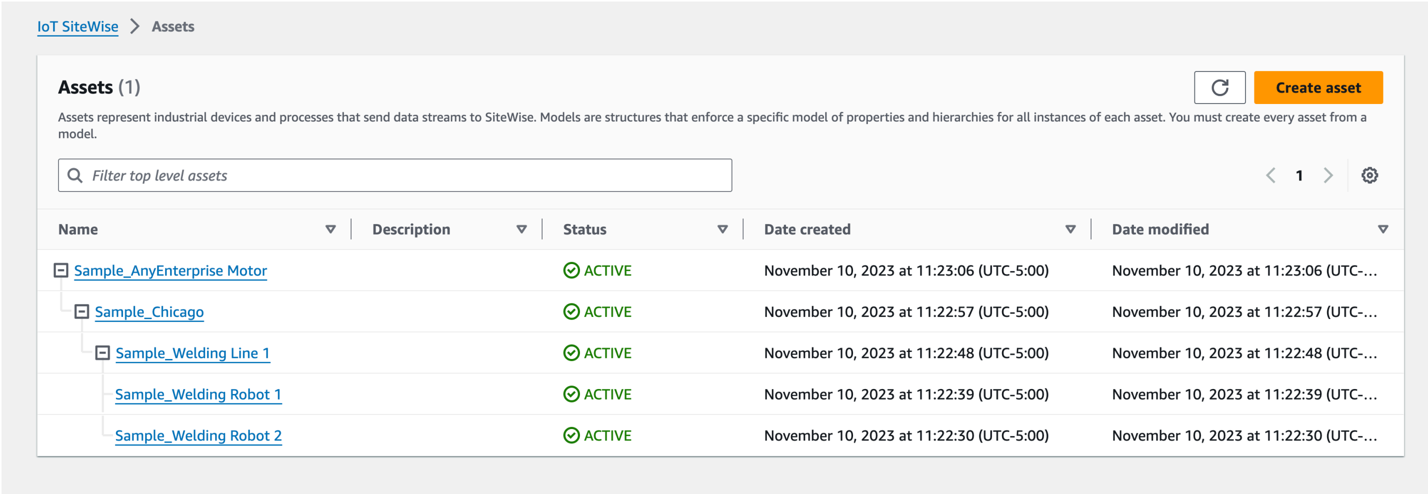
```
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Welding_Robot_2",
            "assetName": "Sample_Welding Robot 2",
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetProperties": [
                {
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "attributeValue": "S2000"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "alias": "AnyCompany/Chicago/Welding Line/S2000/Count"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "alias": "AnyCompany/Chicago/Welding Line/S2000/1/Current"
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Welding_Line_1",
            "assetName": "Sample_Welding Line 1",
            "assetModelExternalId": "External_Id_Welding_Line",
            "assetProperties": [
                {
                    "externalId": "External_Id_Welding_Line_Availability",
                    "alias": "AnyCompany/Chicago/Welding Line/Availability"
                }
            ],
            "assetHierarchies": [
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_1"
                },
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_2"
                }
            ]
        },
        {
```

```
                "assetExternalId": "External_Id_Site_Chicago",
                "assetName": "Sample_Chicago",
                "assetModelExternalId": "External_Id_Site",
                "assetHierarchies": [
                    {
                        "externalId": "External_Id_Site_TO_Line",
                        "childAssetExternalId": "External_Id_Welding_Line_1"
                    }
                ]
            },
            {
                "assetExternalId": "External_Id_Enterprise_AnyCompany",
                "assetName": "Sample_AnyEnterprise Motor",
                "assetModelExternalId": "External_Id_Enterprise",
                "assetHierarchies": [
                    {
                        "externalId": "External_Id_Enterprise_TO_Site",
                        "childAssetExternalId": "External_Id_Site_Chicago"
                    }
                ]
            }
        ]
}
```

The following screenshot is of models that display in the AWS IoT SiteWise console after you run the previous code example.



The following screenshot is of models, assets, and hierarchies that display in the AWS IoT SiteWise console after you run the previous code example.



Example of onboarding additional assets

This example defines additional assets to import to an existing asset model in your account:

- `Sample_Welding Line 2` – This asset is created from `Sample_Welding Line` asset model.
- `Sample_Welding Robot 3`– This asset is created from `Sample_Welding Robot` asset model.
- `Sample_Welding Robot 4`– This asset is created from `Sample_Welding Robot` asset model.

To create the initial assets for this example, see Example of initial on-boarding of models and assets.

The following JSON metadata file defines these asset models and assets. Running a bulk import with this metadata creates the asset models and assets within AWS IoT SiteWise, including their hierarchical relationships.

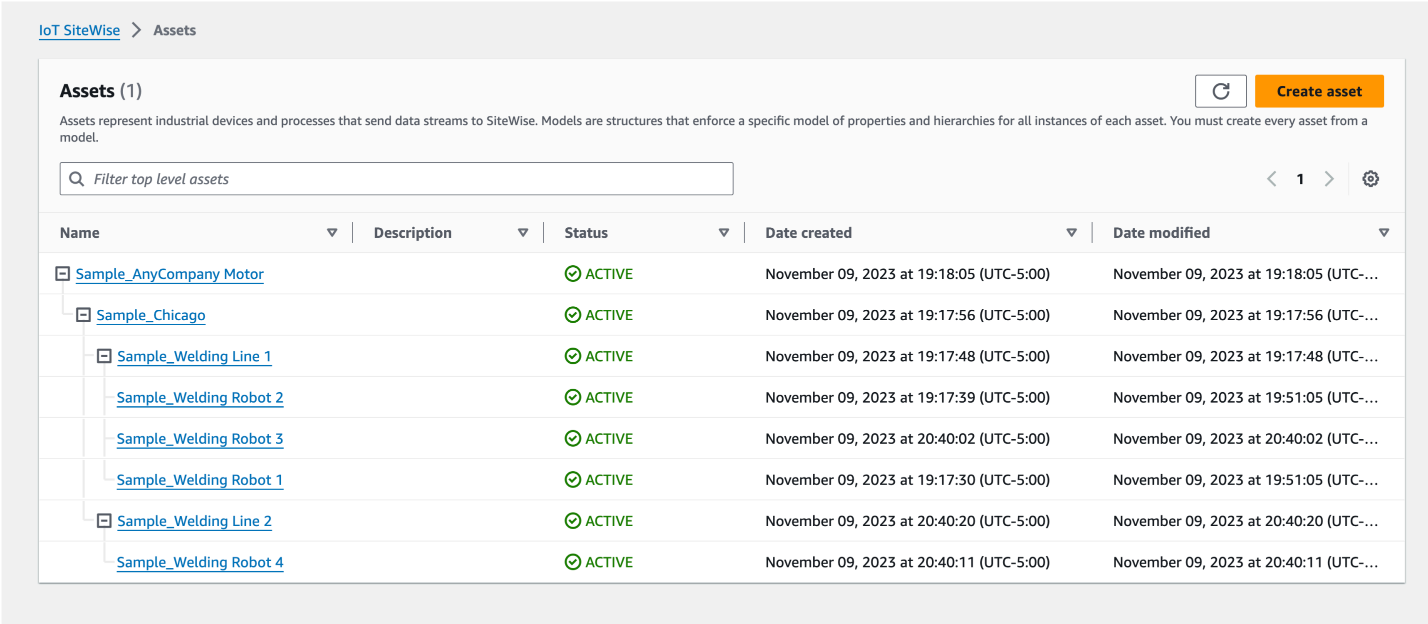
JSON file to onboard additional assets

```
{
    "assets": [
        {
            "assetExternalId": "External_Id_Welding_Robot_3",
            "assetName": "Sample_Welding Robot 3",
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetProperties": [
                {
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "attributeValue": "S3000"
```

```
                },
                {
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "alias": "AnyCompany/Chicago/Welding Line/S3000/Count"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "alias": "AnyCompany/Chicago/Welding Line/S3000/1/Current"
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Welding_Robot_4",
            "assetName": "Sample_Welding Robot 4",
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetProperties": [
                {
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "attributeValue": "S4000"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "alias": "AnyCompany/Chicago/Welding Line/S4000/Count"
                },
                {
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "alias": "AnyCompany/Chicago/Welding Line/S4000/1/Current"
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Welding_Line_1",
            "assetName": "Sample_Welding Line 1",
            "assetModelExternalId": "External_Id_Welding_Line",
            "assetHierarchies": [
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_1"
                },
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_2"
                },
                {
```

```
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_3"
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Welding_Line_2",
            "assetName": "Sample_Welding Line 2",
            "assetModelExternalId": "External_Id_Welding_Line",
            "assetHierarchies": [
                {
                    "externalId": "External_Id_Welding_Line_TO_Robot",
                    "childAssetExternalId": "External_Id_Welding_Robot_4"
                }
            ]
        },
        {
            "assetExternalId": "External_Id_Site_Chicago",
            "assetName": "Sample_Chicago",
            "assetModelExternalId": "External_Id_Site",
            "assetHierarchies": [
                {
                    "externalId": "External_Id_Site_TO_Line",
                    "childAssetExternalId": "External_Id_Welding_Line_1"
                },
                {
                    "externalId": "External_Id_Site_TO_Line",
                    "childAssetExternalId": "External_Id_Welding_Line_2"
                }
            ]
        }
    ]
}
```

The following screenshot is of models, assets, and hierarchies that display in the AWS IoT SiteWise console after you run the previous code example.



Example of onboarding new properties

This example defines new properties on existing asset models. See Example of onboarding additional assets to onboard additional assets and models.

- `Joint 1 Temperature` – This property is added to the `Sample_Welding Robot` asset model. This new property will also propagate to each asset created from the `Sample_Welding Robot` asset model.

To add a new property to an existing asset model, see the following JSON metadata file example. As shown in the JSON, the entire existing `Sample_Welding Robot` asset model definition must be provided along with the new property. If the entire property list from the existing definition is not provided, AWS IoT SiteWise deletes the omitted properties.

JSON file to onboard new properties

This example adds a new property `Joint 1 Temperature` to the asset model.

```
{
    "assetModels": [
        {
            "assetModelExternalId": "External_Id_Welding_Robot",
            "assetModelName": "Sample_Welding Robot",
            "assetModelProperties": [
```

```
                {
                    "dataType": "STRING",
                    "externalId": "External_Id_Welding_Robot_Serial_Number",
                    "name": "Serial Number",
                    "type": {
                        "attribute": {
                            "defaultValue": "-"
                        }
                    },
                    "unit": "-"
                },
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Cycle_Count",
                    "name": "CycleCount",
                    "type": {
                        "measurement": {}
                    },
                    "unit": "EA"
                },
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Joint_1_Current",
                    "name": "Joint 1 Current",
                    "type": {
                        "measurement": {}
                    },
                    "unit": "Amps"
                },
                {
                    "dataType": "DOUBLE",
                    "externalId": "External_Id_Welding_Robot_Joint_1_Max_Current",
                    "name": "Max Joint 1 Current",
                    "type": {
                        "metric": {
                            "expression": "max(joint1current)",
                            "variables": [
                                {
                                    "name": "joint1current",
                                    "value": {
                                        "propertyExternalId":
 "External_Id_Welding_Robot_Joint_1_Current"
                                    }
                                }
```

```
                              ],
                              "window": {
                                  "tumbling": {
                                      "interval": "5m"
                                  }
                              }
                          }
                      },
                      "unit": "Amps"
                  },
                  {
                      "dataType": "DOUBLE",
                      "externalId": "External_Id_Welding_Robot_Joint_1_Temperature",
                      "name": "Joint 1 Temperature",
                      "type": {
                          "measurement": {}
                      },
                      "unit": "degC"
                  }
              ]
          }
      ]
}
```

Example of managing data streams

This example shows two ways of managing data streams associated with an asset property. When renaming an asset property alias, there are two options for the historical data currently stored in the asset property's data stream.

- Option one – Keep the current data stream and rename the alias alone, allowing the historical data to be accessible with the new alias.

  In the JSON metadata file example, the asset property with ID `External_Id_Welding_Robot_Cycle_Count` changes its alias to `AnyCompany/Chicago/Welding Line/S3000/Count-Updated`. The historical data for this asset property remains the same after this change.
- Option two – Assign a new data stream to the asset property which is accessible with the new alias. The old data stream along with its historical data is still accessible with the old alias, but not associated with any asset property.

In the JSON metadata file example, the asset property with ID `External_Id_Welding_Robot_Joint_1_Current` changes its alias to `AnyCompany/Chicago/Welding Line/S4999/1/Current`. This time the additional value `retainDataOnAliasChange` is present and set to `False`. With this setting, the original data stream is disassociated from the asset property, and a new data stream is created containing no historical data.

To access the old data stream with the original historical data, in the AWS Console Home, go to the *Data Streams* page and search for the old alias `AnyCompany/Chicago/Welding Line/S3000/1/Current`.

JSON file to update property aliases

```
{
    "assetExternalId": "External_Id_Welding_Robot_3",
    "assetName": "Sample_Welding Robot 3",
    "assetModelExternalId": "External_Id_Welding_Robot",
    "assetProperties": [
        {
            "externalId": "External_Id_Welding_Robot_Serial_Number",
            "attributeValue": "S3000"
        },
        {
            "externalId": "External_Id_Welding_Robot_Cycle_Count",
            "alias": "AnyCompany/Chicago/Welding Line/S3000/Count-Updated"
        },
        {
            "externalId": "External_Id_Welding_Robot_Joint_1_Current",
            "alias": "AnyCompany/Chicago/Welding Line/S4999/1/Current",
            "retainDataOnAliasChange": "FALSE"
        }
    ]
}
```

Export metadata examples

When you perform a bulk export of your AWS IoT SiteWise content to Amazon S3, you can specify *filters* to limit which specific asset models and assets you'd like to export.

You specify the filters in an `iotSiteWiseConfiguration` section within the `sources` section of your request body.

Note

You can include multiple filters. The bulk operation will export any asset model or asset that matches any of the filters.
If you don't provide any filters, then the operation will export all of your asset models and assets.

```
{
    "metadataTransferJobId": "your-transfer-job-id",
    "sources": [{
        "type": "iotsitewise",
        "iotSiteWiseConfiguration": {
            "filters": [{
                list of filters
            }]
        }
    }],
    "destination": {
        "type": "s3",
        "s3Configuration": {
            "location": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    }
}
```

Filter by asset model

You can filter a specific asset model. You can also include all assets using that model, or all asset models within its hierarchy. You can't include both assets and hierarchy.

For more information about hierarchies, see Define asset model hierarchies.

Asset model

This filter includes the specified asset model:

```
"filterByAssetModel": {
```

```
    "assetModelId": "asset model ID"
}
```

Asset model and its assets

This filter includes the specified asset model, along with all assets using that asset model:

```
"filterByAssetModel": {
    "assetModelId": "asset model ID",
    "includeAssets": true
}
```

Asset model and its hierarchy

This filter includes the specified asset model, along with all associated asset models in its hierarchy:

```
"filterByAssetModel": {
    "assetModelId": "asset model ID",
    "includeOffspring": true
}
```

Filter by asset

You can filter a specific asset. You can also include its asset model, or all associated assets within its hierarchy. You can't include both asset model and hierarchy.

For more information about hierarchies, see Define asset model hierarchies.

Asset

This filter includes the specified asset:

```
"filterByAsset": {
    "assetId": "asset ID"
}
```

Asset and its asset model

This filter includes the specified asset, along with the asset model it uses:

```
"filterByAsset": {
```

```
    "assetId": "asset ID",
    "includeAssetModel": true
}
```

Asset and its hierarchy

This filter includes the specified asset, along with all associated assets in its hierarchy:

```
"filterByAsset": {
    "assetId": "asset ID",
    "includeOffspring": true
}
```

AWS IoT SiteWise metadata transfer job schema

Use the AWS IoT SiteWise metadata transfer job schema for reference when performing your own bulk import and export operations:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "IoTSiteWise",
  "description": "Metadata transfer job resource schema for IoTSiteWise",
  "definitions": {
    "Name": {
      "type": "string",
      "minLength": 1,
      "maxLength": 256,
      "pattern": "[^\\u0000-\\u001F\\u007F]+"
    },
    "Description": {
      "type": "string",
      "minLength": 1,
      "maxLength": 2048,
      "pattern": "[^\\u0000-\\u001F\\u007F]+"
    },
    "ID": {
      "type": "string",
      "minLength": 36,
      "maxLength": 36,
      "pattern": "^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$"
    },
    "ExternalId": {
```

```
      "type": "string",
      "minLength": 2,
      "maxLength": 128,
      "pattern": "[a-zA-Z0-9_][a-zA-Z_\\-0-9.:]*[a-zA-Z0-9_]+"
    },
    "AttributeValue": {
      "description": "The value of the property attribute.",
      "type": "string",
      "pattern": "[^\\u0000-\\u001F\\u007F]+"
    },
    "PropertyUnit": {
      "description": "The unit of measure (such as Newtons or RPM) of the asset
 property.",
      "type": "string",
      "minLength": 1,
      "maxLength": 256,
      "pattern": "[^\\u0000-\\u001F\\u007F]+"
    },
    "PropertyAlias": {
      "description": "The property alias that identifies the property.",
      "type": "string",
      "minLength": 1,
      "maxLength": 1000,
      "pattern": "[^\\u0000-\\u001F\\u007F]+"
    },
    "AssetProperty": {
      "description": "The asset property's definition, alias, unit, and notification
 state.",
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "id"
          ]
        },
        {
          "required": [
            "externalId"
          ]
        }
      ],
      "properties": {
        "id": {
```

```
          "description": "The ID of the asset property.",
          "$ref": "#/definitions/ID"
        },
        "externalId": {
          "description": "The ExternalID of the asset property.",
          "$ref": "#/definitions/ExternalId"
        },
        "alias": {
          "$ref": "#/definitions/PropertyAlias"
        },
        "unit": {
          "$ref": "#/definitions/PropertyUnit"
        },
        "attributeValue": {
          "$ref": "#/definitions/AttributeValue"
        },
        "retainDataOnAliasChange": {
          "type": "string",
          "default": "TRUE",
          "enum": [
            "TRUE",
            "FALSE"
          ]
        },
        "propertyNotificationState": {
          "description": "The MQTT notification state (ENABLED or DISABLED) for this
asset property.",
          "type": "string",
          "enum": [
            "ENABLED",
            "DISABLED"
          ]
        }
      }
    },
    "AssetHierarchy": {
      "description": "A hierarchy specifies allowed parent/child asset relationships.",
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "id",
            "childAssetId"
```

```
          ]
        },
        {
          "required": [
            "externalId",
            "childAssetId"
          ]
        },
        {
          "required": [
            "id",
            "childAssetExternalId"
          ]
        },
        {
          "required": [
            "externalId",
            "childAssetExternalId"
          ]
        }
      ],
      "properties": {
        "id": {
          "description": "The ID of a hierarchy in the parent asset's model.",
          "$ref": "#/definitions/ID"
        },
        "externalId": {
          "description": "The ExternalID of a hierarchy in the parent asset's model.",
          "$ref": "#/definitions/ExternalId"
        },
        "childAssetId": {
          "description": "The ID of the child asset to be associated.",
          "$ref": "#/definitions/ID"
        },
        "childAssetExternalId": {
          "description": "The ExternalID of the child asset to be associated.",
          "$ref": "#/definitions/ExternalId"
        }
      }
    },
    "Tag": {
      "type": "object",
      "additionalProperties": false,
      "required": [
```

```
        "key",
        "value"
      ],
      "properties": {
        "key": {
          "type": "string"
        },
        "value": {
          "type": "string"
        }
      }
    },
    "AssetModelType": {
      "type": "string",
      "default": null,
      "enum": [
        "ASSET_MODEL",
        "COMPONENT_MODEL"
      ]
    },
    "AssetModelCompositeModel": {
      "description": "Contains a composite model definition in an asset model. This
 composite model definition is applied to all assets created from the asset model.",
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "id"
          ]
        },
        {
          "required": [
            "externalId"
          ]
        }
      ],
      "required": [
        "name",
        "type"
      ],
      "properties": {
        "id": {
          "description": "The ID of the asset model composite model.",
```

```
          "$ref": "#/definitions/ID"
        },
        "externalId": {
          "description": "The ExternalID of the asset model composite model.",
          "$ref": "#/definitions/ExternalId"
        },
        "parentId": {
          "description": "The ID of the parent asset model composite model.",
          "$ref": "#/definitions/ID"
        },
        "parentExternalId": {
          "description": "The ExternalID of the parent asset model composite model.",
          "$ref": "#/definitions/ExternalId"
        },
        "composedAssetModelId": {
          "description": "The ID of the composed asset model.",
          "$ref": "#/definitions/ID"
        },
        "composedAssetModelExternalId": {
          "description": "The ExternalID of the composed asset model.",
          "$ref": "#/definitions/ExternalId"
        },
        "description": {
          "description": "A description for the asset composite model.",
          "$ref": "#/definitions/Description"
        },
        "name": {
          "description": "A unique, friendly name for the asset composite model.",
          "$ref": "#/definitions/Name"
        },
        "type": {
          "description": "The type of the composite model. For alarm composite models,
 this type is AWS/ALARM.",
          "$ref": "#/definitions/Name"
        },
        "properties": {
          "description": "The property definitions of the asset model.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetModelProperty"
          }
        }
      }
    },
```

```
    "AssetModelProperty": {
      "description": "Contains information about an asset model property.",
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "id"
          ]
        },
        {
          "required": [
            "externalId"
          ]
        }
      ],
      "required": [
        "name",
        "dataType",
        "type"
      ],
      "properties": {
        "id": {
          "description": "The ID of the asset model property.",
          "$ref": "#/definitions/ID"
        },
        "externalId": {
          "description": "The ExternalID of the asset model property.",
          "$ref": "#/definitions/ExternalId"
        },
        "name": {
          "description": "The name of the asset model property.",
          "$ref": "#/definitions/Name"
        },
        "dataType": {
          "description": "The data type of the asset model property.",
          "$ref": "#/definitions/DataType"
        },
        "dataTypeSpec": {
          "description": "The data type of the structure for this property.",
          "$ref": "#/definitions/Name"
        },
        "unit": {
```

```
          "description": "The unit of the asset model property, such as Newtons or
 RPM.",
          "type": "string",
          "minLength": 1,
          "maxLength": 256,
          "pattern": "[^\\u0000-\\u001F\\u007F]+"
        },
        "type": {
          "description": "The property type",
          "$ref": "#/definitions/PropertyType"
        }
      }
    },
    "DataType": {
      "type": "string",
      "enum": [
        "STRING",
        "INTEGER",
        "DOUBLE",
        "BOOLEAN",
        "STRUCT"
      ]
    },
    "PropertyType": {
      "description": "Contains a property type, which can be one of attribute,
 measurement, metric, or transform.",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "attribute": {
          "$ref": "#/definitions/Attribute"
        },
        "transform": {
          "$ref": "#/definitions/Transform"
        },
        "metric": {
          "$ref": "#/definitions/Metric"
        },
        "measurement": {
          "$ref": "#/definitions/Measurement"
        }
      }
    },
    "Attribute": {
```

```
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "defaultValue": {
          "type": "string",
          "pattern": "[^\\u0000-\\u001F\\u007F]+"
        }
      }
    },
    "Transform": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "expression",
        "variables"
      ],
      "properties": {
        "expression": {
          "description": "The mathematical expression that defines the transformation
 function.",
          "type": "string",
          "minLength": 1,
          "maxLength": 1024
        },
        "variables": {
          "description": "The list of variables used in the expression.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/ExpressionVariable"
          }
        },
        "processingConfig": {
          "$ref": "#/definitions/TransformProcessingConfig"
        }
      }
    },
    "TransformProcessingConfig": {
      "description": "The processing configuration for the given transform property.",
      "type": "object",
      "additionalProperties": false,
      "required": [
        "computeLocation"
      ],
      "properties": {
```

```
        "computeLocation": {
          "description": "The compute location for the given transform property.",
          "$ref": "#/definitions/ComputeLocation"
        },
        "forwardingConfig": {
          "description": "The forwarding configuration for a given property.",
          "$ref": "#/definitions/ForwardingConfig"
        }
      }
    },
    "Metric": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "expression",
        "variables",
        "window"
      ],
      "properties": {
        "expression": {
          "description": "The mathematical expression that defines the metric
 aggregation function.",
          "type": "string",
          "minLength": 1,
          "maxLength": 1024
        },
        "variables": {
          "description": "The list of variables used in the expression.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/ExpressionVariable"
          }
        },
        "window": {
          "description": "The window (time interval) over which AWS IoT SiteWise
 computes the metric's aggregation expression",
          "$ref": "#/definitions/MetricWindow"
        },
        "processingConfig": {
          "$ref": "#/definitions/MetricProcessingConfig"
        }
      }
    },
    "MetricProcessingConfig": {
```

```
      "description": "The processing configuration for the metric.",
      "type": "object",
      "additionalProperties": false,
      "required": [
        "computeLocation"
      ],
      "properties": {
        "computeLocation": {
          "description": "The compute location for the given metric property.",
          "$ref": "#/definitions/ComputeLocation"
        }
      }
    },
    "ComputeLocation": {
      "type": "string",
      "enum": [
        "EDGE",
        "CLOUD"
      ]
    },
    "ForwardingConfig": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "state"
      ],
      "properties": {
        "state": {
          "type": "string",
          "enum": [
            "ENABLED",
            "DISABLED"
          ]
        }
      }
    },
    "MetricWindow": {
      "description": "Contains a time interval window used for data aggregate
 computations (for example, average, sum, count, and so on).",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "tumbling": {
          "description": "The tumbling time interval window.",
```

```
          "type": "object",
          "additionalProperties": false,
          "required": [
            "interval"
          ],
          "properties": {
            "interval": {
              "description": "The time interval for the tumbling window.",
              "type": "string",
              "minLength": 2,
              "maxLength": 23
            },
            "offset": {
              "description": "The offset for the tumbling window.",
              "type": "string",
              "minLength": 2,
              "maxLength": 25
            }
          }
        }
      }
    },
    "ExpressionVariable": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "name",
        "value"
      ],
      "properties": {
        "name": {
          "description": "The friendly name of the variable to be used in the 
expression.",
          "type": "string",
          "minLength": 1,
          "maxLength": 64,
          "pattern": "^[a-z][a-z0-9_]*$"
        },
        "value": {
          "description": "The variable that identifies an asset property from which to 
use values.",
          "$ref": "#/definitions/VariableValue"
        }
      }
```

```
    },
    "VariableValue": {
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "propertyId"
          ]
        },
        {
          "required": [
            "propertyExternalId"
          ]
        }
      ],
      "properties": {
        "propertyId": {
          "$ref": "#/definitions/ID"
        },
        "propertyExternalId": {
          "$ref": "#/definitions/ExternalId"
        },
        "hierarchyId": {
          "$ref": "#/definitions/ID"
        },
        "hierarchyExternalId": {
          "$ref": "#/definitions/ExternalId"
        }
      }
    },
    "Measurement": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "processingConfig": {
          "$ref": "#/definitions/MeasurementProcessingConfig"
        }
      }
    },
    "MeasurementProcessingConfig": {
      "type": "object",
      "additionalProperties": false,
      "required": [
```

```
        "forwardingConfig"
      ],
      "properties": {
        "forwardingConfig": {
          "description": "The forwarding configuration for the given measurement 
property.",
          "$ref": "#/definitions/ForwardingConfig"
        }
      }
    },
    "AssetModelHierarchy": {
      "description": "Contains information about an asset model hierarchy.",
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "id",
            "childAssetModelId"
          ]
        },
        {
          "required": [
            "id",
            "childAssetModelExternalId"
          ]
        },
        {
          "required": [
            "externalId",
            "childAssetModelId"
          ]
        },
        {
          "required": [
            "externalId",
            "childAssetModelExternalId"
          ]
        }
      ],
      "required": [
        "name"
      ],
      "properties": {
```

```
        "id": {
          "description": "The ID of the asset model hierarchy.",
          "$ref": "#/definitions/ID"
        },
        "externalId": {
          "description": "The ExternalID of the asset model hierarchy.",
          "$ref": "#/definitions/ExternalId"
        },
        "name": {
          "description": "The name of the asset model hierarchy.",
          "$ref": "#/definitions/Name"
        },
        "childAssetModelId": {
          "description": "The ID of the asset model. All assets in this hierarchy must
 be instances of the child AssetModelId asset model.",
          "$ref": "#/definitions/ID"
        },
        "childAssetModelExternalId": {
          "description": "The ExternalID of the asset model. All assets in this
 hierarchy must be instances of the child AssetModelId asset model.",
          "$ref": "#/definitions/ExternalId"
        }
      }
    },
    "AssetModel": {
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "assetModelId"
          ]
        },
        {
          "required": [
            "assetModelExternalId"
          ]
        }
      ],
      "required": [
        "assetModelName"
      ],
      "properties": {
        "assetModelId": {
```

```
          "description": "The ID of the asset model.",
          "$ref": "#/definitions/ID"
        },
        "assetModelExternalId": {
          "description": "The ID of the asset model.",
          "$ref": "#/definitions/ExternalId"
        },
        "assetModelName": {
          "description": "A unique, friendly name for the asset model.",
          "$ref": "#/definitions/Name"
        },
        "assetModelDescription": {
          "description": "A description for the asset model.",
          "$ref": "#/definitions/Description"
        },
        "assetModelType": {
          "description": "The type of the asset model.",
          "$ref": "#/definitions/AssetModelType"
        },
        "assetModelProperties": {
          "description": "The property definitions of the asset model.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetModelProperty"
          }
        },
        "assetModelCompositeModels": {
          "description": "The composite asset models that are part of this asset model.
 Composite asset models are asset models that contain specific properties.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetModelCompositeModel"
          }
        },
        "assetModelHierarchies": {
          "description": "The hierarchy definitions of the asset model. Each hierarchy
 specifies an asset model whose assets can be children of any other assets created from
 this asset model.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetModelHierarchy"
          }
        },
        "tags": {
```

```
          "description": "A list of key-value pairs that contain metadata for the asset
 model.",
          "type": "array",
          "items": {
            "$ref": "#/definitions/Tag"
          }
        }
      }
    },
    "Asset": {
      "type": "object",
      "additionalProperties": false,
      "anyOf": [
        {
          "required": [
            "assetId",
            "assetModelId"
          ]
        },
        {
          "required": [
            "assetExternalId",
            "assetModelId"
          ]
        },
        {
          "required": [
            "assetId",
            "assetModelExternalId"
          ]
        },
        {
          "required": [
            "assetExternalId",
            "assetModelExternalId"
          ]
        }
      ],
      "required": [
        "assetName"
      ],
      "properties": {
        "assetId": {
          "description": "The ID of the asset",
```

```
          "$ref": "#/definitions/ID"
        },
        "assetExternalId": {
          "description": "The external ID of the asset",
          "$ref": "#/definitions/ExternalId"
        },
        "assetModelId": {
          "description": "The ID of the asset model from which to create the asset.",
          "$ref": "#/definitions/ID"
        },
        "assetModelExternalId": {
          "description": "The ExternalID of the asset model from which to create the 
asset.",
          "$ref": "#/definitions/ExternalId"
        },
        "assetName": {
          "description": "A unique, friendly name for the asset.",
          "$ref": "#/definitions/Name"
        },
        "assetDescription": {
          "description": "A description for the asset",
          "$ref": "#/definitions/Description"
        },
        "assetProperties": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetProperty"
          }
        },
        "assetHierarchies": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/AssetHierarchy"
          }
        },
        "tags": {
          "description": "A list of key-value pairs that contain metadata for the 
asset.",
          "type": "array",
          "uniqueItems": false,
          "items": {
            "$ref": "#/definitions/Tag"
          }
        }
```

```
      }
    }
  },
  "additionalProperties": false,
  "properties": {
    "assetModels": {
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$ref": "#/definitions/AssetModel"
      }
    },
    "assets": {
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$ref": "#/definitions/Asset"
      }
    }
  }
}
```

Monitor data with alarms in AWS IoT SiteWise

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

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**

  > **Note**
  >
  > End of support notice: On May 20, 2026, AWS will end support for AWS IoT Events. After May 20, 2026, you will no longer be able to access the AWS IoT Events console or AWS IoT Events resources. For more information, see AWS IoT Events end of support.

  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 Define alarms on asset models in AWS IoT SiteWise.

The alarm state data object contains the following information:

`stateName`

The state of the alarm. For more information, see Alarm states.

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`

Define alarms on asset models in AWS IoT SiteWise

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.

> **Note**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS IoT Events. After May 20, 2026, you will no longer be able to access the AWS IoT Events console or AWS IoT Events resources. For more information, see AWS IoT Events end of support.

- `EXTERNAL` – An external alarm. You ingest the state of the alarm as a measurement.

Property name: `AWS/ALARM_TYPE`

Property type: attribute

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.

Property name: `AWS/ALARM_STATE`

Property type: measurement

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

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.

```
{
  "assetModelName": "Boiler",
  "assetModelDescription": "A boiler that alarms when its temperature exceeds its
 limit.",
  "assetModelProperties": [
    {
      "name": "Temperature",
      "dataType": "DOUBLE",
      "unit": "Celsius",
      "type": {
        "measurement": {}
      }
    },
    {
```

```
        "name": "High Temperature",
        "dataType": "DOUBLE",
        "unit": "Celsius",
        "type": {
          "attribute": {
            "defaultValue": "105.0"
          }
        }
      }
    ],
    "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": {}
            }
          },
          {
            "name": "AWS/ALARM_SOURCE",
            "dataType": "STRING",
            "type": {
              "attribute": {}
            }
          }
        ]
      }
    ]
  }
```

Topics

Requirements for alarm notifications in AWS IoT SiteWise

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.

> **Note**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS IoT Events. After May 20, 2026, you will no longer be able to access the AWS IoT Events console or AWS IoT Events resources. For more information, see AWS IoT Events end of support.

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*.

Define AWS IoT Events alarms for AWS IoT SiteWise

> **Note**
>
> End of support notice: On May 20, 2026, AWS will end support for AWS IoT Events. After May 20, 2026, you will no longer be able to access the AWS IoT Events console or AWS IoT Events resources. For more information, see AWS IoT Events end of support.

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.

> **Note**
>
> The alarm notifications feature isn't available in the China (Beijing) Region.

Topics

- Define an AWS IoT Events alarm (AWS IoT SiteWise console)
- Define an AWS IoT Events alarm (AWS IoT Events console)
- Define an AWS IoT Events alarm (AWS CLI)

Define an AWS IoT Events alarm (AWS IoT SiteWise 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 Create asset models in AWS IoT SiteWise.

> **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](https://console.aws.amazon.com/iotsitewise/).
2. In the navigation pane, choose **Models**.
3. Choose the asset model for which to define an alarm.
4. Choose the **Alarm** tab.
5. Choose **Add alarm**.
6. In the **Alarm type options** section, choose **AWS IoT Events alarm**.
7. In the **Alarm details** section, do the following:
    a. Enter a name for your alarm.
    b. (Optional) Enter a description for your alarm.
8. In the **Threshold definitions** section, 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. For **Value**, 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.

9. (Optional) In the **Notification settings - *optional*** section, do the following:

a. Choose **Active**.

> **Note**
>
> If you choose **Inactive**, you and your team won't receive any alarm notifications.

b. For **Recipient**, choose the recipient.

> **Important**
>
> You can send alarm notifications to AWS IAM Identity Center users. To use this feature, you must enable IAM Identity Center. You can only enable IAM Identity Center in one AWS Region at a time. This means that you can define alarm notifications only in the Region where you enable IAM Identity Center. For more information, see Getting started in the *AWS IAM Identity Center User Guide*.

c. For **Protocol**, choose from the following options:

- **Email & text –** The alarm notifies IAM Identity Center users with an SMS message and an email message.
- **Email** – The alarm notifies IAM Identity Center users with an email message.
- **Text –** The alarm notifies IAM Identity Center 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 an email address identity, in the *Amazon Simple Email Service Developer Guide*.

10. In the **Default asset state** section, you can set the default state for alarms created from this asset model.

> **Note**
>
> You activate or deactivate this alarm for assets that you create from this asset model in a later step.

11. In the **Advanced settings** section, 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** section, 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.

12. (Optional) In the **Additional notification settings** section, do the following:

    a. For **Recipient attribute**, you define an attribute whose value specifies the recipient of the notification. You can choose IAM Identity Center users as recipients.

       You can create an attribute or use an existing attribute on the asset model.

       - If you choose **Create a new recipient attribute**, specify the **Recipient attribute name** and **Recipient default value - optional** for the attribute.
       - If you choose **Use an existing recipient attribute**, choose the attribute in **Recipient 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 a new custom message attribute**, specify the **Custom message attribute name** and **Custom message default value - *optional*** for the attribute.
       - If you choose **Use an existing custom message attribute**, choose the attribute in **Custom message 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 **Manage your Lambda function**, do one of the following:

       - To have AWS IoT SiteWise create a new Lambda function, choose **Create a new lambda from an AWS managed template**.
       - To use an existing Lambda function, choose **Use an existing lambda** and choose the name of the function.

       For more information, see Managing alarm notifications in the *AWS IoT Events Developer Guide*.

13. (Optional) In the **Set state action** section, do the following:

a. Choose **Edit action**.

b. Under **Add alarm state actions**, add actions. and the choose **Save**.

   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*.

14. (Optional) Under **Manage alarm model in SiteWise Monitor - *optional***, choose **Active** or **Inactive**.

    Use this option so that you can update the alarm model in SiteWise Monitorss. This option is enabled by default.

15. Under **Acknowledge flow**, choose **Active** or **Inactive**. For more information about the acknowledge flow, see Alarm states.

16. 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 requests 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.

Define an AWS IoT Events alarm (AWS IoT Events console)

You can use the AWS IoT Events 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 Create asset models in AWS IoT SiteWise.

> **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 Events console.
2. In the navigation pane, choose **Alarm models**.
3. Choose **Create alarm model**.
4. Enter a name for your alarm.
5. (Optional) Enter a description for your alarm.
6. In the **Alarm target** section, do the following:
   a. For **Target options**, choose **AWS IoT SiteWise asset property**.
   b. Choose the asset model for which you want to add the alarm.
7. In the **Threshold definitions** section, 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. For **Value**, 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.

8. (Optional) In the **Notification settings - *optional*** section, do the following:

    a. For **Protocol**, choose from the following options:

        - **Email & text** – The alarm notifies IAM Identity Center users with an SMS message and an email message.
        - **Email** – The alarm notifies IAM Identity Center users with an email message.
        - **Text** – The alarm notifies IAM Identity Center users with an SMS message.

    b. 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*.

    c. Choose the attribute in **Recipient attribute - *optional***. The alarm uses the default value of the attribute that you choose.

    d. Choose the attribute in **Custom message attribute - *optional***. The alarm uses the default value of the attribute that you choose.

9. In the **Instance** section, specify the **Default state** for this alarm. You can activate or deactivate this alarm for all assets that you create from this asset model in a later step.

10. In the **Advanced settings** settings, 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.

a. In the **Acknowledge flow** section, choose **Enabled** or **Disabled**. For more information about the acknowledge flow, see Alarm states.

b. In the **Permissions** section, do the following:

   i. 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.

   ii. For the **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.

c. (Optional) In the **Additional notification settings** pane, do the following:

   - For **Manage your Lambda function**, do one of the following:
     - To have AWS IoT Events create a new Lambda function, choose **Create a new Lambda function**.
     - To use an existing Lambda function, choose **Use an existing Lambda function** and choose the name of the function.

   For more information, see Managing alarm notifications in the *AWS IoT Events Developer Guide*.

d. (Optional) In the **Set state action - *optional*** section, do the following:

   - Under **Alarm state actions**, add actions. and the choose **Save**.

     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*.

11. Choose **Create**.

> **Note**
>
> The AWS IoT Events 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 requests 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.

Define an AWS IoT Events alarm (AWS 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: Define an alarm on an asset model
- Step 2: Define an AWS IoT Events alarm model
- Step 3: Enable data flow between AWS IoT SiteWise and AWS IoT Events

Step 1: Define 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.
   - For more information about how to create an asset model, see Create an asset model (AWS CLI).
   - For more information about how to update an existing asset model, see Update an asset model, component model, or interface (AWS CLI).

> **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.

```
{
  ...
  "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`.

```
{
  ...
  "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 IAM Identity Center 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 IAM Identity Center users. To use this feature, you must enable IAM Identity Center. You can only enable IAM Identity Center in one

AWS Region at a time. This means that you can define alarm notifications only in the Region where you enable IAM Identity Center. For more information, see Getting started in the *AWS IAM Identity Center User Guide*.

Do the following:

a. Add an attribute that specifies the ID of your IAM Identity Center identity store. You can use the IAM Identity Center ListInstances API operation to list your identity stores. This operation works only in the Region where you enable IAM Identity Center.

```
aws sso-admin list-instances
```

Then, specify the identity store ID (for example, `d-123EXAMPLE`) as the default value for the attribute.

```
{
  ...
  "assetModelProperties": [
    ...
    {
      "name": "identityStoreId",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "d-123EXAMPLE"
        }
      }
    }
  ]
}
```

b. Add an attribute that specifies the ID of the IAM Identity Center user who receives notifications. To define a default notification recipient, add an IAM Identity Center user ID as the default value. Do one of the following to get an IAM Identity Center user ID:

   i. You can use the IAM Identity Center 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.

```
aws identitystore list-users \
  --identity-store-id d-123EXAMPLE \
  --filters AttributePath=UserName,AttributeValue=Name
```

ii. Use the IAM Identity Center 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.

```
{
  ...
  "assetModelProperties": [
    ...
    {
      "name": "userId",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "123EXAMPLE-a1b2c3d4-5678-90ab-cdef-33333EXAMPLE"
        }
      }
    }
  ]
}
```

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 Request a sender ID in AWS End User Messaging SMS in the *AWS End User Messaging SMS User 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..

```
{
  ...
  "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.

```
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.

```
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.

```
{
  "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": "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",
      "properties": [
        {
          "name": "AWS/ALARM_TYPE",
          "dataType": "STRING",
```

```
          "type": {
            "attribute": {
              "defaultValue": "IOT_EVENTS"
            }
          }
        },
        {
          "name": "AWS/ALARM_STATE",
          "dataType": "STRUCT",
          "dataTypeSpec": "AWS/ALARM_STATE",
          "type": {
            "measurement": {}
          }
        }
      ]
    }
  ]
}
```

Step 2: Define 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 IAM Identity Center identity store ID attribute
   - (Optional) The IAM Identity Center 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.

> **Important**
>
> The alarm model must have the same name as the alarm composite model that you defined on your asset model earlier.
> Alarm model names can contain only alphanumeric characters.

```
{
  "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 *`alarmPropertyId`* with the ID of the property that the alarm monitors.
- Replace *`thresholdAttributeId`* 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`

```
{
```

```
  "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"
    }
  }
}
```

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.

```
{
  "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'"
        }
      }
    ]
  }
}
```

f. (Optional) Configure alarm notification settings. The alarm notification action uses a Lambda function in your account to send alarm notifications. For more information, see [Requirements for alarm notifications in AWS IoT SiteWise](). In the alarm notification settings, you can configure SMS and email notifications to send to IAM Identity Center users. Do the following:

   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.

```
{
  "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 IAM Identity Center user when the alarm changes state.

- Replace *`identityStoreIdAttributeId`* with the ID of the attribute that contains the ID of the IAM Identity Center identity store.
- Replace *`userIdAttributeId`* with the ID of the attribute that contains the ID of the IAM Identity Center 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.

```
{
  "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.va
                "userId":
"$sitewise.assetModel.`assetModelId`.`userIdAttributeId`.propertyValue.value"
              }
            }
          ],
          "senderId":
"$sitewise.assetModel.`assetModelId`.`senderIdAttributeId`.propertyValue.value",
```

```
              "additionalMessage":
"$sitewise.assetModel.`assetModelId`.`additionalMessageAttributeId`.propertyValue.
            }
          ]
        }
      ]
    }
  }
```

iii. (Optional) Configure the email notifications (`emailConfigurations`) to send to an IAM Identity Center user when the alarm changes state.

- Replace *identityStoreIdAttributeId* with the ID of the IAM Identity Center identity store ID attribute property.
- Replace *userIdAttributeId* with the ID of the IAM Identity Center 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.

```
{
  "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.va
                  "userId":
"$sitewise.assetModel.`assetModelId`.`userIdAttributeId`.propertyValue.value"
                }
              }
            ],
            "senderId":
"$sitewise.assetModel.`assetModelId`.`senderIdAttributeId`.propertyValue.value",
            "additionalMessage":
"$sitewise.assetModel.`assetModelId`.`additionalMessageAttributeId`.propertyValue.
          }
        ],
        "emailConfigurations": [
          {
            "from":
"$sitewise.assetModel.`assetModelId`.`fromAddressAttributeId`.propertyValue.value"
            "recipients": {
              "to": [
                {
                  "ssoIdentity": {
                    "identityStoreId":
"$sitewise.assetModel.`assetModelId`.`identityStoreIdAttributeId`.propertyValue.va
```

```
                        "userId":
 "$sitewise.assetModel.`assetModelId`.`userIdAttributeId`.propertyValue.value"
                      }
                    }
                  ]
                },
                "content": {
                  "subject":
 "$sitewise.assetModel.`assetModelId`.`emailSubjectAttributeId`.propertyValue.value
                  "additionalMessage":
 "$sitewise.assetModel.`assetModelId`.`additionalMessageAttributeId`.propertyValue.
                }
              }
            ]
          }
        ]
      }
    }
```

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](#).

```
{
  "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`.propertyValu
         }
       ],
       "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.valu
              }
            }
          ]
        }
      ]
  },
  "alarmCapabilities": {
    "initializationConfiguration": {
      "disabledOnInitialization": false
    },
    "acknowledgeFlow": {
      "enabled": true
    }
  }
}
```

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 `sso-directory: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.

```
{
  "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`.propertyValu
          }
        ],
```

```
          "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.valu
              }
            }
          ]
        }
      ]
    },
    "alarmCapabilities": {
      "initializationConfiguration": {
        "disabledOnInitialization": false
      },
      "acknowledgeFlow": {
        "enabled": false
      }
    },
    "roleArn": "arn:aws:iam::123456789012:role/MyIoTEventsAlarmRole"
}
```

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: Enable 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.

  ```
  {
    ...
    "assetModelCompositeModels": [
  ```

```
    ...
    {
      "name": "BoilerTemperatureHighAlarm",
      "type": "AWS/ALARM",
      "properties": [
        {
          "id": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
          "name": "AWS/ALARM_TYPE",
          "dataType": "STRING",
          "type": {
            "attribute": {
              "defaultValue": "IOT_EVENTS"
            }
          }
        },
        {
          "id": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
          "name": "AWS/ALARM_STATE",
          "dataType": "STRUCT",
          "dataTypeSpec": "AWS/ALARM_STATE",
          "type": {
            "measurement": {}
          }
        },
        {
          "name": "AWS/ALARM_SOURCE",
          "dataType": "STRING",
          "type": {
            "attribute": {
              "defaultValue": "alarmModelArn"
            }
          }
        }
      ]
    }
  ]
}
```

e. Run the following command to update the asset model with the definition stored in the `update-asset-model-payload.json` file. 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://update-asset-model-payload.json
```

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 IAM Identity Center recipient for each asset. For more information, see Configure alarms on assets in AWS IoT SiteWise.

Define external alarms in AWS IoT SiteWise

External alarms contain the state of an alarm that you detect outside of AWS IoT SiteWise.

Define 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 Create asset models in AWS IoT SiteWise.

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**.

Define 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:

- Create an asset model (AWS CLI)
- Update an asset model, component model, or interface (AWS CLI)

After you define the external alarm, you can ingest alarm state to assets based on the asset model. For more information, see Ingest an external alarm state in AWS IoT SiteWise.

Configure alarms on assets in AWS IoT SiteWise

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 on external alarms.

Topics

Configure 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**.

Configure a threshold value (AWS 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.

You must know your asset's `assetId` and property's `propertyId` to complete this procedure. You can also use the external ID. If you created an asset and don't know its `assetId`, use the ListAssets API to list all the assets for a specific model. 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 (AWS 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.

```
{
  "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`
    - `doubleValue`
    - `integerValue`
    - `stringValue`
    - `nullValue`
  - `timestamp` – A structure that contains the current Unix epoch time in seconds, `timeInSeconds`. AWS IoT SiteWise rejects any data points with timestamps that existed longer 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 Ingest data with AWS IoT SiteWise APIs.

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
```

Configure notification settings in AWS IoT SiteWise

You can configure alarm notification settings using either the AWS IoT SiteWise console or the AWS Command Line Interface (AWS CLI).

Configure 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 for which you want to update the alarm settings.
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**.

Configure notification settings (CLI)

You can use the AWS Command Line Interface (AWS CLI) to update the value of the attribute that specifies the notification settings for an alarm.

You must know your asset's `assetId` and property's `propertyId` to complete this procedure. You can also use the external ID. If you created an asset and don't know its `assetId`, use the ListAssets API to list all the assets for a specific model. 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 (AWS 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.

```
{
  "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`
        - `doubleValue`
        - `integerValue`
        - `stringValue`
        - `nullValue`
    - `timestamp` – A structure that contains the current Unix epoch time in seconds, `timeInSeconds`. AWS IoT SiteWise rejects any data points with timestamps that existed longer 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 Ingest data with AWS IoT SiteWise APIs.

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
```

Respond to alarms in AWS IoT SiteWise

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

Respond 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)

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](https://console.aws.amazon.com/iotsitewise/).
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](https://console.aws.amazon.com/iotsitewise/).
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](https://console.aws.amazon.com/iotsitewise/).
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](https://console.aws.amazon.com/iotsitewise/).
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**.

Respond 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*.

Ingest an external alarm state in AWS IoT SiteWise

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.

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 Define data streams from equipment (measurements) and Ingest data to AWS IoT SiteWise.

Map external alarm state streams in AWS IoT SiteWise

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 Manage data streams for AWS IoT SiteWise.

Topics

- Map external alarm state streams (console)
- Map external alarm state streams (AWS CLI)

Map external alarm state streams (console)

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 Manage data streams for AWS IoT SiteWise.

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 **Edit**.
5. Scroll to **Alarms** and expand the section.
6. Under **External Alarms**, enter the alias in **Property alias – *optional***.
7. Choose **Save**.

Map external alarm state streams (AWS CLI)

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 Manage data streams for AWS IoT SiteWise.

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. You can also use the external ID. If you created an asset and don't know its `assetId`, use the ListAssets API to list all the assets for a specific model. 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 Update an asset property alias.

Ingest alarm state data in AWS IoT SiteWise

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 Map external alarm state streams in AWS IoT SiteWise.

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 Example BatchPutAssetPropertyValue payload for alarm state data

```
{
    "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 Ingest data with AWS IoT SiteWise APIs.

For more information about other ways to ingest data, see Ingest data to AWS IoT SiteWise.

AWS IoT SiteWise Assistant

The AWS IoT SiteWise Assistant is a generative AI-powered assistant. It allows users like plant managers, quality engineers, and maintenance technicians to gain insights, solve problems, and take actions directly from their operational and enterprise data.

The AWS IoT SiteWise Assistant consolidates information from AWS IoT data, asset models, manuals and documentation into understandable summaries of critical events. It also enables interactive deep dive question and answer sessions for easy diagnosis, root cause explorations and guided recommendations.

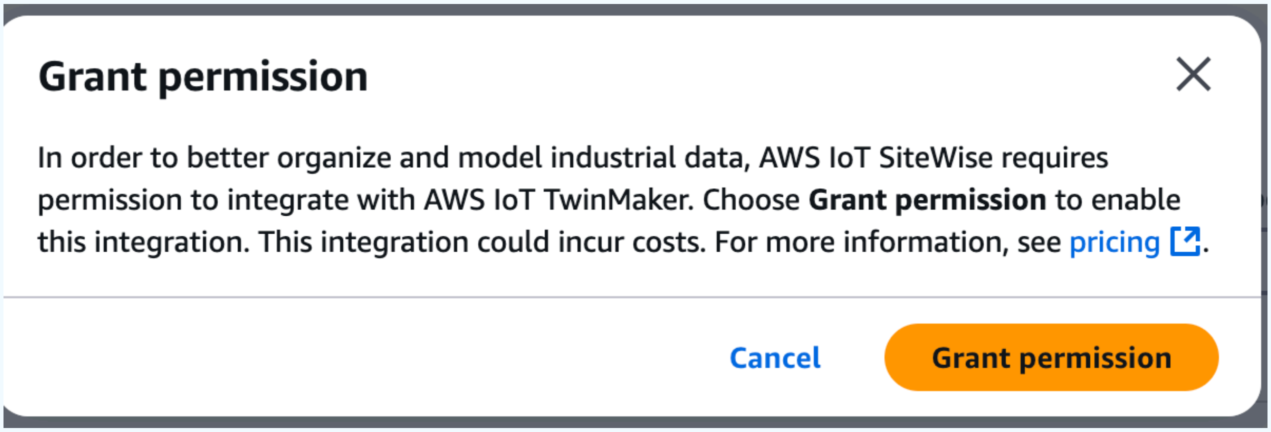
Topics

Configure the AWS IoT SiteWise Assistant

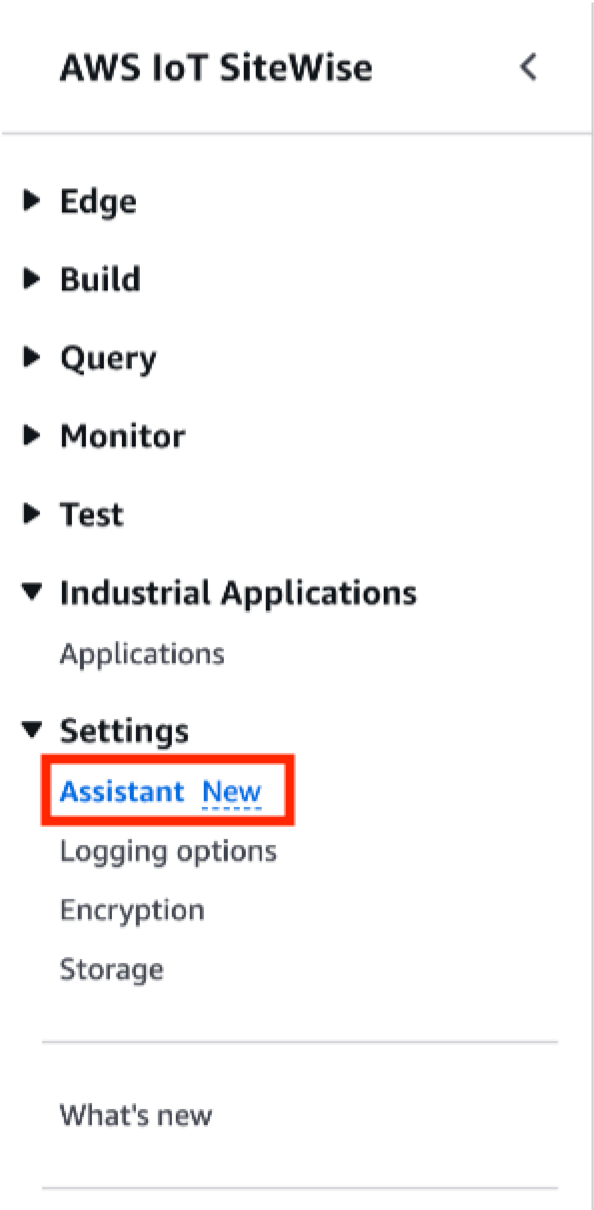
AWS IoT SiteWise Assistant configuration

1. Sign in to the AWS IoT SiteWise console.

> **Note**
>
> Grant permissions to enable integration with AWS IoT TwinMaker service. This is required for the AWS IoT SiteWise Assistant, and the dashboard to execute SQL queries in AWS IoT SiteWise resources. See Integrate AWS IoT SiteWise and AWS IoT TwinMaker.



2. Choose **Assistant** from the left navigation panel.



Create a dataset

> **Note**
>
> The AWS IoT SiteWise Assistant must use a dataset with an Amazon Kendra index for enterprise level knowledge and guidance. If you do not have a Amazon Kendra index, see Creating an index to create one. Adding a dataset improves the quality of the Assistant's response, and minimizes hallucinations.

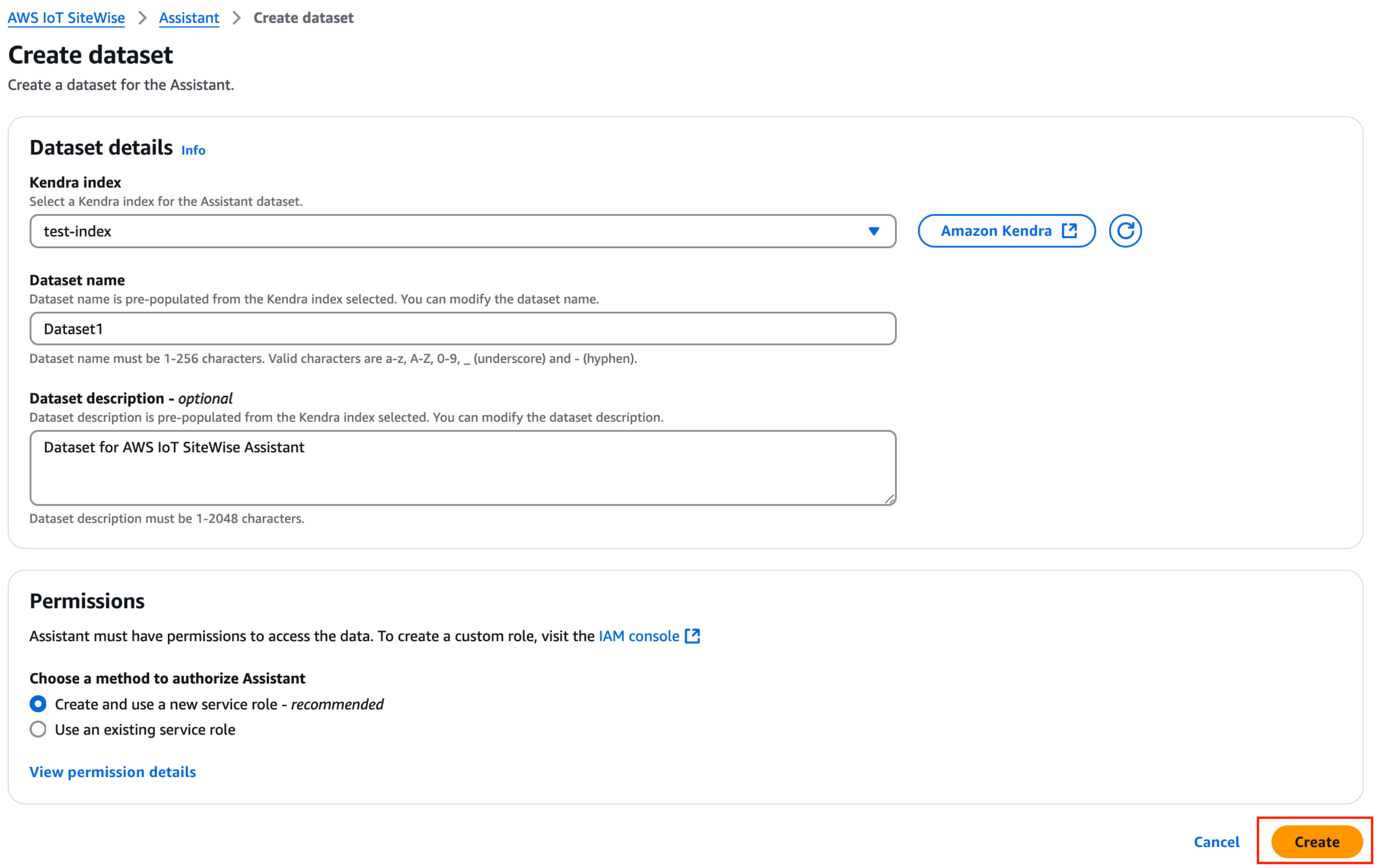
Console

Create a dataset in the AWS IoT SiteWise console

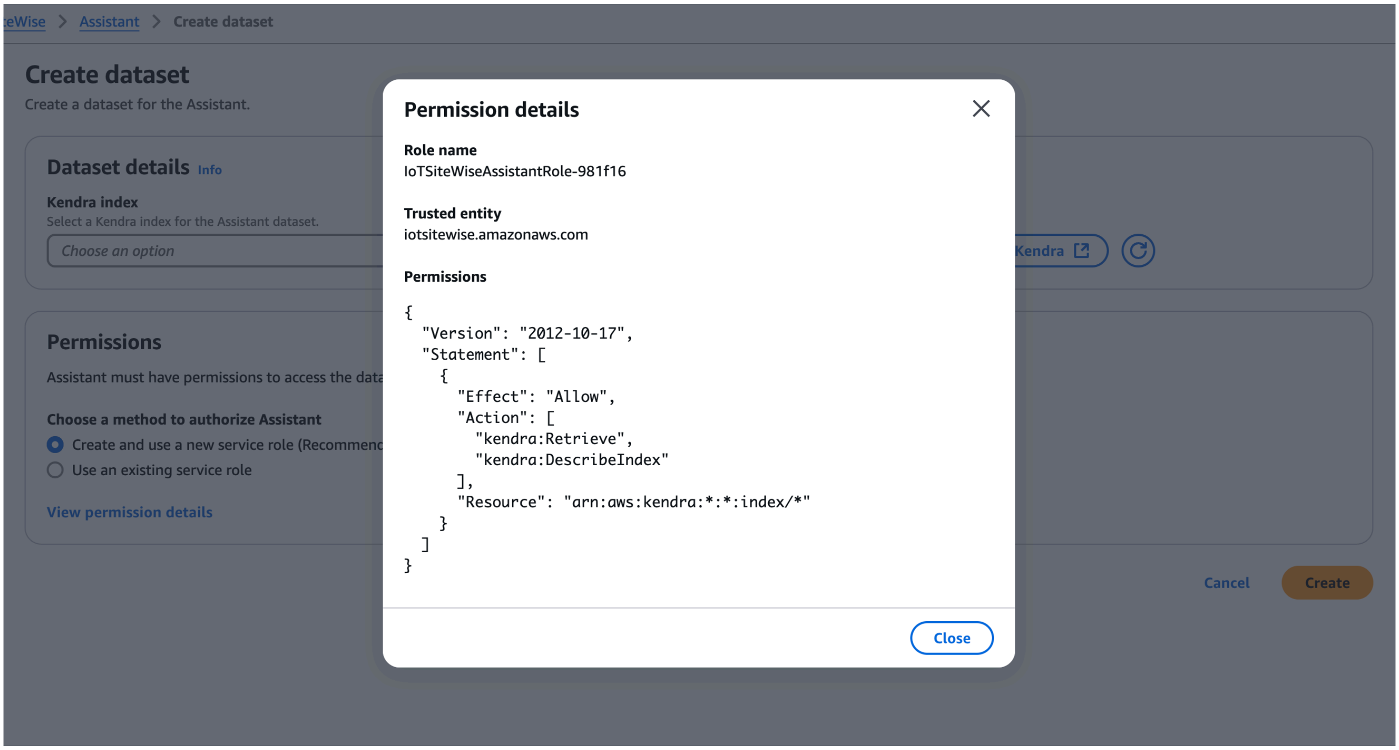
1. Datasets are displayed in the **Datasets** section of the **AWS IoT SiteWise Assistant** page.
2. If no datasets exist, choose **Create dataset**.
3. In the **Dataset details** page, choose a Kendra index from the drop down menu to associate with the dataset.
4. The dataset name is populated by the Kendra index selected in Step 3. Edit the name if needed.
5. (Optional) The dataset description is populated by the Kendra index selected in Step 3. Edit the description if needed.
6. In the **Permissions** section, choose from below:

    a. Choose **Create and use a new service role**. By default, AWS IoT SiteWise automatically creates a service role. This role allows the AWS IoT SiteWise Assistant to access your Kendra indexes.

    b. Choose **Use an existing service role**, and then choose the target role.
7. Choose **Create**.



The service role created by AWS IoT SiteWise for the user, if the user chose to **Create and use a new service role**.



AWS CLI

Create a dataset in AWS CLI

1. Create an IAM role used to create a dataset. Use the following permissions policy:

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kendra:Retrieve"
            ],
            "Resource": "arn:aws:kendra:*:*:index/*"
        }
    ]
}
```

Use the following trust relationship:

JSON

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iotsitewise.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Create a file **create-dataset.json** with the template provided in the example. Populate `datasetId`, `kendra knowledgeBaseArn` and `roleArn` to connect with this dataset.

```
{
    "datasetId": "<UUID>",
    "datasetName": "DatasetForAssistant",
    "datasetSource": {
       "sourceType": "KENDRA",
       "sourceFormat": "KNOWLEDGE_BASE",
       "sourceDetail": {
          "kendra": {
            "knowledgeBaseArn": "arn:aws:kendra::%s:index/index",
            "roleArn": "arn:aws:iam::%s:role/role"
          }
       }
    }
}
```

3. Create the dataset with the following command:

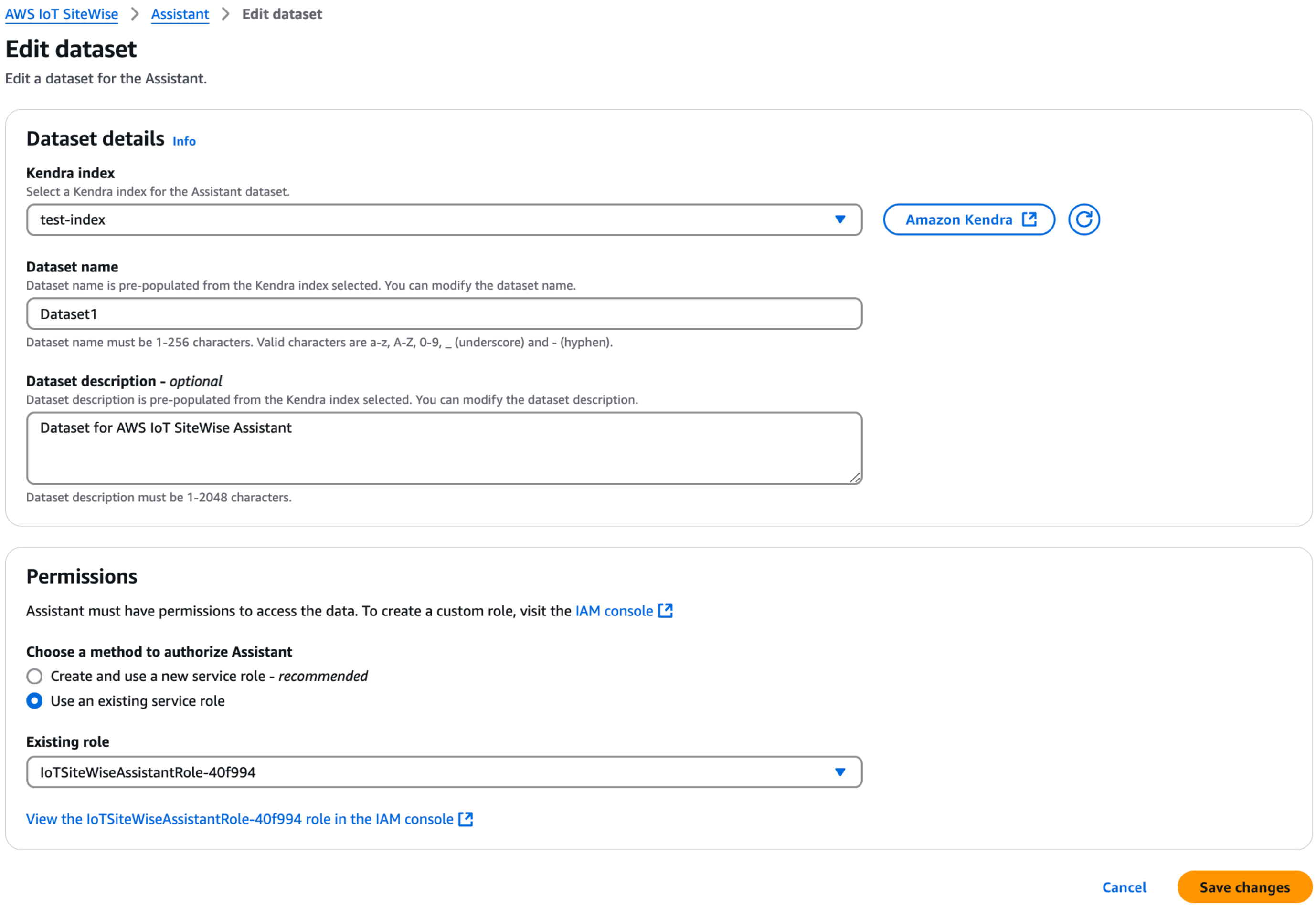
```
aws iotsitewise create-dataset --cli-input-json file://create-dataset.json --
region us-east-1
```

Edit a dataset

Console

Edit a dataset

1. Datasets are displayed in the **Datasets** section of the **Assistant** page. Choose a dataset to edit. Choose **Edit** to start editing.
2. In the **Dataset details** page, choose a Kendra index from the drop down menu to associate with the dataset.
3. The dataset name is populated by the Kendra index selected in Step 2. Edit the name if needed.
4. (Optional) The dataset description is populated by the Kendra index selected in Step 2. Edit the description if needed.
5. In the **Permissions** section, choose from below:
    a. Choose **Create and use a new service role**. By default, AWS IoT SiteWise automatically creates a service role. This role allows the AWS IoT SiteWise Assistant to access your Kendra indexes.
    b. Choose **Use an existing service role**, and then choose the target role.
6. Choose **Save changes** to save your selection.



AWS CLI

Edit a dataset in AWS CLI

1. Create a file **update-dataset.json** with the template provided in the example. Populate `datasetId`, `kendra knowledgeBaseArn` and `roleArn` to connect with this dataset.

```
{
    "datasetId": "<UUID>",
    "datasetName": "DatasetForAssistant",
    "datasetSource": {
       "sourceType": "KENDRA",
       "sourceFormat": "KNOWLEDGE_BASE",
       "sourceDetail": {
          "kendra": {
            "knowledgeBaseArn": "arn:aws:kendra::%s:index/index",
            "roleArn": "arn:aws:iam::%s:role/role"
```

```
            }
        }
    }
}
```

2. Update the dataset with the following command:

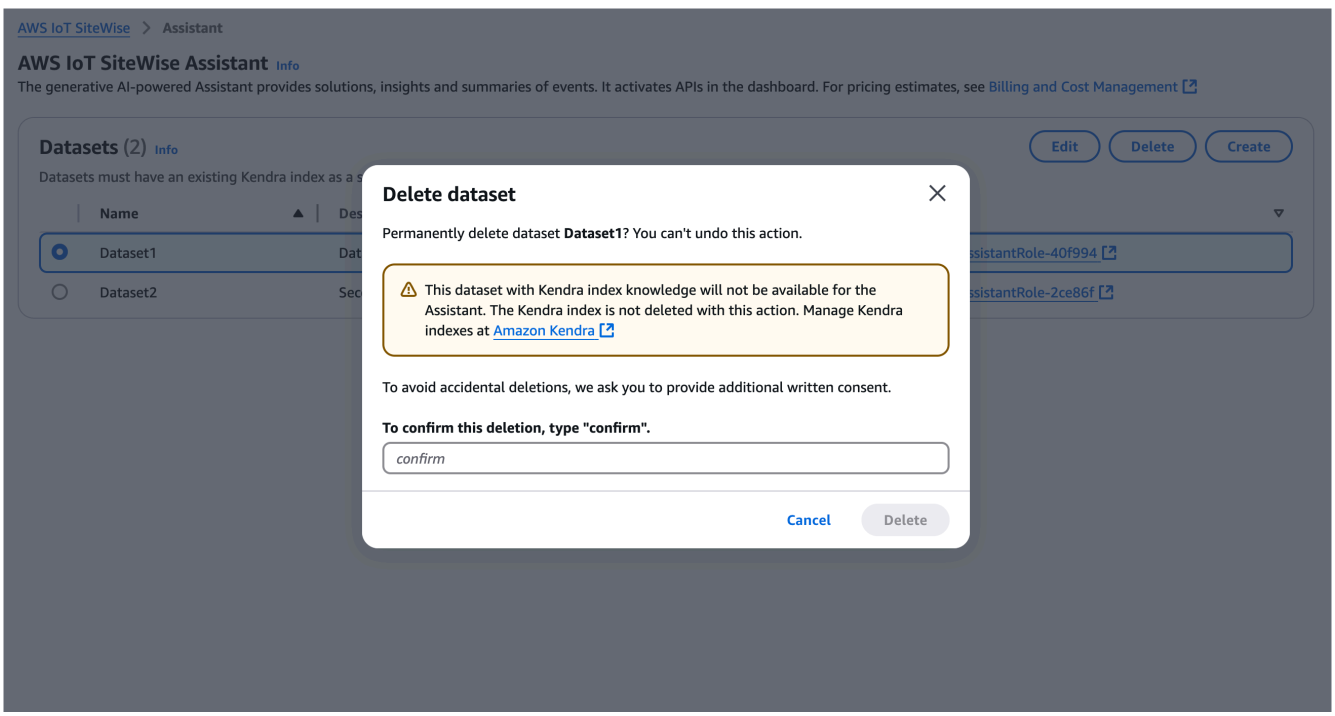
```
aws iotsitewise update-dataset --cli-input-json file://update-dataset.json --
region us-east-1
```

Delete a dataset

Console

Delete a dataset

1. Datasets are displayed in the **Datasets** section of the **Assistant** page. Choose a dataset. Choose **Delete**.
2. Type **confirm** in the popup to confirm the delete.



3. Choose **Delete**.

AWS CLI

Delete a dataset

- Delete the dataset with `datasetId`.

```
aws iotsitewise delete-dataset --region us-east-1 --dataset-id <UUID>
```

AWS IoT SiteWise Assistant questions

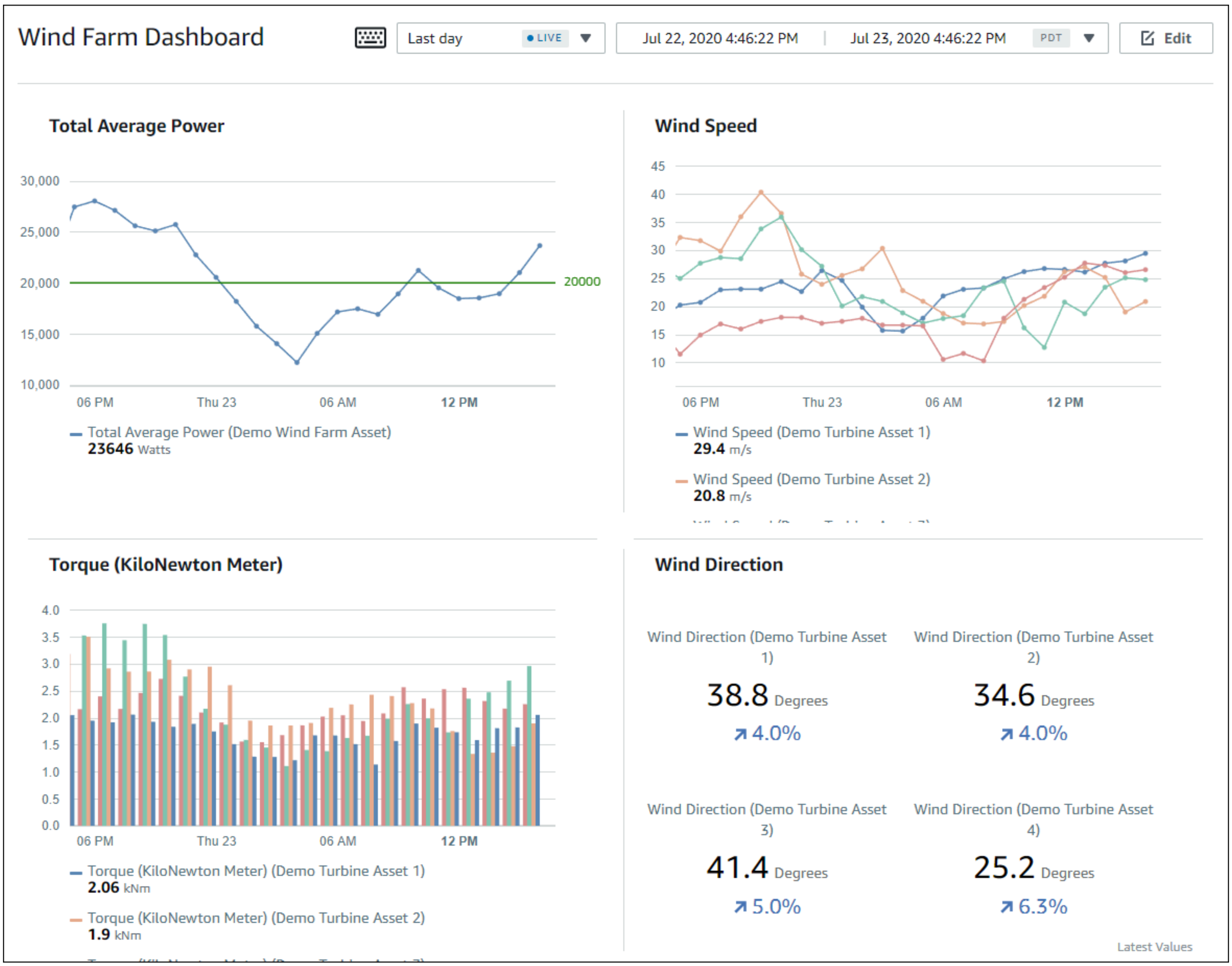
See Sample questions to ask AWS IoT SiteWise Assistant to learn more about querying AWS IoT SiteWise Assistant.

Monitor 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 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

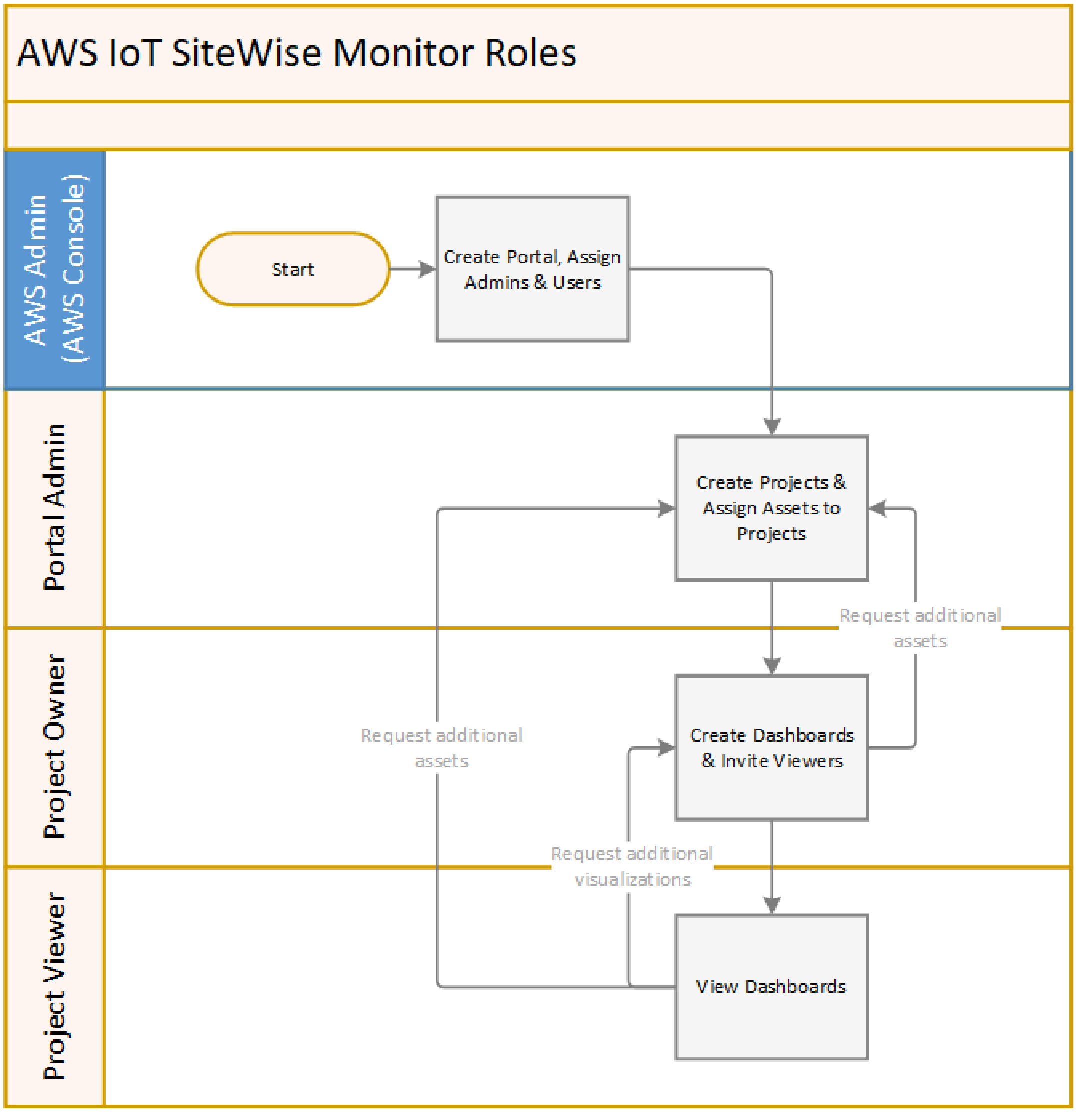
Each SiteWise Monitor project has owners. Project owners create visualizations in the form of dashboards to represent operational data in a consistent manner. When dashboards are ready to share, the project owner can invite viewers to the project. Project owners can also assign other owners to the project. 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 IAM Identity Center or IAM. Your data users can sign in to SiteWise Monitor from a desktop or mobile browser using their IAM Identity Center or IAM credentials.

SAML federation

IAM Identity Center 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 IAM Identity Center and IAM to use SAML-based federation for access to your SiteWise Monitor portals.

IAM Identity Center

Your portal administrators and users can sign in to the AWS access portal with their corporate usernames and passwords. They can then navigate to their assigned SiteWise Monitor portals. IAM Identity Center uses certificates to set up a SAML trust relationship between your identity provider and AWS. For more information, SCIM profile and SAML 2.0 implementation in the *AWS IAM Identity Center 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*.

> **Note**
>
> When adding users or administrators to the portal, avoid creating IAM policies that restrict user permissions, such as limited IP. Any attached policies with restricted permissions will not be able to connect to the AWS IoT SiteWise portal.

SiteWise Monitor concepts

To use SiteWise Monitor, you should be familiar with the following concepts:

Portal

An AWS IoT 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 customize time range to look at specific data.

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.

Get started with AWS IoT SiteWise Monitor (Classic)

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.

> **Note**
>
> When adding users or administrators to the portal, avoid creating AWS Identity and Access Management (IAM) policies that restrict user permissions, such as limited IP. Any attached policies with restricted permissions will not be able to connect to the AWS IoT SiteWise portal.

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 Visualize and share wind farm data in SiteWise Monitor.

Topics

- Create a portal in SiteWise Monitor

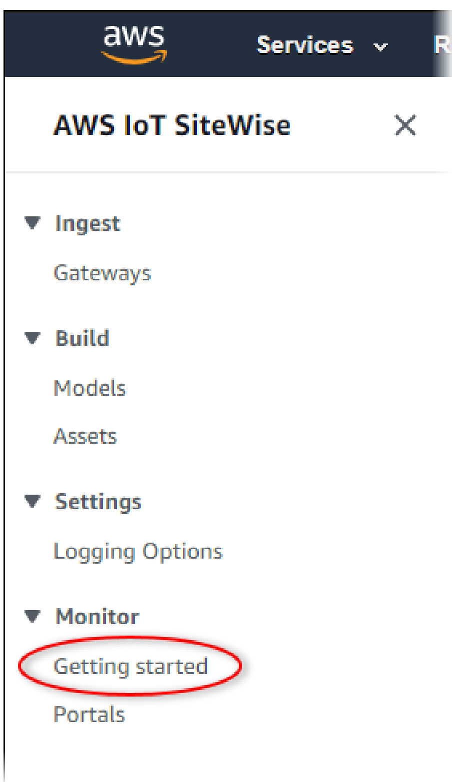
- Configure your portal in SiteWise Monitor
- Invite administrators in SiteWise Monitor
- Add portal users in SiteWise Monitor
- Create AWS IoT SiteWise dashboards (AWS CLI)
- Turn on alarms for your portals in AWS IoT SiteWise
- Enabling your AWS IoT SiteWise portal at the edge
- Administer your SiteWise Monitor portals

Create a portal in SiteWise Monitor

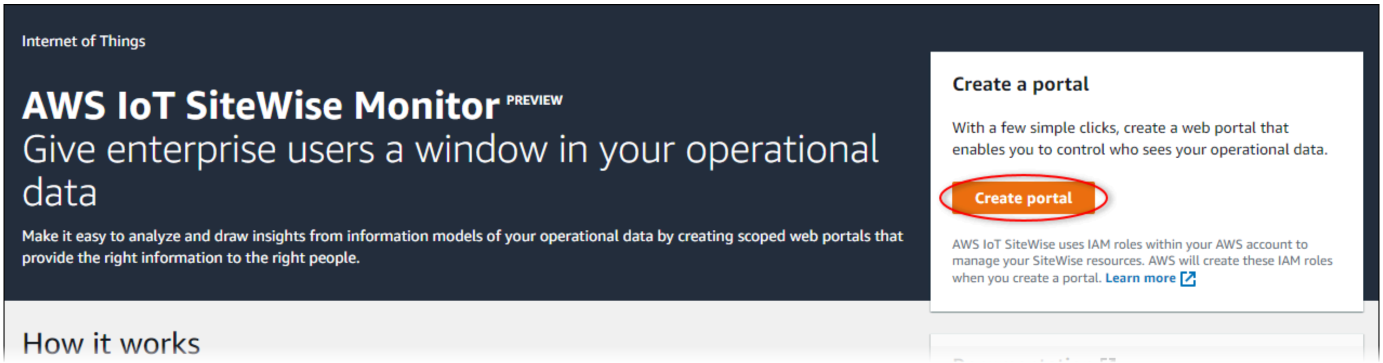
You create a SiteWise Monitor portal in the AWS IoT SiteWise console.

To create a portal

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Monitor**, **Getting started**.



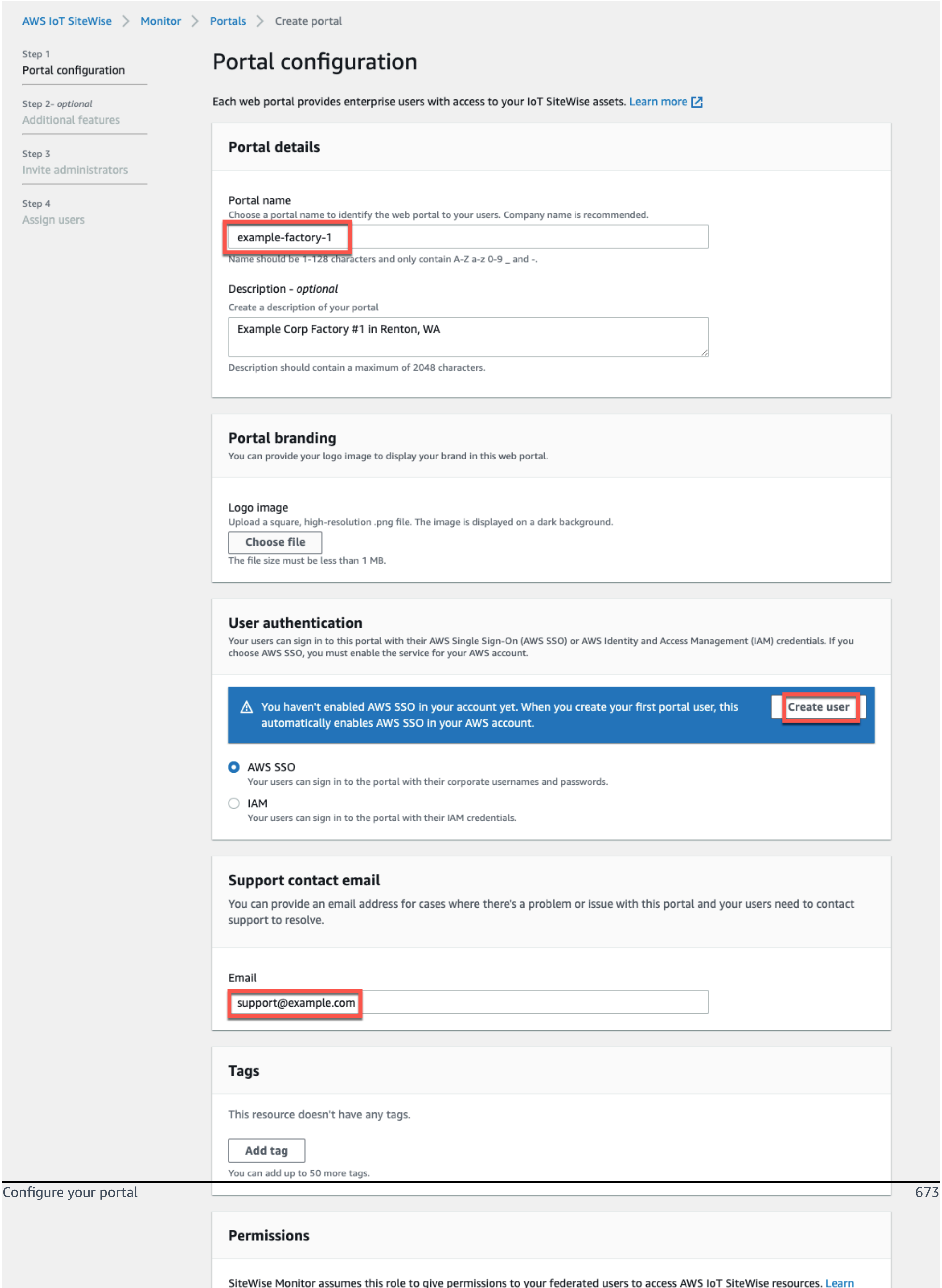
3. Choose **Create Portal**.



Next, you must provide some basic information to configure your portal.

Configure your portal in SiteWise Monitor

Your users use portals to view your data. You can customize a portal's name, description, branding, user authentication, support contact email, and permissions.


AWS IoT SiteWise > Monitor > Portals > Create portal
Step 1
Portal configuration
Step 2- optional
Additional features
Step 3
Invite administrators
Step 4
Assign users
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
Name 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

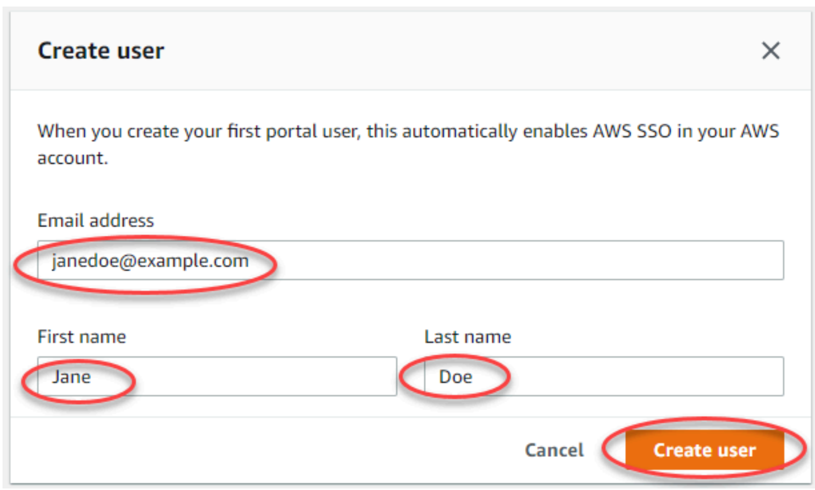
To configure a portal

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 **IAM Identity Center** if your portal users sign in to this portal with their corporate user names and passwords.

     If you haven't enabled IAM Identity Center 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**.

     Note

     - AWS automatically enables IAM Identity Center in your account when you create the first portal user.
     - You can configure IAM Identity Center in only one Region at a time. SiteWise Monitor connects to the Region that you configured for IAM Identity Center. This means that you use one Region for IAM Identity Center 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**
>
> 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 Tag your AWS IoT SiteWise resources.
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 Use service roles for AWS IoT SiteWise Monitor.
   - 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 Turn on alarms for your portals in AWS IoT SiteWise.
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 Add or remove portal administrators in AWS IoT SiteWise. If you don't want to keep this portal, delete it so it doesn't use resources. For more information, see Delete a portal in AWS IoT SiteWise.

The **Status** column can be one of the following values.

- **CREATING** - AWS IoT SiteWise is processing your request to create the portal. This process can take several minutes to complete.

- **UPDATING** - AWS IoT SiteWise is processing your request to update the portal. This process can take several minutes to complete.
- **PENDING** - AWS IoT SiteWise is waiting for the DNS record propagation to finish. This process can take several minutes to complete. You can delete the portal while the status is **PENDING**.
- **DELETING** - AWS IoT SiteWise is processing your request to delete the portal. This process can take several minutes to complete.
- **ACTIVE** - When the portal becomes active, your portal users can access it.
- **FAILED** - AWS IoT SiteWise couldn't process your request to create, update, or delete the portal. 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 CloudWatch Logs.

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.

Invite administrators in SiteWise Monitor

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:

IAM Identity Center

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 IAM Identity Center user from this page. Alternatively, you can connect an external identity provider to IAM Identity Center. For more information, see the AWS IAM Identity Center 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 IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center.

2. (Optional) Choose **Send invite to selected users**. Your email client opens, and an invitation is populated 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 adding your new IAM Identity Center 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**.

IAM

You can choose a user or role to be the portal administrator. If you want to add another user or role as a portal administrator, you can create a 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**
>
> Users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.

> **Note**
>
> If you use IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center.

You can change the list of portal administrators later. For more information, see Add or remove portal administrators in AWS IoT SiteWise.

> **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.

Add portal users in SiteWise Monitor

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:

IAM Identity Center

If you want to add a user to the **Users** list, complete the following steps.

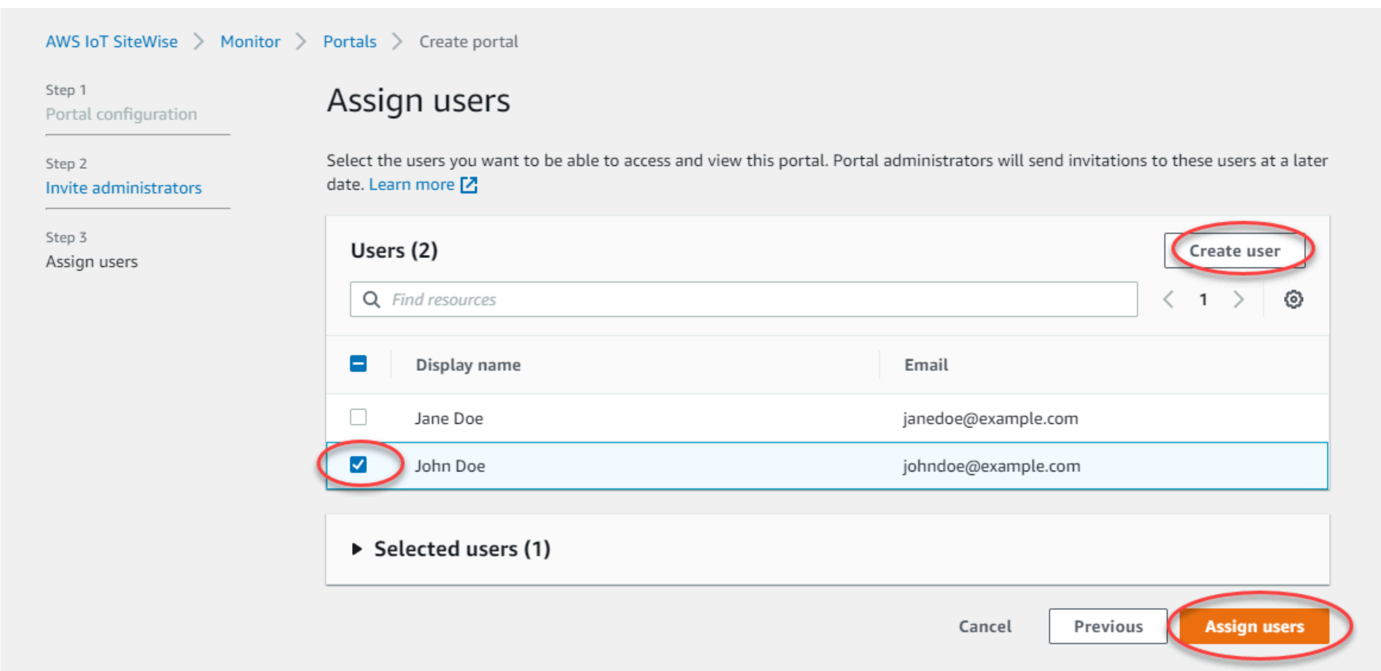
To add portal users

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 IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center](#).

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.

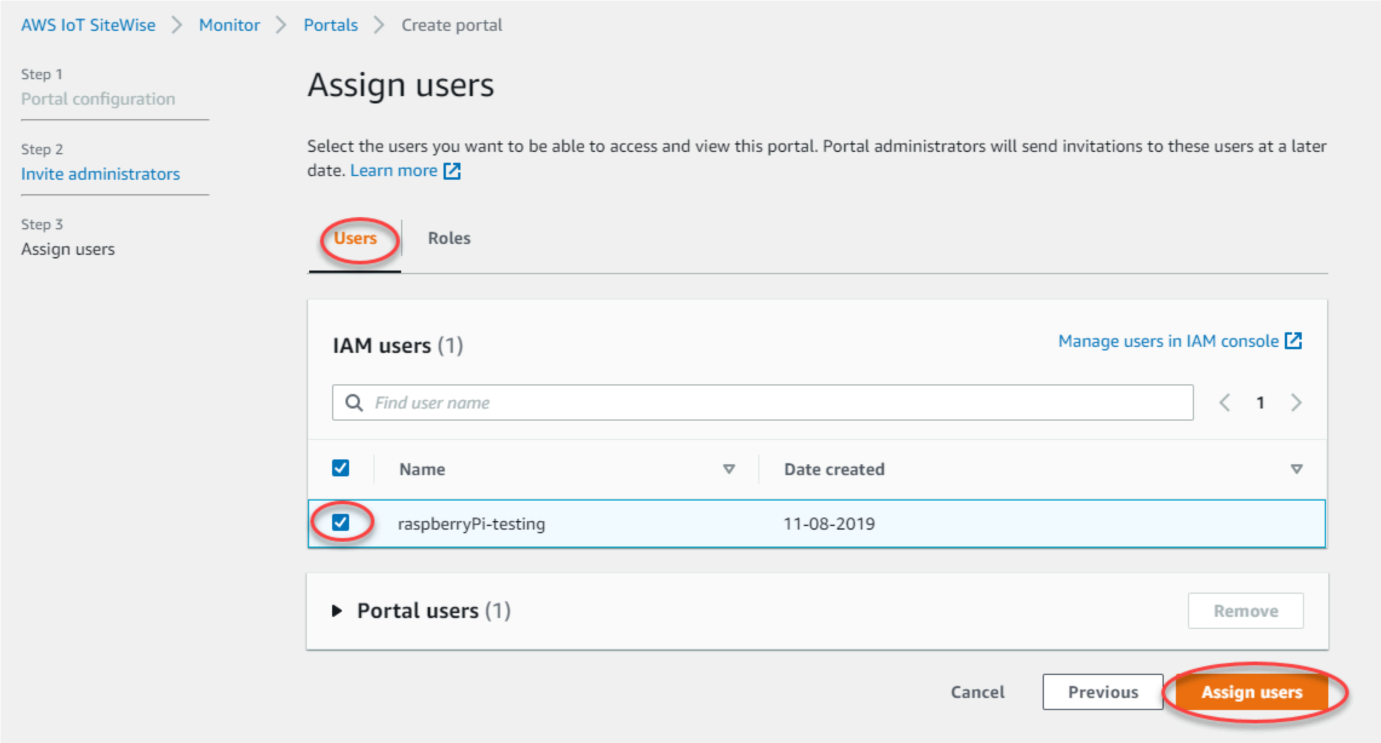
To add portal users

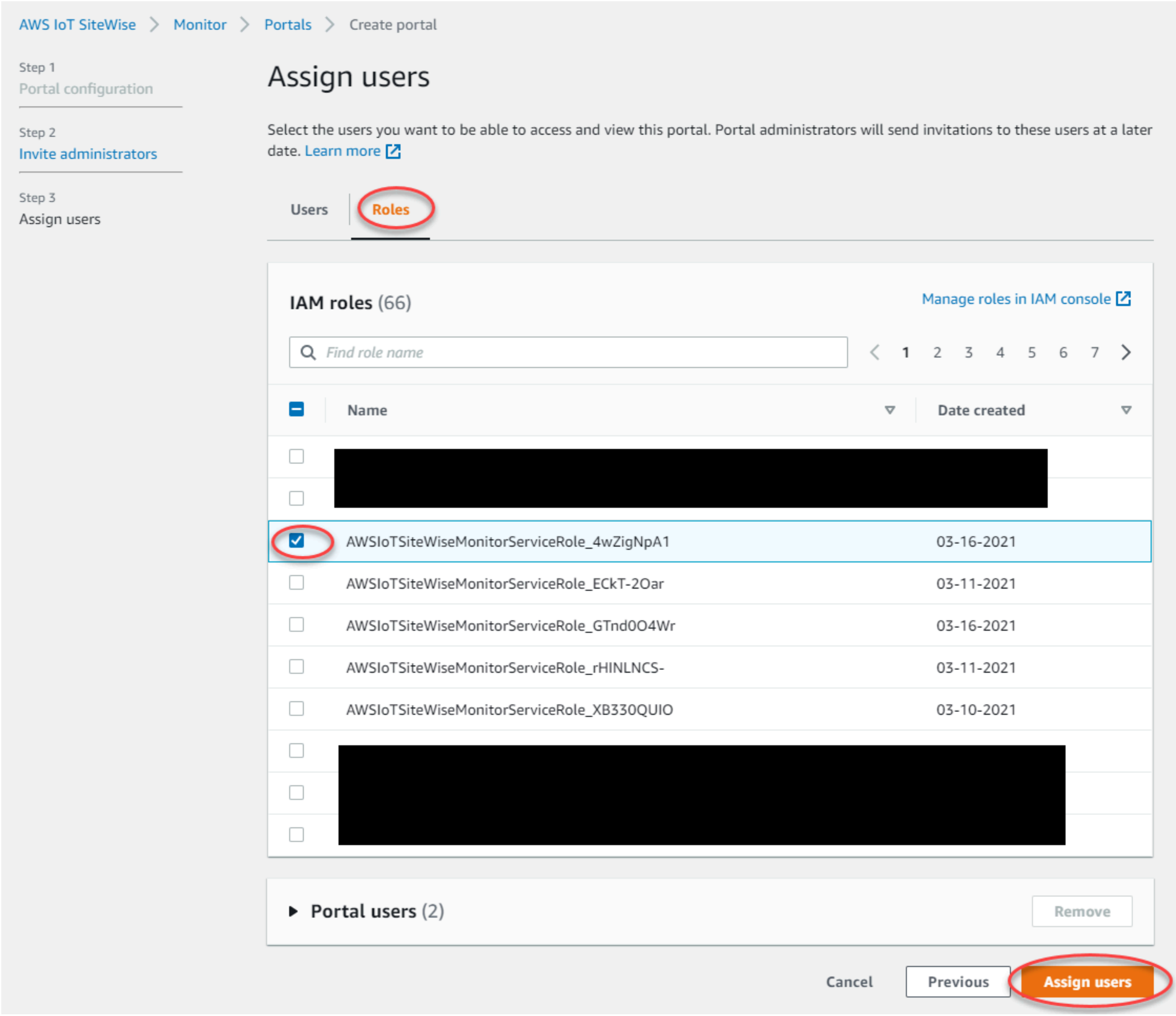
1. Do the following options:

   - 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**
>
> Users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.


AWS IoT SiteWise > Monitor > Portals > Create portal
Step 1
Portal configuration
Step 2
Invite administrators
Step 3
Assign users
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
Users
Roles
IAM users (1)
Manage users in IAM console
Find user name
1
Name
Date created
raspberryPi-testing
11-08-2019
Portal users (1)
Remove
Cancel
Previous
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 Add or remove portal users in AWS IoT SiteWise.

If you need to make changes to the portal, see Administer your SiteWise Monitor portals.

To get started in the portal, see Getting started in the *SiteWise Monitor Application Guide*.

Create AWS IoT SiteWise dashboards (AWS CLI)

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`).

`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, scatter charts, and bar charts 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 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-eeeeeEXAMPLE \
```

```
  --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.

```
{
  "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"
        }
```

```
      ]
    }
  ]
}
```

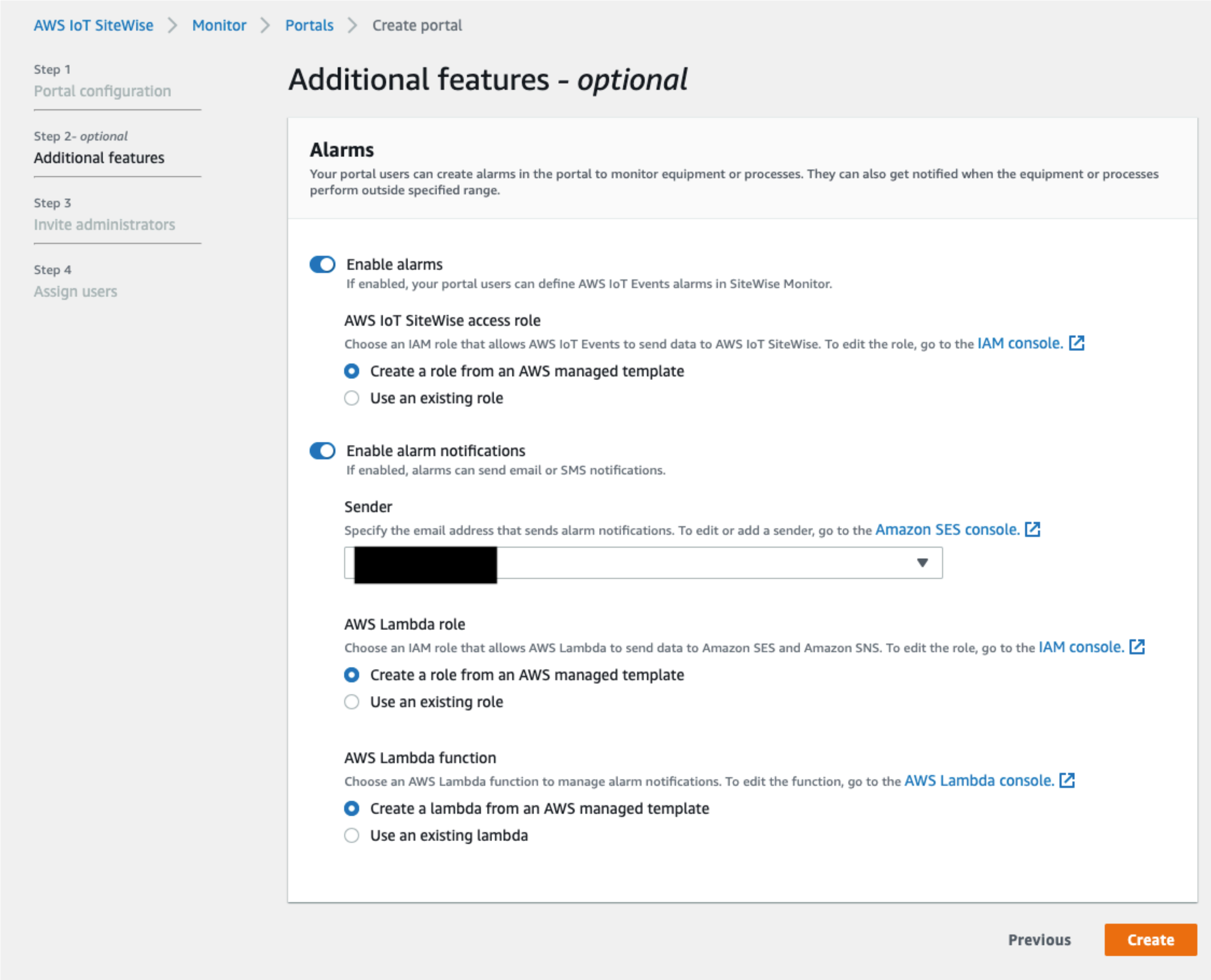
Turn on alarms for your portals in AWS IoT SiteWise

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 IAM Identity Center for the user authentication service.
> - The alarm notifications feature isn't available in the China (Beijing) AWS 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:

IAM Identity Center



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**.

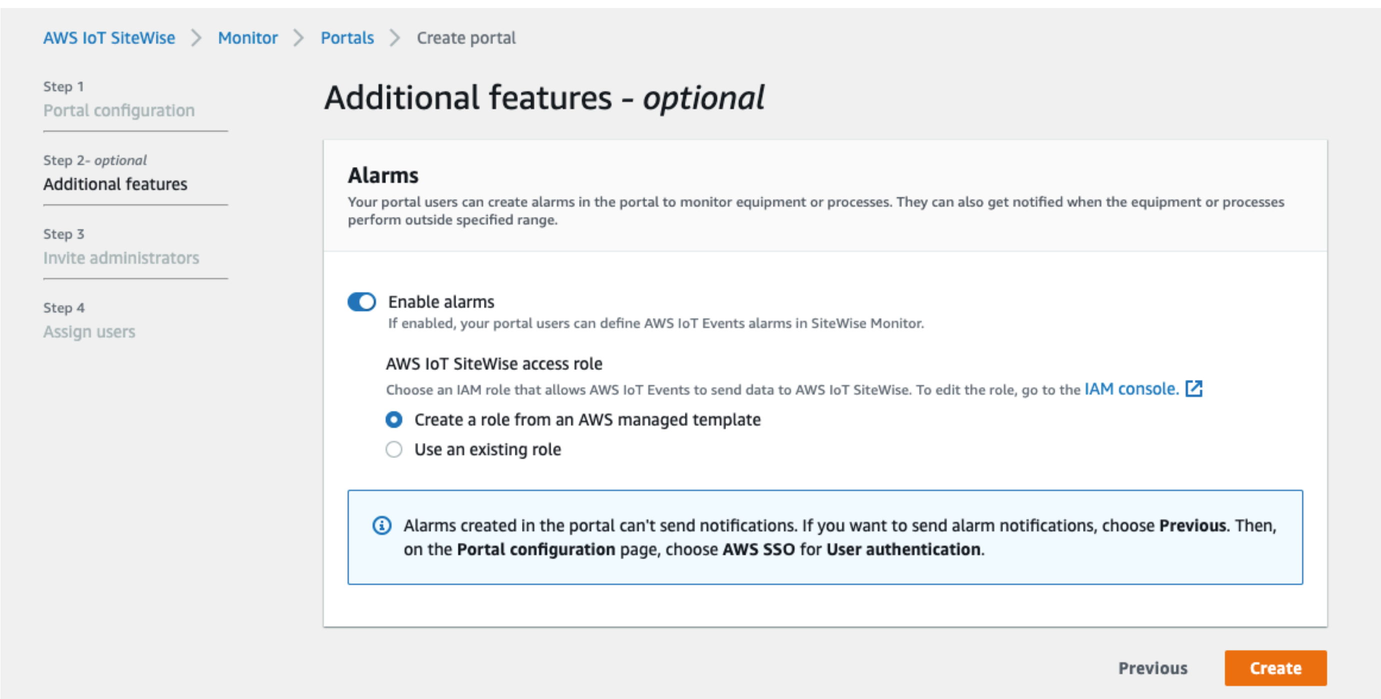
   a. For **Sender**, choose the sender.

> **Important**
>
> You must verify the sender email address in Amazon SES. For more information, see Verifying email addresses in Amazon SES, 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 in the *AWS IoT Events Developer Guide*.

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 AWS IoT SiteWise portal at the edge

After you enable your portal at the edge, this portal is available on all SiteWise Edge gateways with the data processing pack enabled in your account.

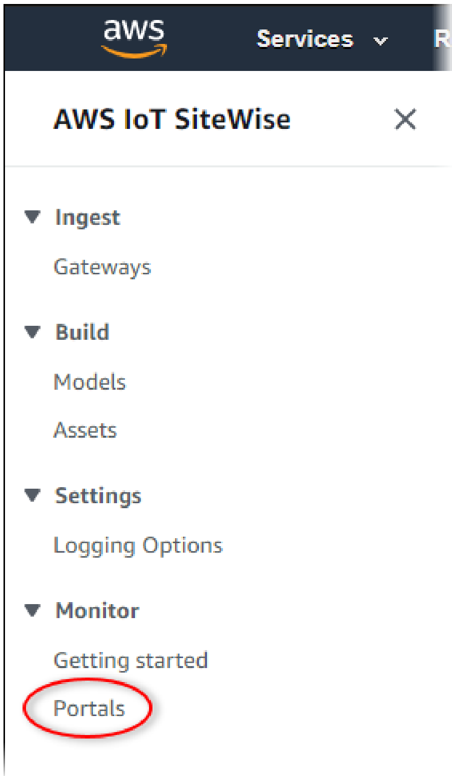
To enable the portal at the edge

1. In the **Edge configuration** section, turn on **Enable this portal at the edge**.
2. Choose **Create**.

Administer your SiteWise Monitor portals

You have the ability to manage and configure various aspects of the portal. This includes adding and removing users or administrators, setting user permissions and roles, customizing the portal's URL, name, setting a support contact information, and sending email invitations to portal administrators.

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Monitor**, **Portals**.



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:
   - Change portal details in AWS IoT SiteWise
   - Add or remove portal administrators in AWS IoT SiteWise
   - Send email invitations to portal administrators
   - Add or remove portal users in AWS IoT SiteWise
   - Delete a portal in AWS IoT SiteWise

For information about how to create a portal, see Get started with AWS IoT SiteWise Monitor (Classic).

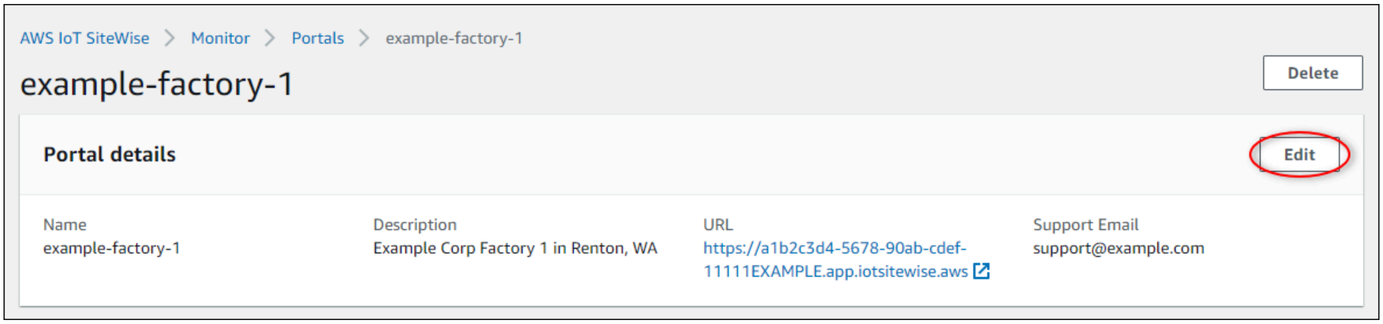
Topics

- Change portal details in AWS IoT SiteWise
- Add or remove portal administrators in AWS IoT SiteWise
- Send email invitations to portal administrators
- Add or remove portal users in AWS IoT SiteWise
- Delete a portal in AWS IoT SiteWise

Change portal details in AWS IoT SiteWise

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**.

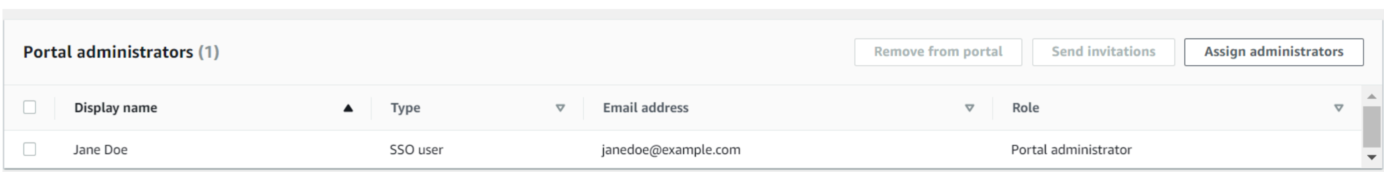


2. Update the **Name**, **Description**, **Portal branding**, **Support contact email**, or **Permissions**.
3. When you're finished, choose **Save**.

Add or remove portal administrators in AWS IoT SiteWise

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.

IAM Identity Center



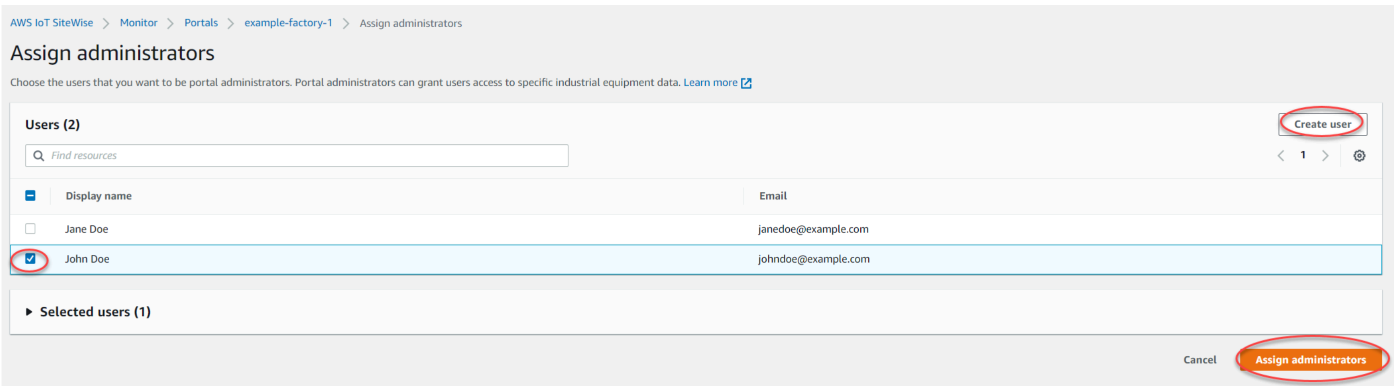
To add portal administrators

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 IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center](#).

3. Choose **Assign administrators**.

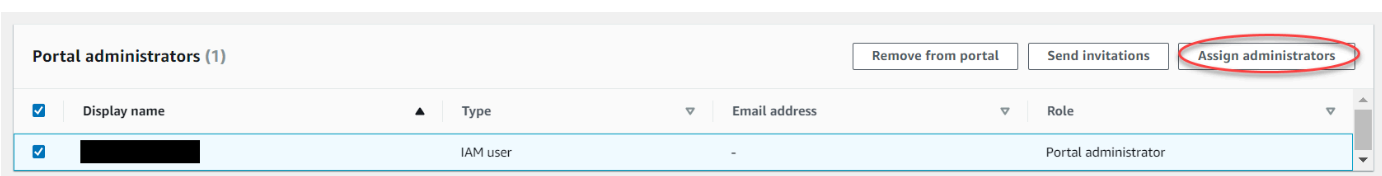


To remove portal administrators

- 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:

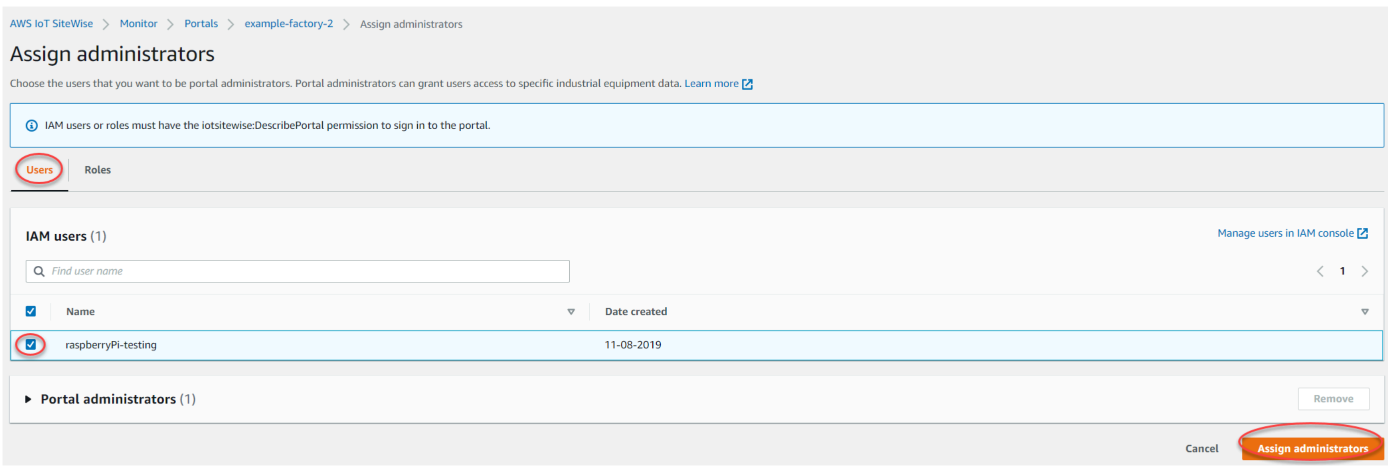
   - Choose **IAM users**, if you want to add an IAM user as your portal administrator.

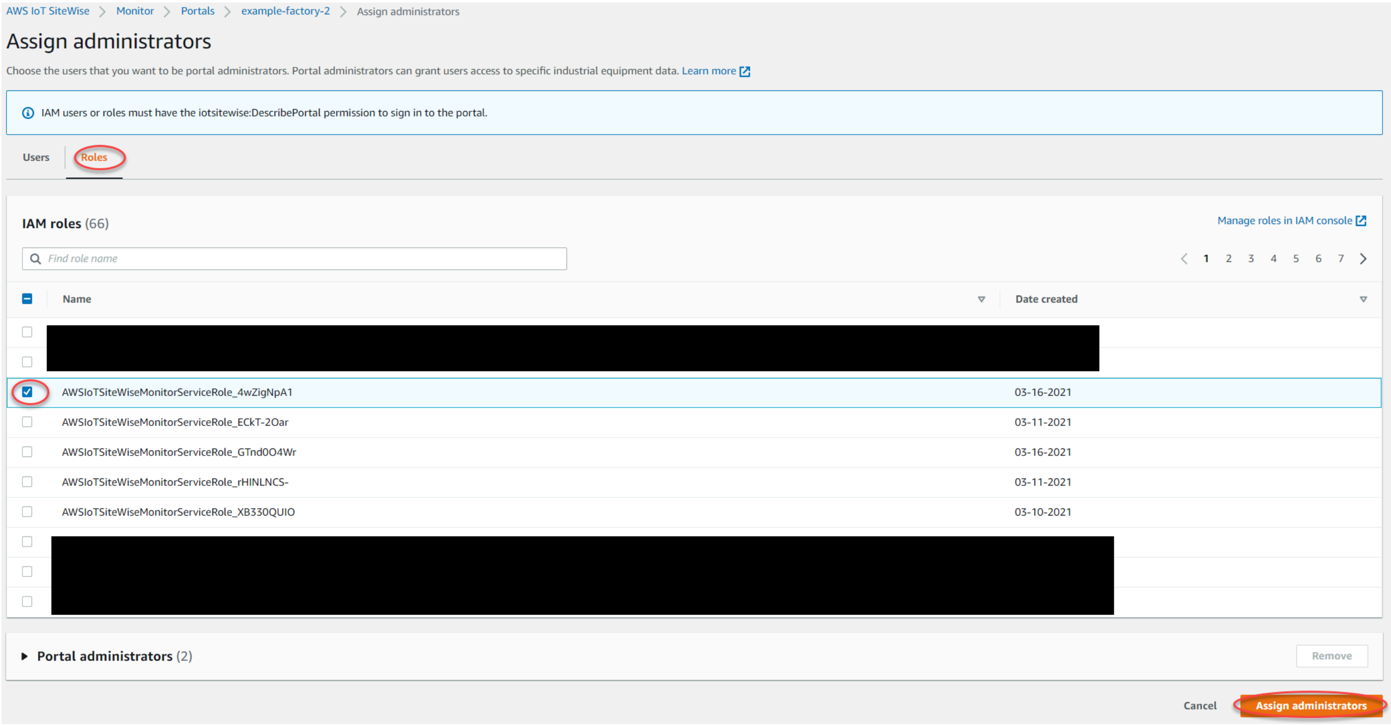
- Choose **IAM roles**, if you want to add an IAM role as your portal administrator.

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**
>
> Users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.





To remove portal administrators

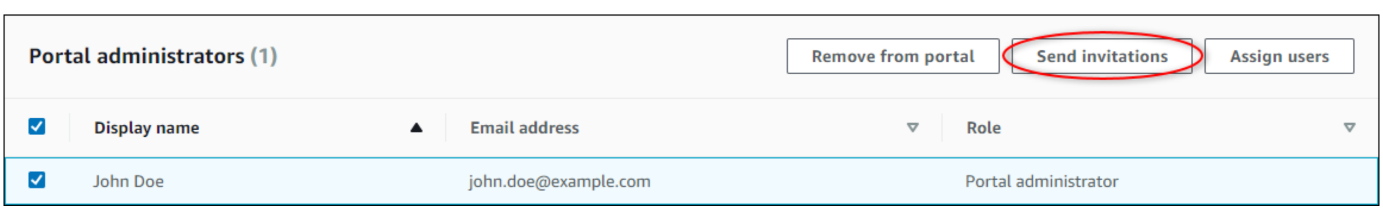
- On the portal details page, in the **Portal administrators** section, select the check box for each user to remove, and then choose **Remove from portal**.

> **Note**
>
> Leaving a portal without a portal administrator is not recommended.

Send 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.



2. Choose **Send invitations**. Your email client opens, and an invitation is populated in the message body.

   You can customize the email before you send it to your portal administrators.

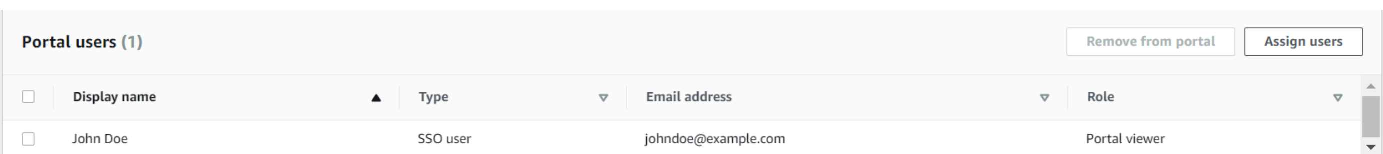
Add or remove portal users in AWS IoT SiteWise

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.

IAM Identity Center

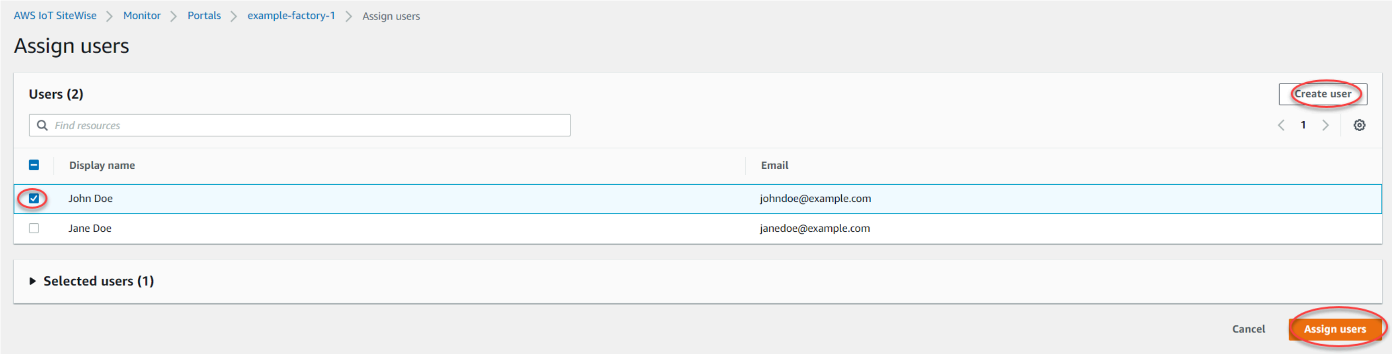


To add portal users

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 IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center](#).

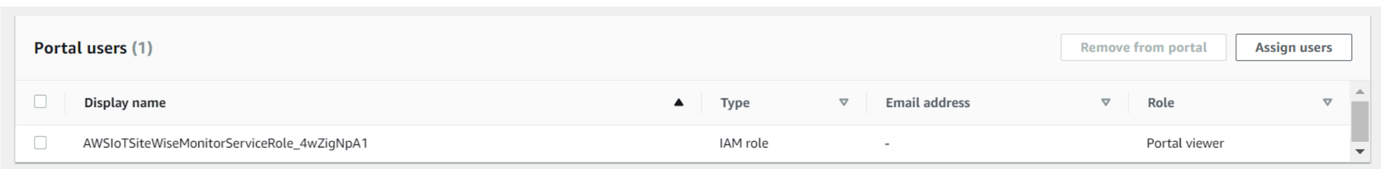
3. Choose **Assign users**.



To remove portal users

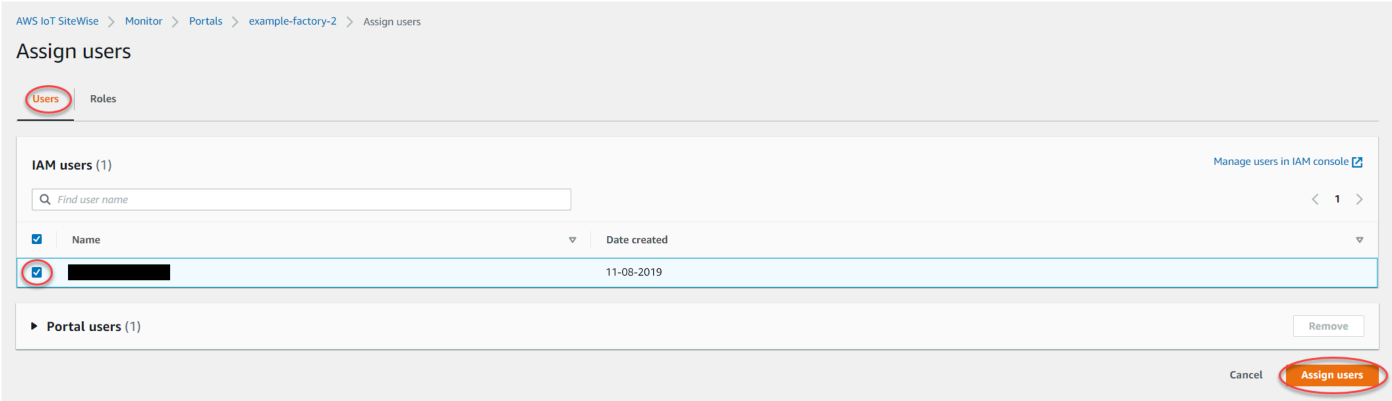
- On the portal details page, in the **Portal users** section, select the check box for the users to remove from the portal, and then choose **Remove from portal**.

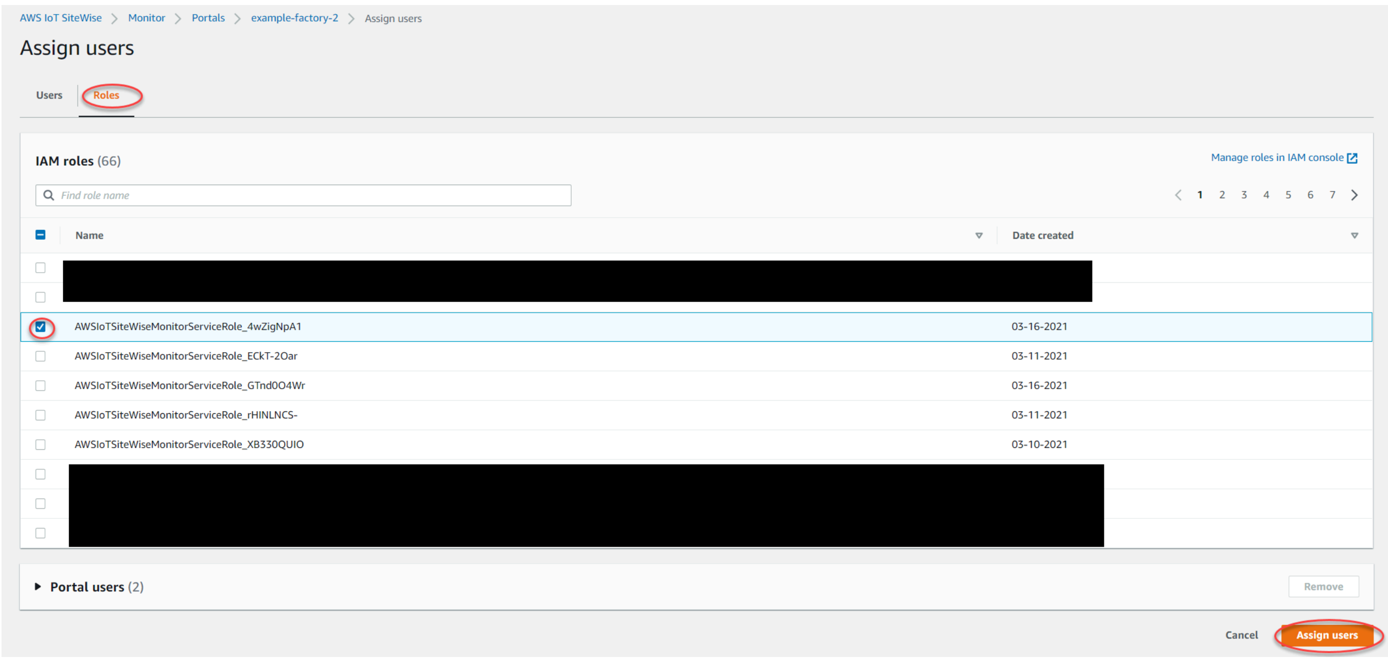
IAM



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**.





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**.

> **Important**
>
> Users or roles must have the `iotsitewise:DescribePortal` permission to sign in to the portal.

Delete a portal in AWS IoT SiteWise

You might delete a portal if you created it for testing purposes or if you created a duplicate of a portal that already exists.

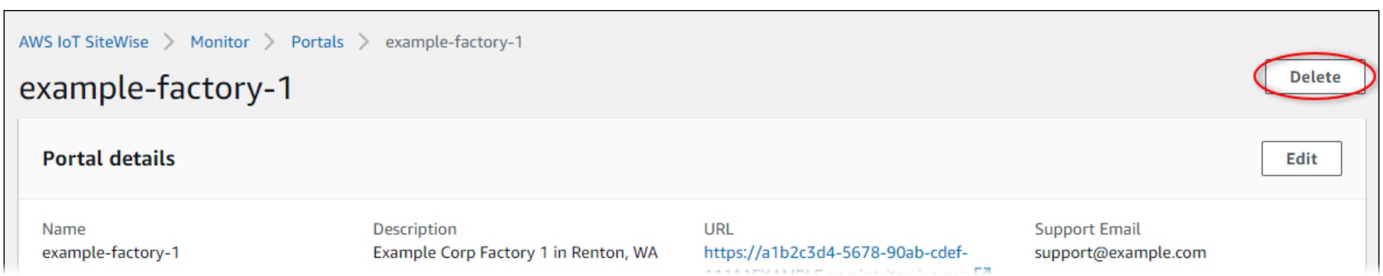
Note

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**.

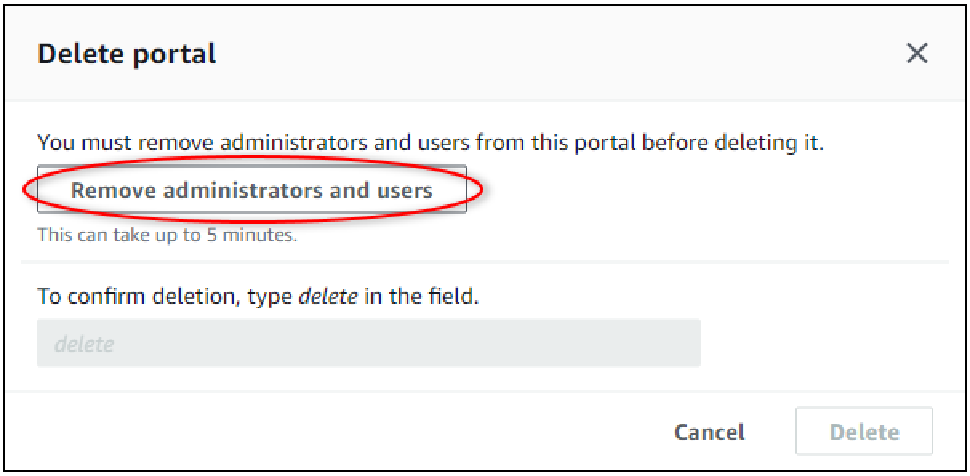
   Important

   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.

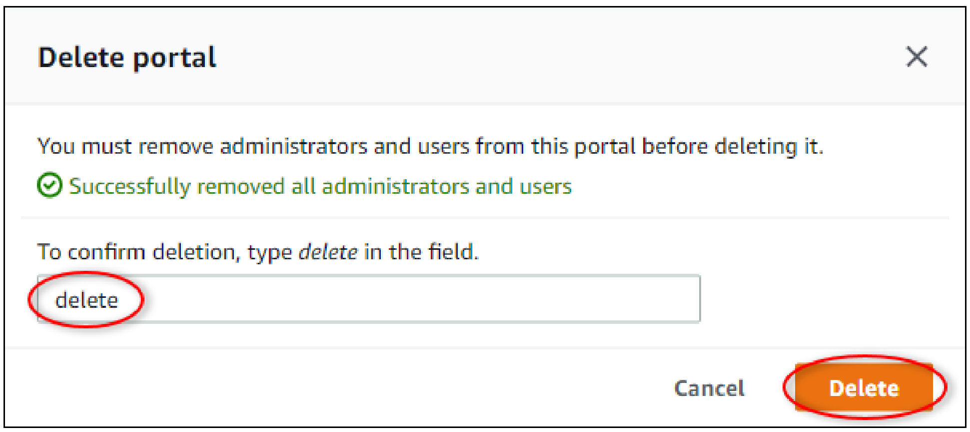
   

2. In the **Delete portals** dialog box, choose **Remove admins and users**.

   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**.

Get started with AWS IoT SiteWise Monitor (AI-aware) - *preview*

As an AWS administrator for your organization, you can create portals from the AWS IoT SiteWise console, enabling members of your organization to view your AWS IoT SiteWise data. Complete the following steps to get started.

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 create projects and add user to the project. Project members then create dashboards to visualize the connected data on AWS IoT SiteWise, enable them to monitor how their connected devices, processes, and equipment are performing.

Note

When adding users or administrators to the portal, avoid creating AWS Identity and Access Management (IAM) policies that restrict user permissions, such as limited IP. Any attached policies with restricted permissions will not be able to connect to the AWS IoT SiteWise portal.

Create projects to share with your teams. Project owners can then create dashboards to visualize the properties of the assets that helps project viewers understand how devices, processes, and equipment are performing. It also provides a consistent view of operations to your teams.

Dashboards help visualize and understand your project data. It helps businesses and application users keep track of their AWS IoT devices and data. Choose a visualization type that best displays your data for your needs. Rearrange and re-size visualizations to create a layout that fits your team. Explore your device, process, and equipment assets and data, and quickly identify issues and improve operational efficiency.

Topics

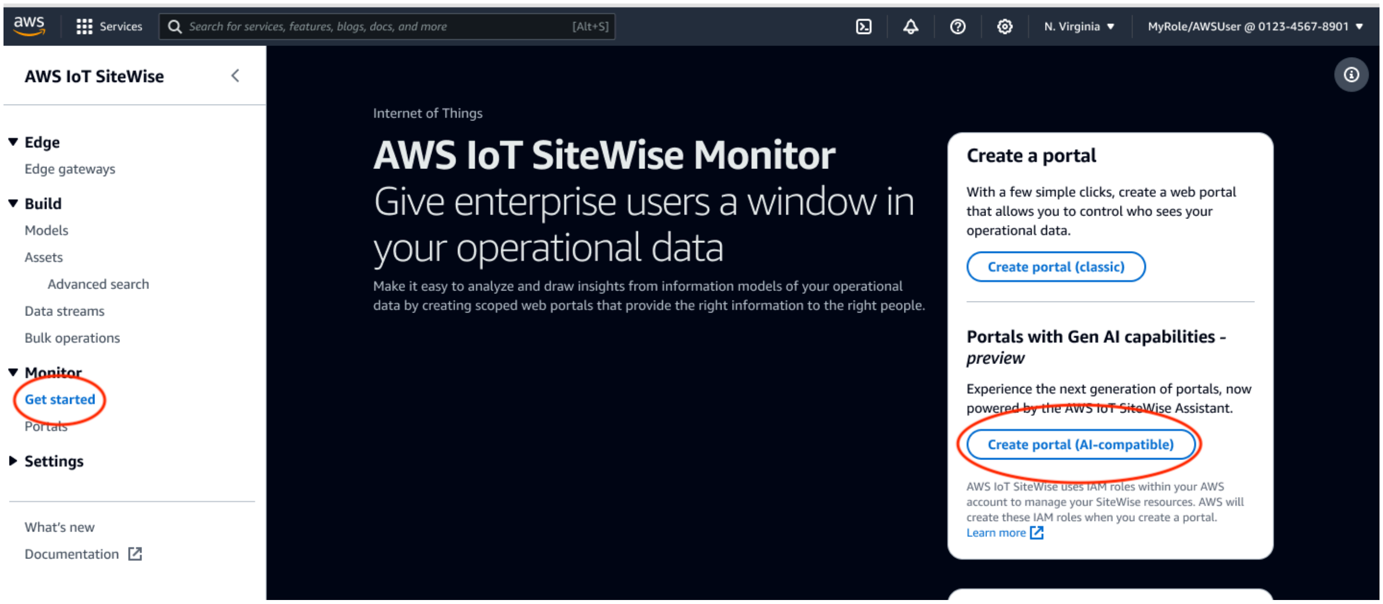
- [Configure dashboard]

Create 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**, **Get started**.
3. Choose **Create portal (AI-aware)**.



Next, you must provide some basic information to configure your portal.

Configure 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.

Steps 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. Enter an email address in the **Support contact email** box for support issues.
5. In the **User authentication** box, choose the following option:
    - Choose **IAM Identity Center** if your portal users sign in to this portal with their corporate user names and passwords.

      If you haven't enabled IAM Identity Center 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**.

    > **Note**
    >
    > Support for IAM credentials is coming soon.

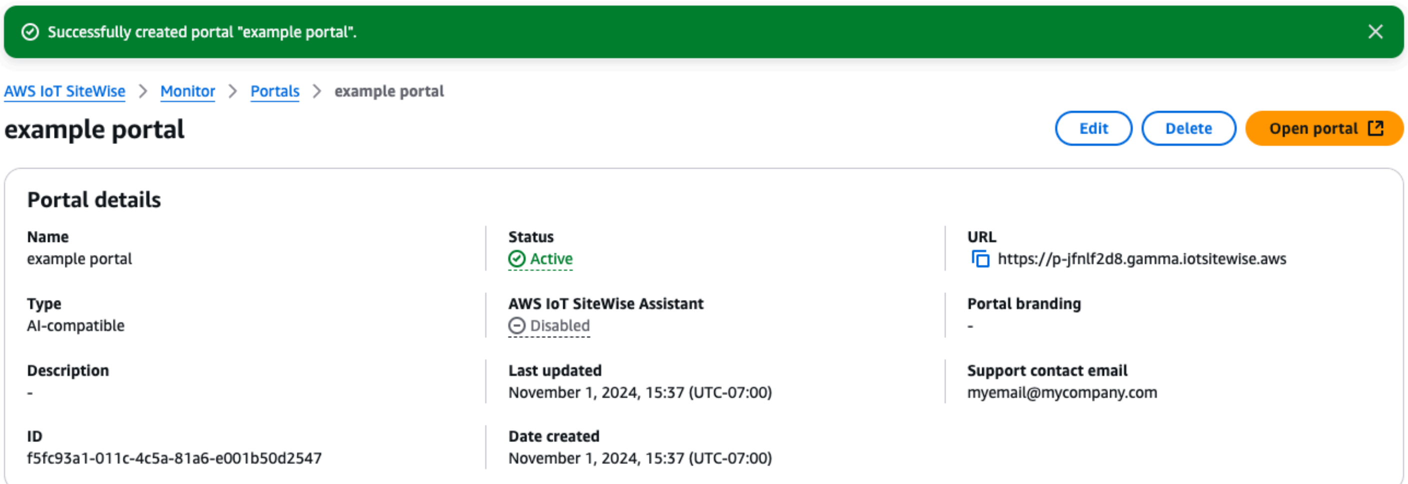
6. Choose from one of the following options in the **Service access** section:
    - Choose **Create and use a new service role**. By default, SiteWise Monitor automatically creates a service role for each portal. This role allows your portal users to access your AWS IoT SiteWise resources. For more information, see Use service roles for AWS IoT SiteWise Monitor.
    - Choose **Use an existing service role**, and then choose the target role.
7. Choose to enable the AWS IoT SiteWise Assistant for this portal. The AWS IoT SiteWise Assistant provides fast data analysis, real-time insights, and guided recommendations.

    > **Note**
    >
    > Enabling the AWS IoT SiteWise Assistant will incur charges. To use enterprise level knowledge solutions and guidance, you must have a dataset associated with Amazon Kendra index.

8. (Optional) Add tags for your portal. For more information, see Tag your AWS IoT SiteWise resources.
9. Choose **Create portal**. 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 Add or remove portal administrators. If you don't want to keep this portal, delete it so it doesn't use resources. For more information, see Delete a portal.

A message appears when your portal is created.



Once a portal is created, it is listed in the **Portals** section. The **Portal details** section lists the name, description, ID, URL, status, last updated and created dates, portal branding and support email for each portal.

The **Status** column can be one of the following values.

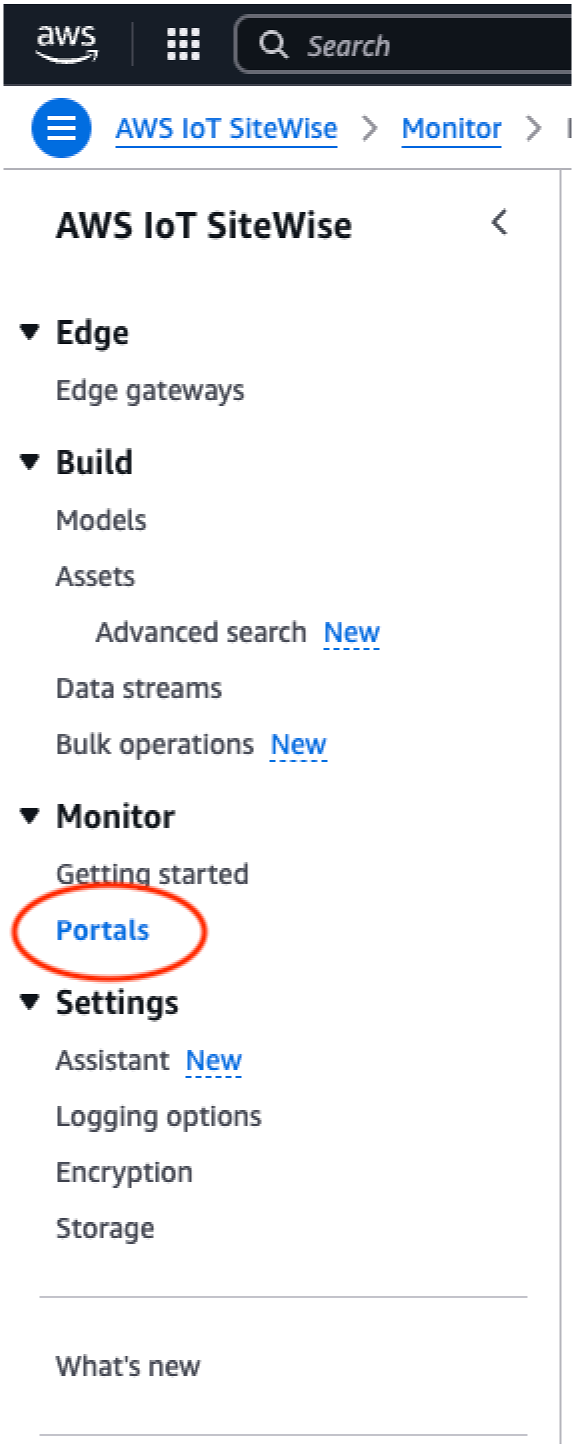
- **CREATING** - AWS IoT SiteWise is processing your request to create the portal. This process can take several minutes to complete.
- **UPDATING** - AWS IoT SiteWise is processing your request to update the portal. This process can take several minutes to complete.
- **PENDING** - AWS IoT SiteWise is waiting for the DNS record propagation to finish. This process can take several minutes to complete. You can delete the portal while the status is **PENDING**.
- **DELETING** - AWS IoT SiteWise is processing your request to delete the portal. This process can take several minutes to complete.
- **ACTIVE** - When the portal becomes active, your portal users can access it.

- **FAILED** - AWS IoT SiteWise couldn't process your request to create, update, or delete the portal. 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 CloudWatch Logs.

Administer your portals

You have the ability to manage and configure various aspects of the portal. This includes adding and removing administrators, setting permissions and roles, customizing the name, description, setting up support email, and inviting to portal administrators.

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Monitor**, **Portals**.



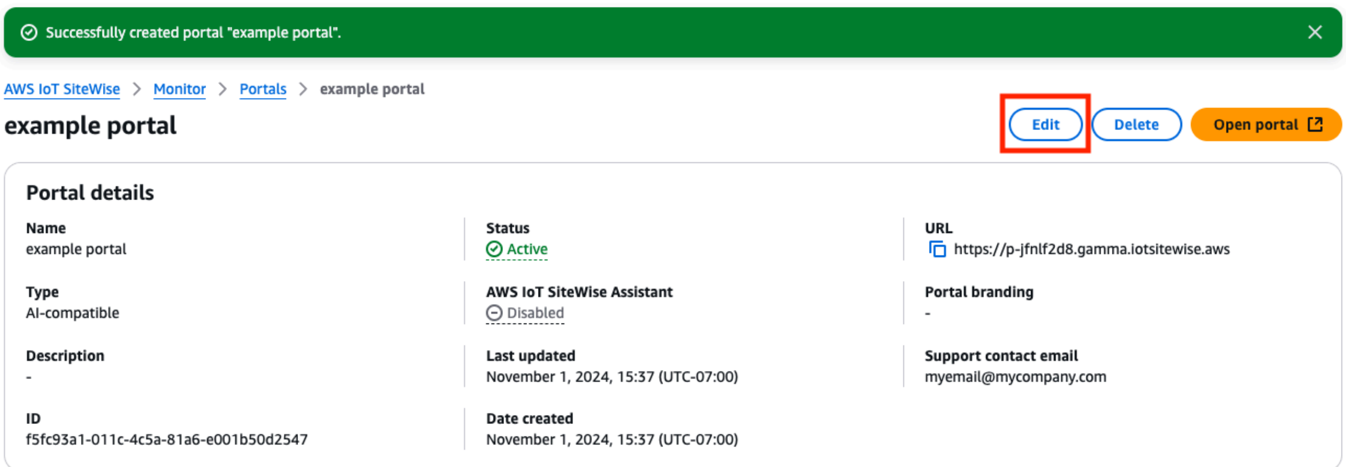
3. Choose a portal, and then choose **Open portal** (or choose the portal's **Name**).
4. You can perform any of the following administrative tasks:
    - [Edit portal attributes]
    - [Add or remove portal administrators]

- Send email invitations to portal administrators
- Delete a portal in AWS IoT SiteWise

Edit portal attributes

You can change a portal's name, description, branding, support email, and service access.

1. On the portal details page, in the **Portal details** section, choose **Edit**.

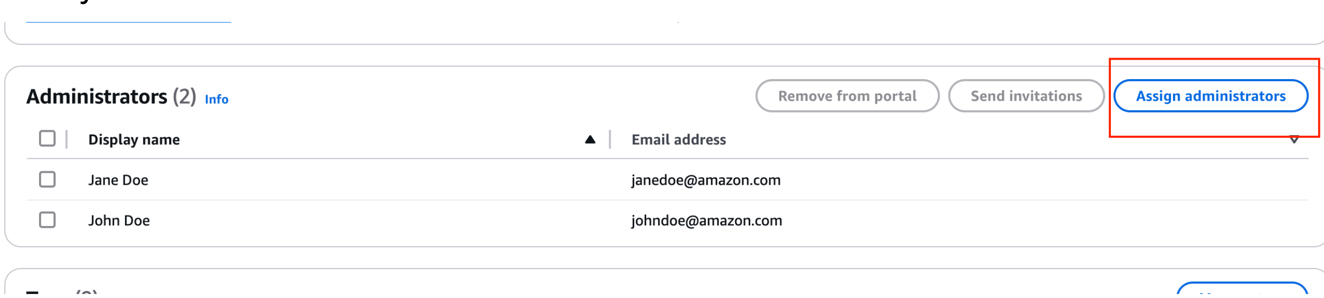


2. Update the **Name**, **Description**, **Portal branding**, **Support contact email**, **AWS IoT SiteWise Assistant** or **Service access**.
3. When you're finished, choose **Save changes**.

Add or remove 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.

IAM Identity Center

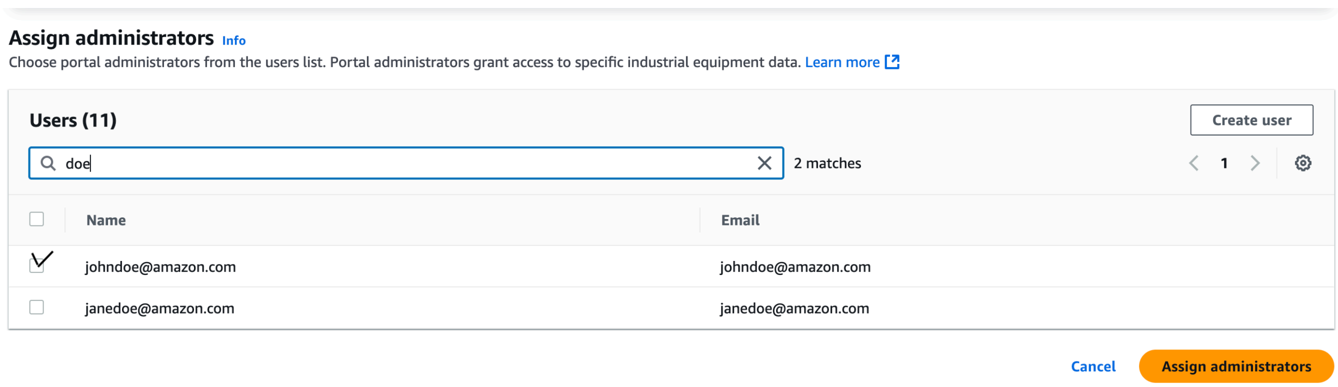


To add portal administrators

1. On the portal details page, in the **Administrators** section, choose **Assign administrators**.
2. On the **Assign administrators** page, select the users to add to the portal as administrators.

> **Note**
>
> If you use IAM Identity Center as your identity store, and you're signed in to your AWS Organizations management account, you can choose **Create user** to create an IAM Identity Center user. IAM Identity Center 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 IAM Identity Center.

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**
>
> The **Administrators(#)** lists the number of administrators for the portal. You can add multiple portal administrators to manage and work on projects.

Send email invitations to portal administrators

You can send email invitations to portal administrators.

1. On the portal details page, in the **Administrators** section, select the check boxes for the portal administrators.
2. Choose **Send invitations**. Your email client opens, and an invitation is populated in the message body.

   You can customize the email before you send it to your portal administrators.

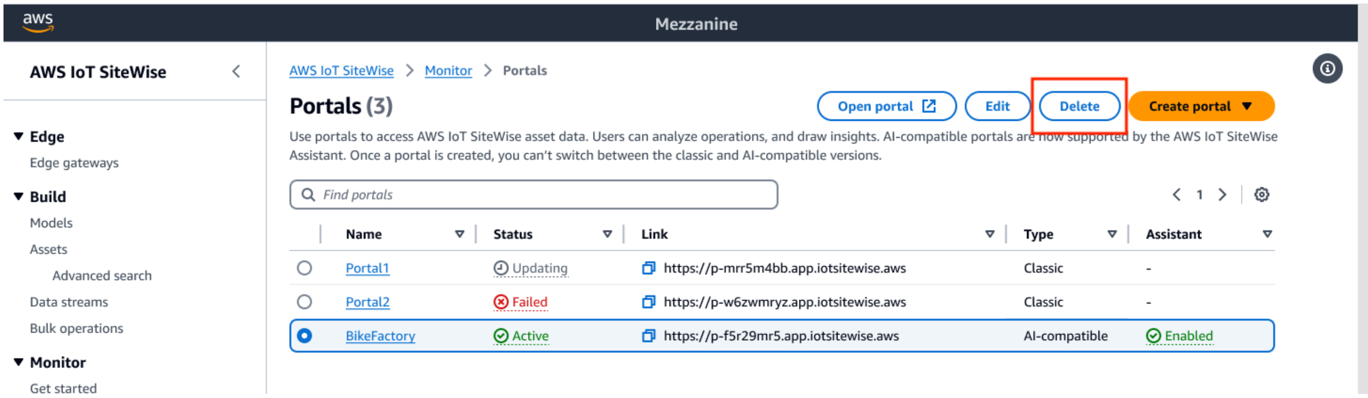
Delete a portal

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.

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 portal** 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 **`confirm`** in the field to confirm deletion.

4. Choose **Delete**.

Create dashboards with AWS CLI

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.

`displaySettings`

The display settings with the following parameters:

- `numRows` – Number of rows in the dashboard layout. Each row is **cellSize** wide.
- `numColumbs` – Number of columns in the dashboard layout. Each column is **cellSize** wide.
- `cellSize` – (Optional) The size of a cell in the layout in pixels. It must be a positive number. Default is 10.
- `significantDigits` – (Optional) Number of significan digits to display in the dashboard. Default is 4.

`querySettings`

The query information with the following parameter:

- `refreshRate` – (Optional) The rate at which data refreshes in milliseconds. Accepts the following values - 1000, 5000, 10000, 60000, 300000.

`defaultViewport`

If not supplied, defaults to the last five minutes. Contains the following parameters:

- `duration` – (Optional) Determines how far into the past to query data starting from the present time.
- `start` – (Optional) It is of type Date. The start time range to query data. Needs an `end` date specified.
- `end` – (Optional) It is of type Date. The end time range to query data. Needs an `start` date specified.

`widgets`

A list of widget definition structures that contain the following information:

`type`

The type of widget. AWS IoT SiteWise provides the following widget types:

- `xy-plot` – A line chart or a scatter plot depending on the configuration.
- `bar-chart` – A bar chart.
- `kpi-chart` – A key performance indicator chart.
- `status-timeline` – A status widget that visualizes and navigates time series data from one or more data sources.
- `text` – A text widget.
- `table` – A table widget.

`id`

An unique identifier for the widget.

`x`

The horizontal position of the widget, starting from the left of the dashboard. 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 dashboard. This value refers to the widget's position in the dashboard's grid.

`z`

The relative ordering of the widgets. A larger Z value widget is displayed in front of the lower Z value widget, if they overlap.

`width`

The width of the widget, expressed in number of cells on the dashboard.

`height`

The height of the widget, expressed in number of cells on the dashboard.

`properties`

A list of properties of the widget. It varies by the type of widget. See IoT App Kit for details.

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-eeeeeEXAMPLE \
  --dashboard-name "Example Dashboard" \
  --dashboard-definition file://dashboard-definition.json
```

The following JSON example for `dashboard-definition.json` defines dashboard with the following visualization widgets:

```
{
    "displaySettings": {
        "numColumns": 200,
        "numRows": 1000,
        "cellSize": 20,
```

```
        "significantDigits": 4
    },
    "widgets": [{
        "id": "Ot73JcxUoc6oEXAMPLE",
        "type": "xy-plot",
        "width": 33,
        "height": 20,
        "x": 0,
        "y": 0,
        "z": 0,
        "properties": {
            "aggregationType": "AVERAGE",
            "queryConfig": {
                "source": "iotsitewise",
                "query": {
                    "assets": [{
                        "assetId": "97c97abf-e883-47bb-a3f4-EXAMPLE",
                        "properties": [{
                            "propertyId": "97cc61f4-57a4-4c5f-a82c-EXAMPLE",
                            "refId": "692ce941-f3d9-4074-a297-EXAMPLE",
                            "aggregationType": "AVERAGE",
                            "color": "#7d2105",
                            "resolution": "1m"
                        }]
                    }],
                    "properties": [],
                    "assetModels": [],
                    "alarms": [],
                    "alarmModels": []
                }
            },
            "line": {
                "connectionStyle": "linear",
                "style": "solid"
            },
            "symbol": {
                "style": "filled-circle"
            },
            "axis": {
                "yVisible": true,
                "xVisible": true
            },
            "legend": {
                "visible": true,
```

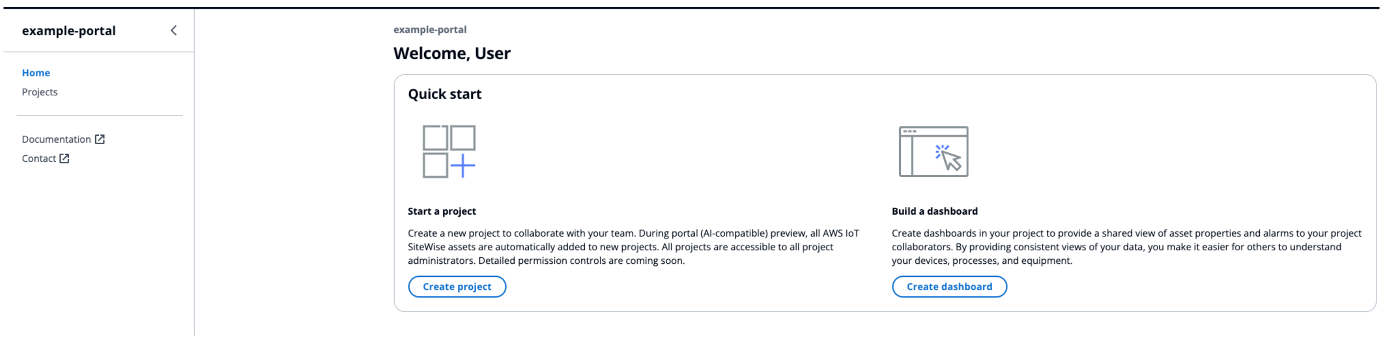
```
                "position": "right",
                "width": "30%",
                "height": "30%",
                "visibleContent": {
                    "unit": true,
                    "asset": true,
                    "latestValue": true,
                    "latestAlarmStateValue": true,
                    "maxValue": false,
                    "minValue": false
                }
            }
        }
    }, {
        "id": "fto7rF40Ny1EXAMPLE-G",
        "type": "bar-chart",
        "width": 33,
        "height": 20,
        "x": 0,
        "y": 20,
        "z": 0,
        "properties": {
            "aggregationType": "AVERAGE",
            "queryConfig": {
                "source": "iotsitewise",
                "query": {
                    "assets": [{
                        "assetId": "97c97abf-e883-47bb-a3f4-EXAMPLE",
                        "properties": [{
                            "propertyId": "c84ca8f3-3dea-478a-afec-EXAMPLE",
                            "aggregationType": "AVERAGE",
                            "refId": "2960b958-2034-4d6e-bcc2-EXAMPLE"
                        }]
                    }],
                    "properties": [],
                    "assetModels": [],
                    "alarms": [],
                    "alarmModels": [],
                    "requestSettings": {
                        "aggregation": "AVERAGE"
                    }
                }
            },
            "axis": {
```

```
                "showX": true,
                "showY": true
            },
            "styleSettings": {
                "2960b958-2034-4d6e-bcc2-360f1f02e505": {
                    "color": "#7d2105"
                }
            }
        }
    }],
    "querySettings": {
        "refreshRate": 5000
    }
}
```

Portal login

User login

1. On your browser, enter the application URL.
2. Input your user name and password and click the **Sign in** button.
3. You are now logged into the application.



Create a project

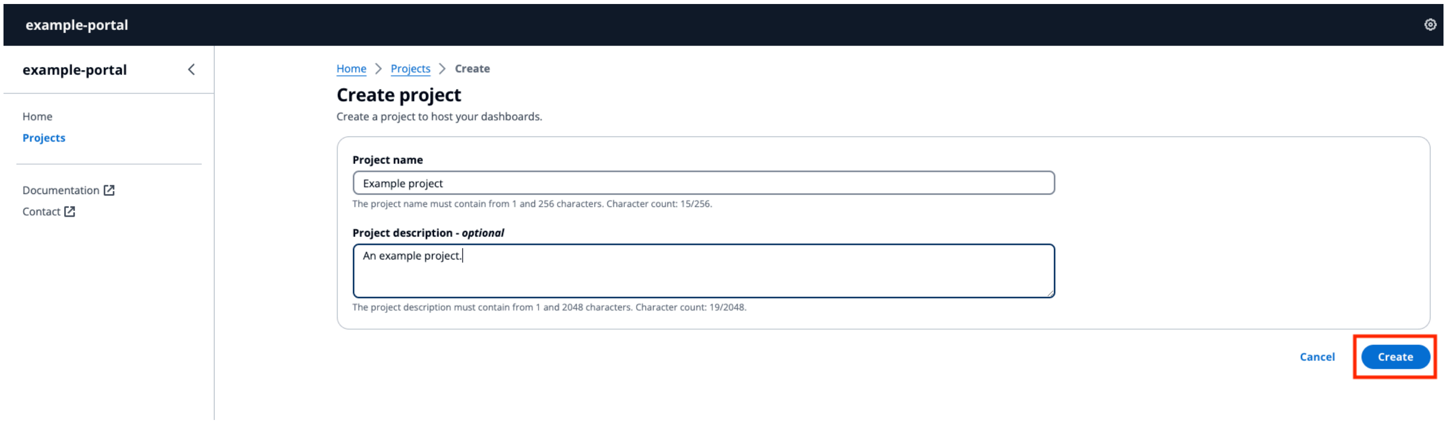
Create project

1. A project is created in two ways:

   a. In the Home page, in the **Welcome** section under **Quick start**, choose **Create project**.

   b. From the left navigation pane, choose **Projects**. Choose **Create** in the top right hand corner to create a project.

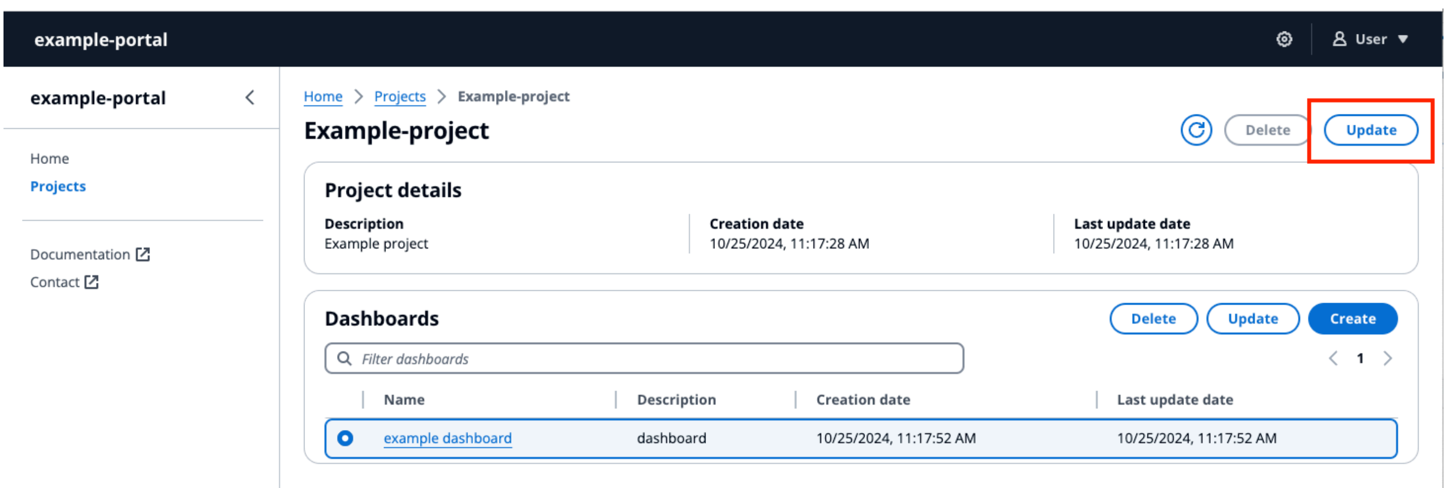
2. In the **Create Projects** section, enter a **Project name**, and give an optional **Description**.
3. Choose **Create**.



Update a project

Edit project

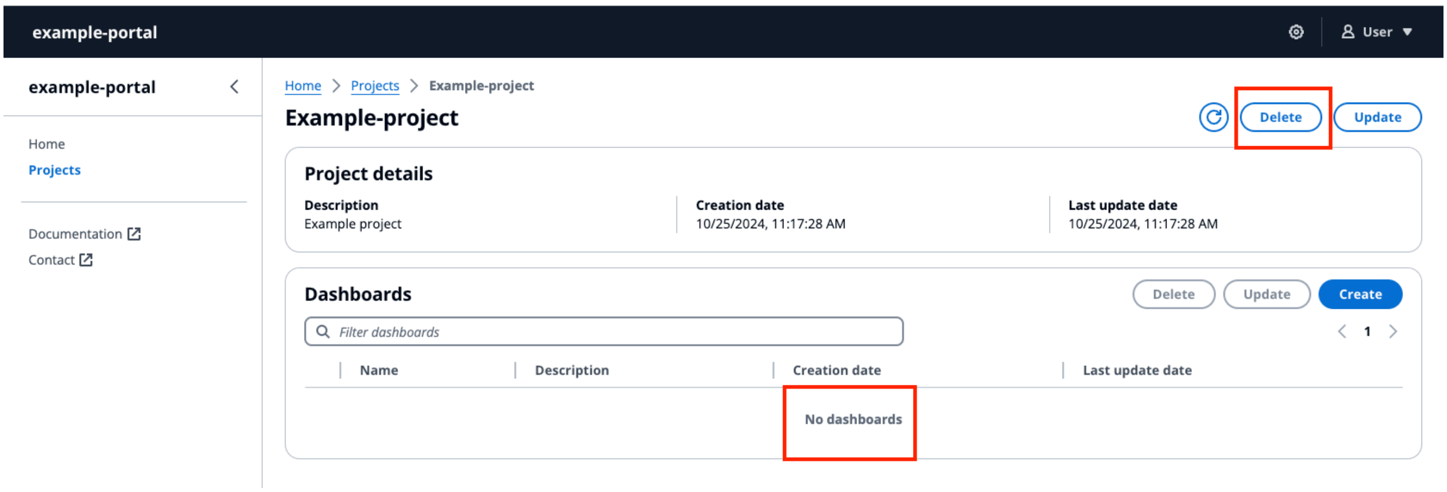
1. Choose the **Update** button on the top right hand corner of the **Project** page, to edit project details.
2. Change the name of the project by editing **Project name**.
3. Change the description of the project by editing the **Description** details.
4. Select **Update** to save your changes.



Delete a project

Delete project

1. You can only delete the project after all the dashboards in the project are deleted.
2. Select the **Delete** button on the top right of the **Project** page.
3. Confirm again that you want to delete the project.
4. Select **Delete** to delete the project.



Create a dashboard

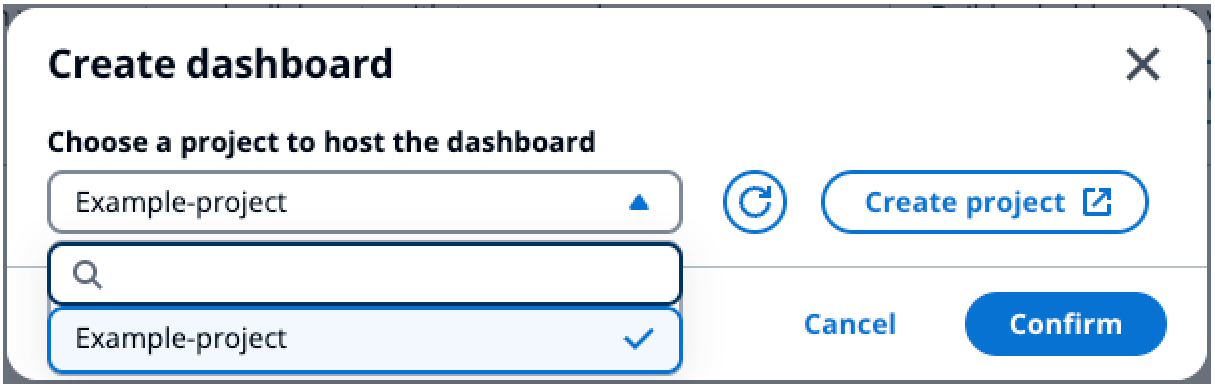
Create a dashboard

1. Create a dashboard in two ways:

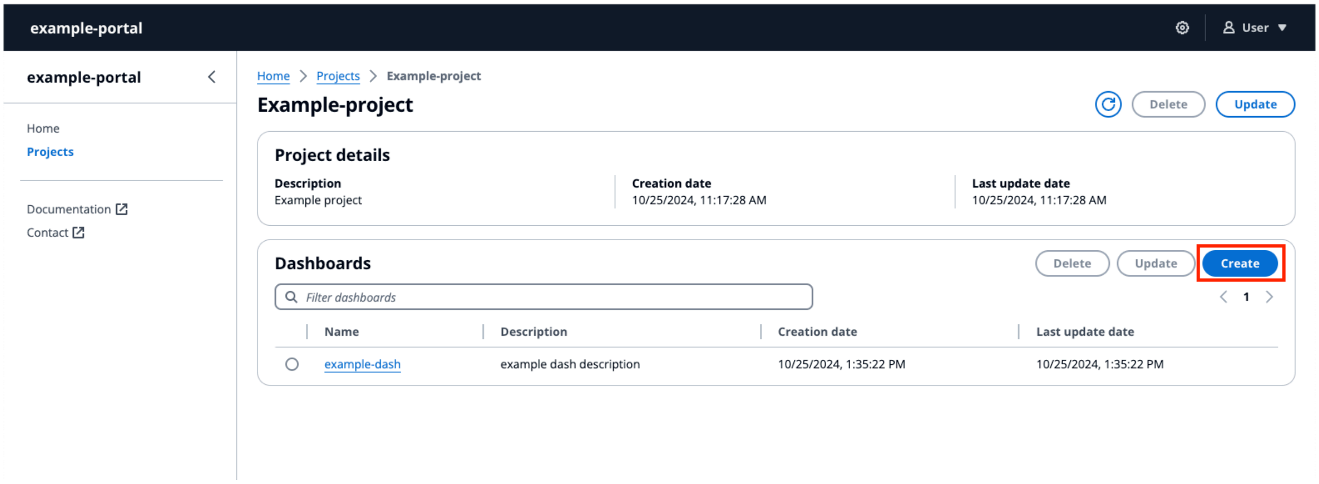
   a. Create a dashboard from **Build a dashboard** in the **Home** page.

      i. To create the dashboard in an existing project, choose a project name from the drop down menu in the **Choose a project to host the dashboard**.

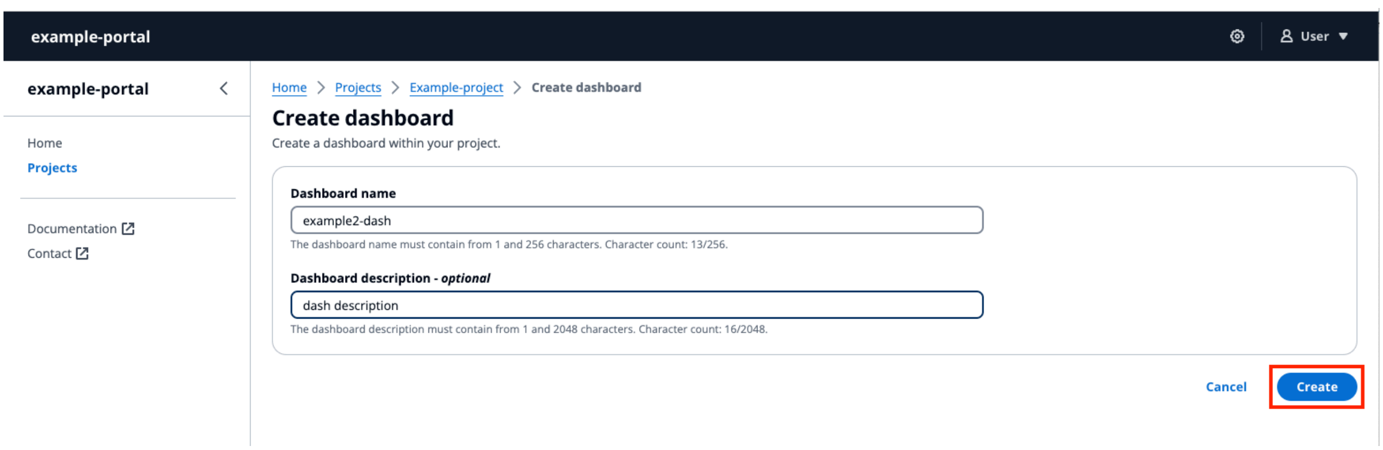
      ii. If you don't have a project, choose **Create project** and select **Confirm**.



b. Create a dashboard from a project in the **Projects** section, under **Dashboards**.



2. Select **Create** in the upper right corner.
3. Enter a **Dashboard name**, and give an optional **Dashboard description**.
4. Select **Create**.



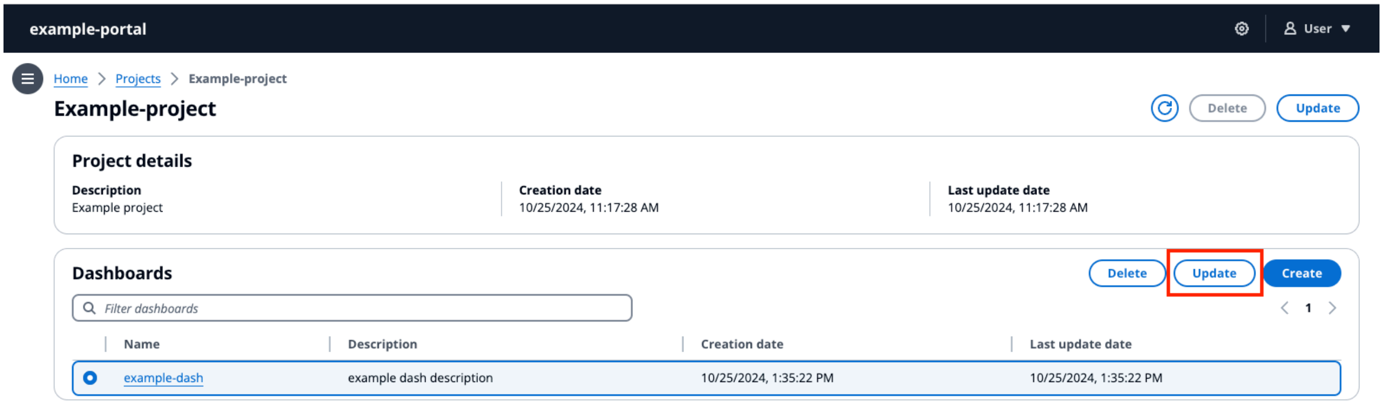
5. Configure your newly created dashboard.

Update a dashboard

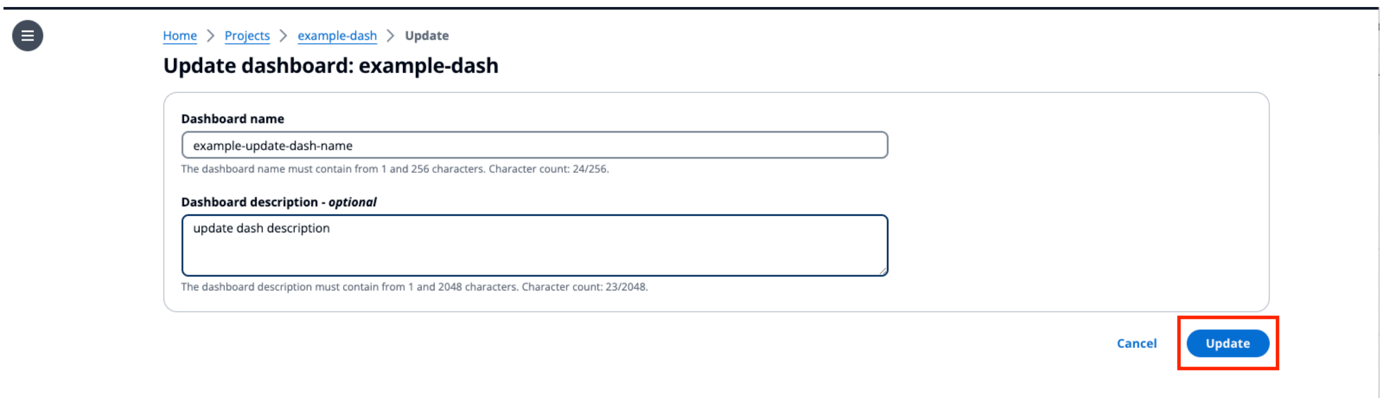
The **Dashboards** section lists the dashboards in the project. Select a dashboard from the list.

Update a dashboard

1. Select a dashboard to update.



2. Update the **Dashboard name** and optionally the **Dashboard description**. Select **Update** to save changes.

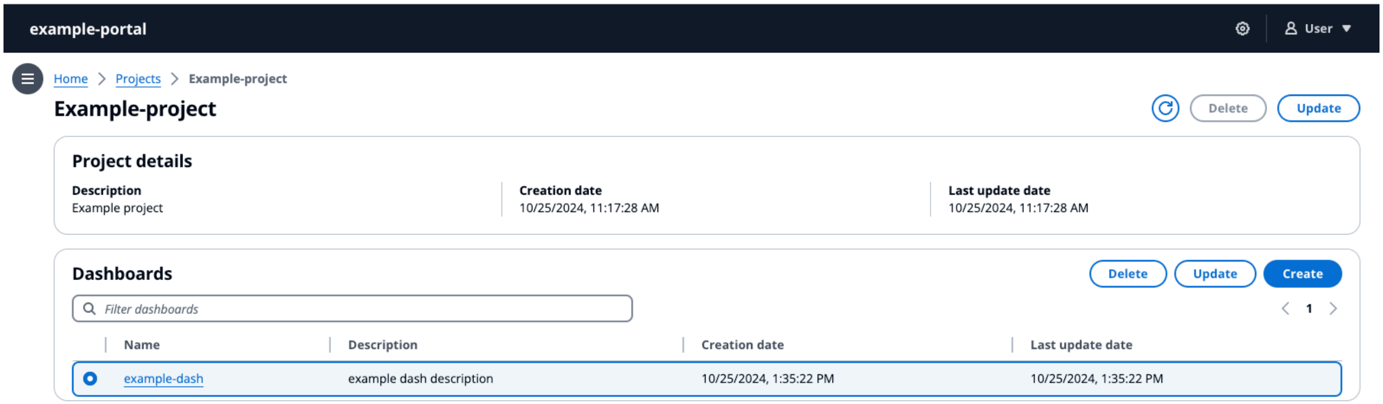


Delete a dashboard

The **Dashboards** section lists the dashboards in the project. Select a dashboard from the list.

Delete a dashboard

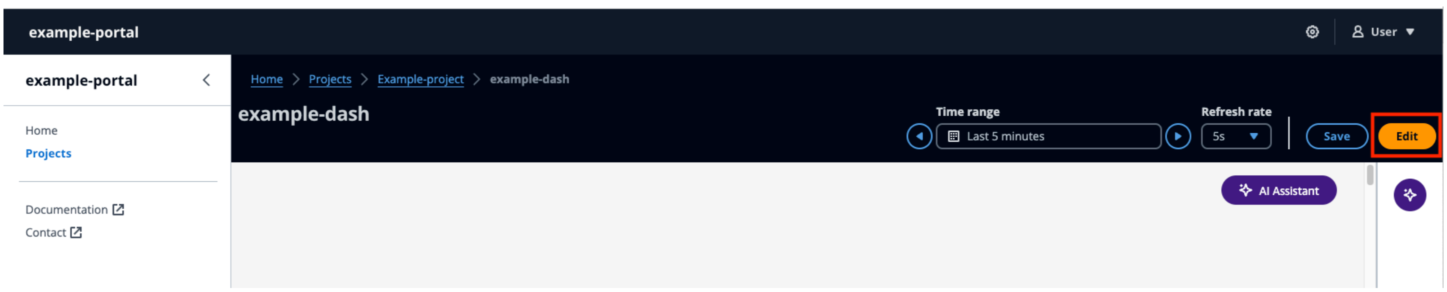
1. Select a dashboard to delete.



2. Select **Delete** to delete the dashboard. This cannot be undone.

Configure dashboard

The **Dashboards** section lists the dashboards in the project. Select a dashboard from the list. The **Edit** mode allows you to configure your dashboard by adding widgets and configuring them. The **Preview** button lets you visualize your changes.



Steps to configure your dashboard:

- Drag and drop different type of data widgets to the dashboard canvas for data visualization.
- Add data to the desired widgets, from the **Resource explorer** on the left. The **Resource explorer** consists of **Modeled**, **Unmodeled**, and **Dynamic assets** sections. Search by asset name or property name. Select the property to add and choose **Add**.
- Fine tune the layout and style by changing the **Configurations** on widgets. Configure components including title, thresholds and other configuration specifics.
- Configure the time range over which data is displayed.

  - Choose the time range over which data is displayed. Choose a **Time range** and **Refresh rate** from the top right hand corner, and personalize the range. Choose a rate at which the data is to be refreshed from the menu.
  - Select the **Time range** on a widget, by using your trackball mouse scroll wheel or Right-click. This moves the time range of display.
- Choose **Save**.

Topics

Resource explorer

This section describes **Modeled**, **Unmodeled**, and **Dynamic assets**. Choose assets from any of the three and add them to your widgets and visualize them.

Topics

Modeled

This section describes the process of selecting and visualization of modeled assets.

Selection of assets

Assets can be queried as follows:

- Search for an asset name. Use a wildcard `*`. For example, `Wind*` returns asset names that start with the text `Wind`. You must integrate with AWS IoT TwinMaker to avail this feature.
- All assets are listed by default.

From the assets listed, filter by name, description, ID, or asset model ID. Select one asset to list its properties (data streams) and alarms.

Data stream selection

Data streams are listed below the **Data Streams** menu. Filter the data streams listed by Property metadata in the *https://docs.aws.amazon.com/iot-sitewise/latest/APIReference/*. Select one or more data streams depending on the selected widget.

- **KPI** and **Gauge** support only a single data stream.
- The remaining widgets support multiple data streams with multi-selection.

Alarm selection

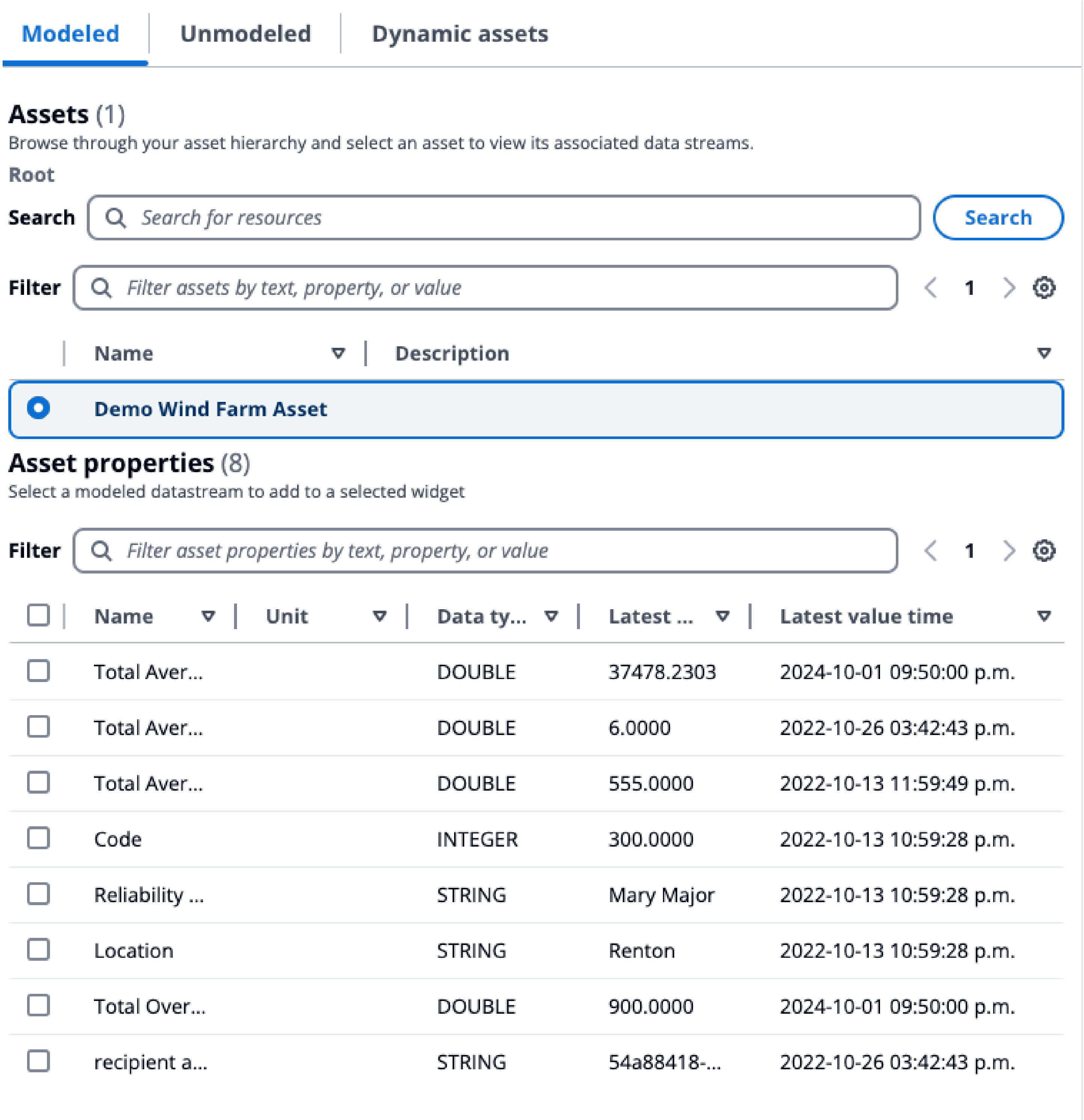
AWS IoT SiteWise alarms are listed below the **Alarm Data Streams** menu. Filter the alarm data streams listed by alarm metadata. **name**, **input property**, and **composite model ID** are some metadata used for filtering. Select one or more data streams depending on the selected widget.

- **KPI** and **Gauge** support only a single alarm.
- The remaining widgets support multiple alarms with multi-selection.

Modeled assets visualization

1. Drag the widget to the canvas. Select the properties for each widget panel to construct a dashboard.
2. The **Filter** option filters the assets to choose the asset to visualize. Filtering is done by text, property or value. Filtering is for assets loaded into the browser, and not backend filtering.
3. **Search** to list an asset to add to your widget.
4. **Add** the asset to the widget in the canvas.
5. Choose **Reset** to select another asset, or make modifications to the asset chosen.
6. Save the dashboard. In the **Preview** mode, choose different assets from the drop down menu to monitor the properties under each asset without reconstructing the data panels.

> **Note**
>
> The configuration settings wheel on the right hand side displays **Preferences** for the user to choose like **Page size**, **Sticky first columns**, **Sticky last columns**, and **Column preferences**. Customize your preferences, and choose **Confirm** to apply the changes.



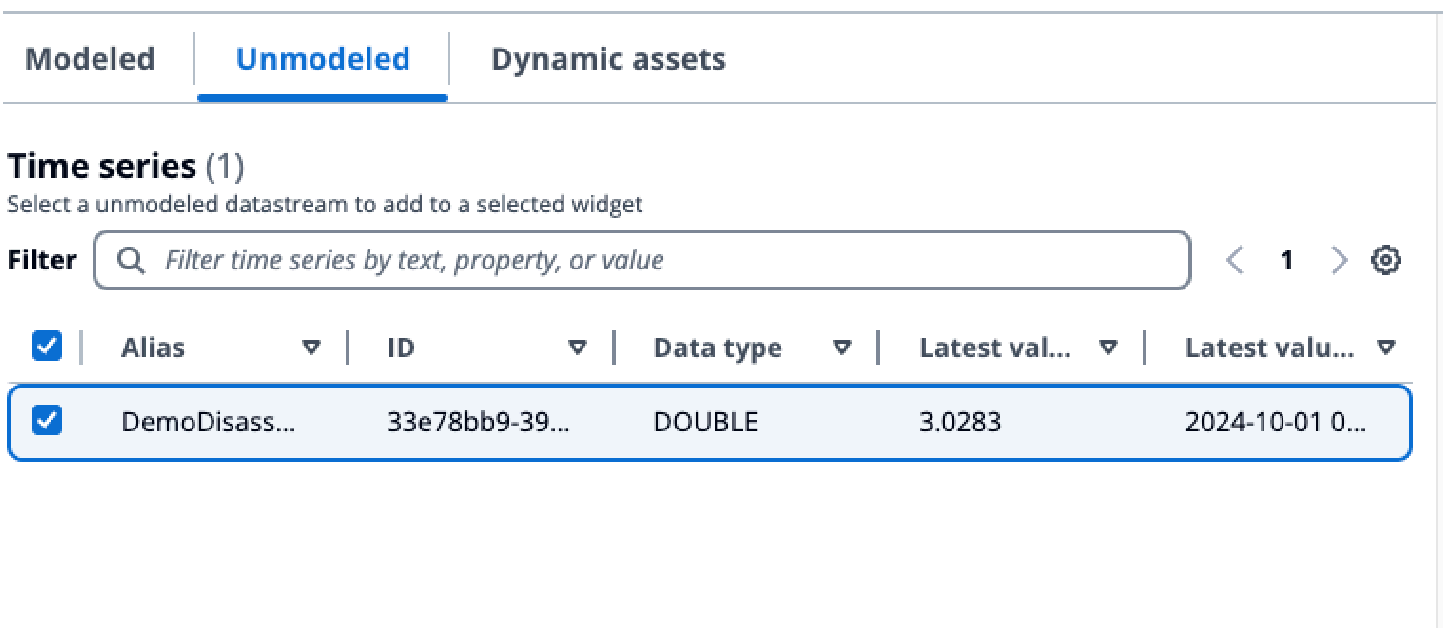
Unmodeled

This section describes searching for unmodeled data streams and adding them to the widgets to visualize.

Unmodeled data streams visualization

1. Drag the widget to the canvas. Select the properties for each widget panel to construct a dashboard.
2. Unmodeled data streams are listed under the **Time series** section. They have properties that are customizable.
3. The **Filter** option filters the data streams to visualize. Filtering is for data streams loaded into the browser, and not back end filtering.
4. **Add** the data stream to the widget in the canvas.
5. Choose **Reset** to deselect the data stream.
6. Save the dashboard. In the **Preview** mode, choose different assets from the drop down menu to monitor the properties under each asset without reconstructing the data panels.

> **Note**
>
> The configuration settings wheel on the right hand side displays **Preferences** for the user to choose like **Page size**, **Sticky first columns**, **Sticky last columns**, and **Column preferences**. Customize your preferences, and choose **Confirm** to apply the changes.

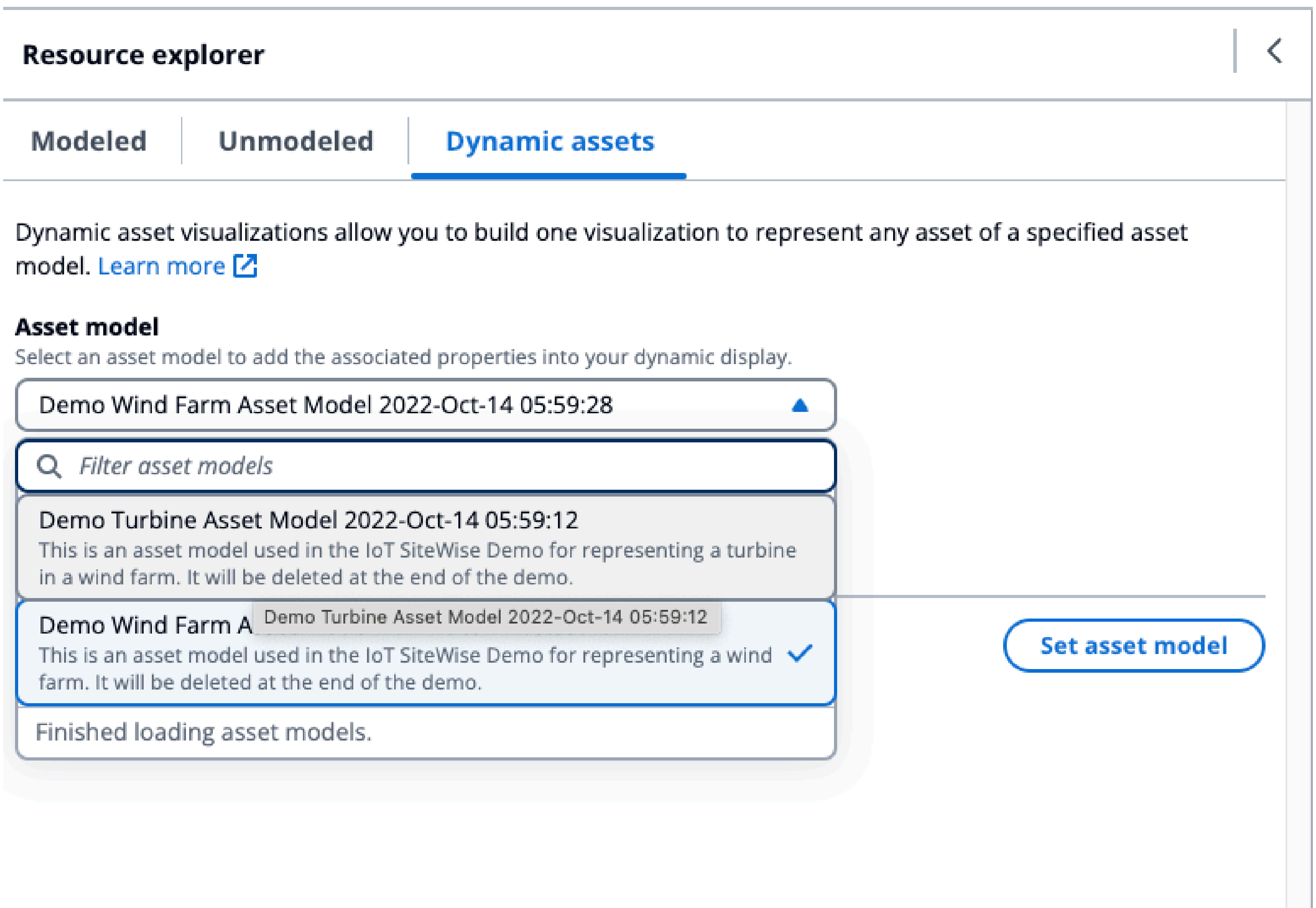


Dynamic assets

The new SiteWise Monitor allows customers to dynamically switch assets for a selected asset model. You can visualize properties from different assets by selecting from a drop down menu.

Dynamic assets visualization

1. Choose the **Dynamic assets** tab on the resource explorer.
2. Select an **Asset model** to list assets for from the drop down menu.
3. Select the **Default asset** from the drop down menu.
4. Choose **Set asset model** to select the asset model.
5. Save the dashboard. In the **Preview** mode, choose different assets from the drop down menu to monitor the properties under each asset, without reconstructing the data panels.



Widgets

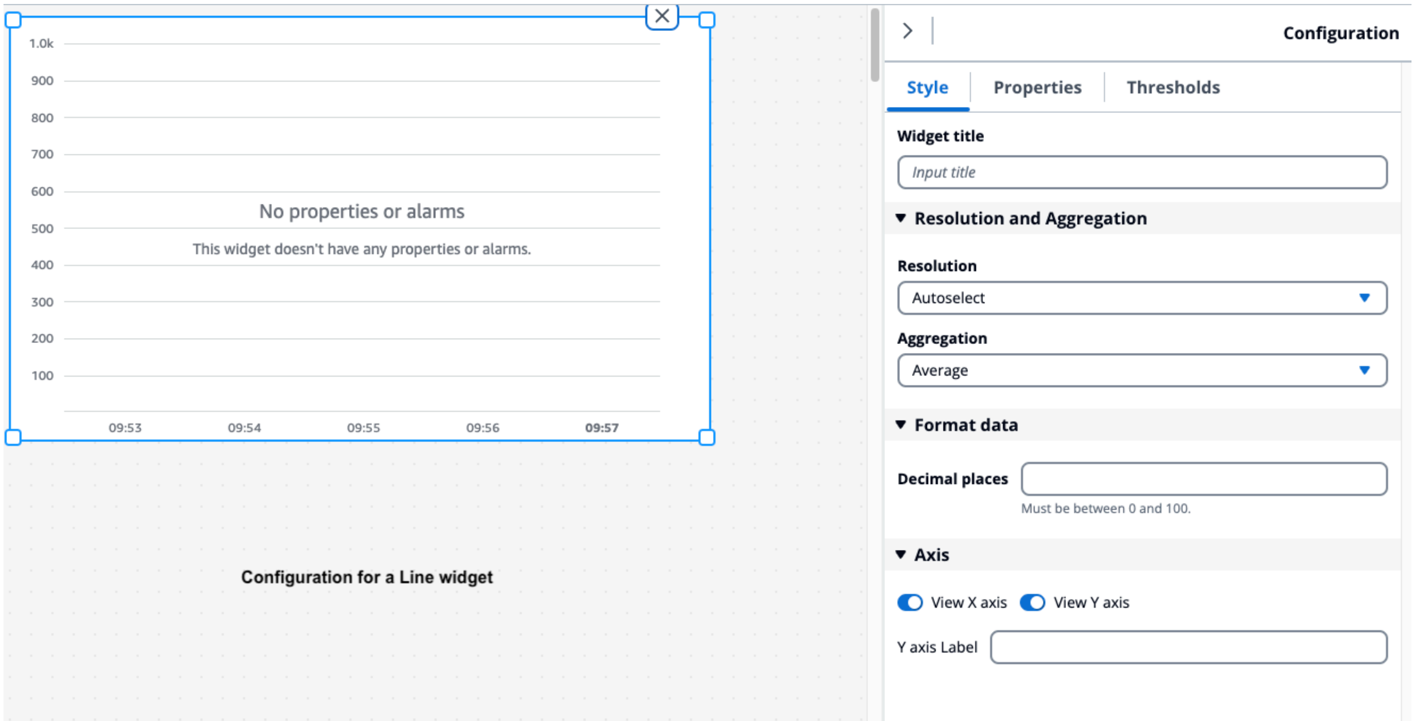
Widgets supports a wide range of features, including alarms, high-performance live-streaming, and smooth synchronization with other IoT App Kit components. The dashboard supports the following widgets:

- **Line –** The Line widget is a visualization widget that displays trends and changes over time. It consists of a series of data points, each represented by a dot or marker, connected by straight line segments to create a line graph. It supports a wide range of features, including alarms, thresholds, high-performance live-streaming, and smooth synchronization with other IoT App Kit components. This widget is customizable to communicate complex data clearly and concisely.
- **Bar chart –** The Bar chart is a powerful visualization tool that displays time-series data. It supports a wide range of features, including alarms, high-performance live-streaming, and smooth synchronization with other IoT App Kit components.
- **Timeline –** The Timeline widget provides a way to visualize and navigate time series data from data sources. It is unique for displaying data stream values are distinct colors on the timeline. It supports a rich set of features including alarms, high performance live-streaming and smooth syncing across other IoT App Kit components. It is best used for displaying non-numerical data types/
- **KPI –** The Key Performance Indicator (KPI) component provides a compact representation of an overview of your asset properties. It supports alarms and thresholds. This overview provides critical insights into the overall performance of your devices, equipment, and processes. KPI only supports a single data stream or alarm, and not multiple data streams.
- **Gauge –** The Gauge component provides a compact representation of an overview of your asset properties. It is used to visualize critical insights into the overall performance of your devices, equipment, or processes. It is functionally the same as KPI, but visually different. Gauge displays the data stream value, threshold, and range of values. You can interact with AWS IoT data from one or more data sources with Gauge.
- **Table –** The Table component provides a compact form for viewing one or more data streams from one or more time series data sources. It displays assets with **Property**, **Latest value** and **Unit** in a tabular form. Supports AWS IoT SiteWise alarms.
- **Text –** The Text widget helps write text with various colors and fonts. You can create a link by associating a text with an URL. The **Properties** and **Thresholds** fields are not enabled for this widget.

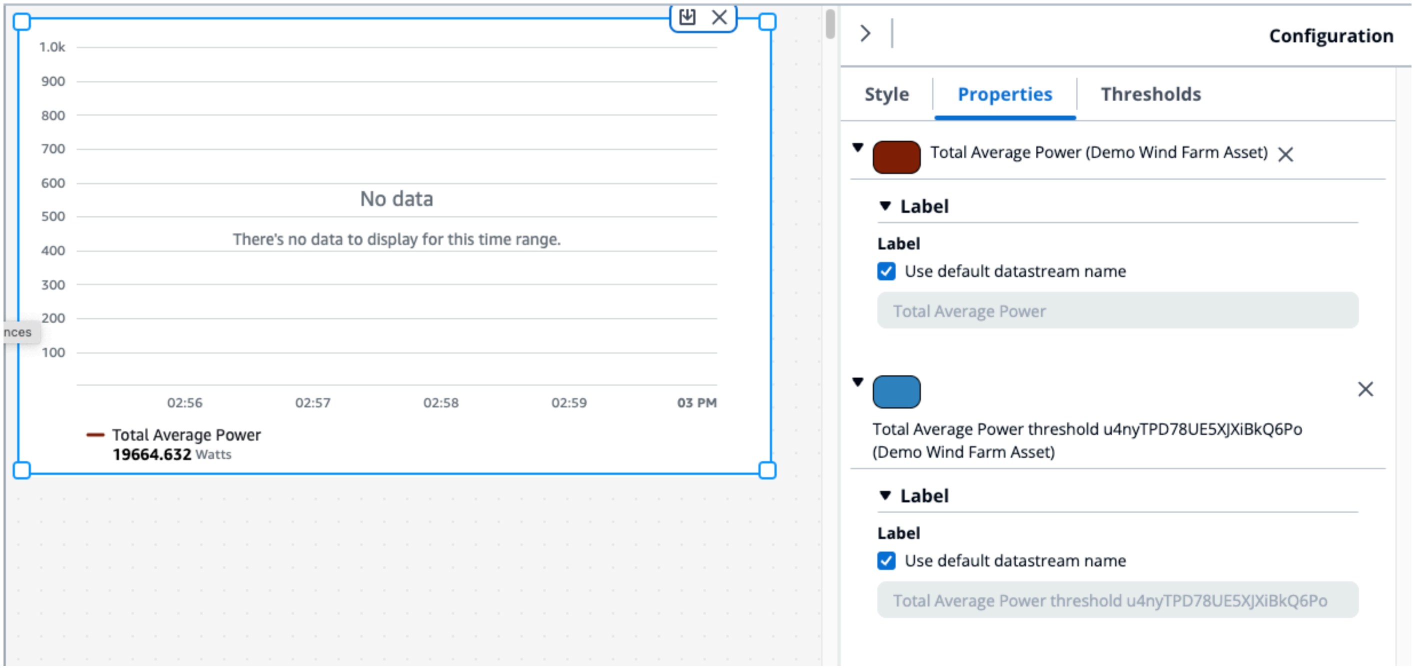
Configure widgets

Once the widget is added to the dashboard, you can configure the widget by choosing the **Configuration** icon in the right panel.

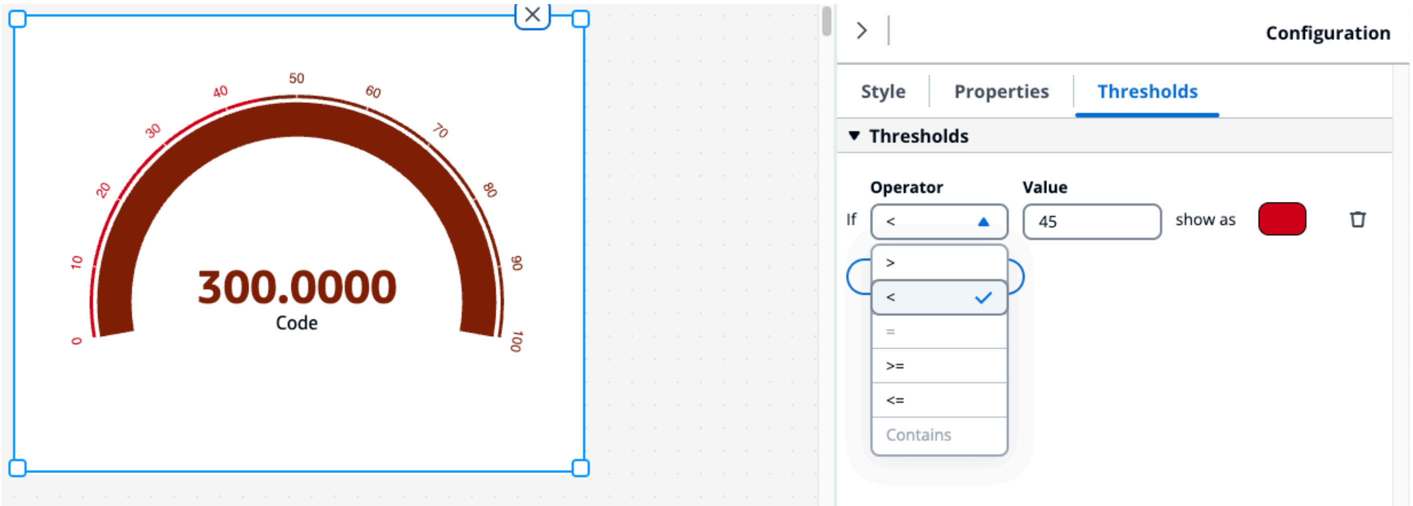
- **Style** – Add a title in the **Widget title**. Different widgets have different configurations. A few examples are listed below.
  - **Bar** widget :
    - **Resolution and Aggregation** – Set values for resolution and aggregation here.
    - **Format data** – Set **Decimal places** to the number of decimals to display.
    - **Display style** – Select values to display.
    - **Axis** – Choose to display the axis.
  - **Line** widget :
    - **Resolution and Aggregation** – Set values for resolution and aggregation here.
    - **Format data** – Set **Decimal places** to the number of decimals to display.
    - **Y-axis** – Add a **Label**, and **Min** and **Max**values.
    - **Widget style** – Select **Line type**, **Line style**, **Line thickness**, and **Data point shape**values.
    - **Legend** – Choose **Alignment**, and **Display**.
  - **Gauge** widget :
    - **Resolution and Aggregation** – Set values for resolution and aggregation here.
    - **Format data** – Set **Decimal places** to the number of decimals to display.
    - **Display style** – Select values to display.
    - **Y-axis** – Add a **Label**, and **Min** and **Max** values.
    - **Fonts** – Select **Font size**, **Unit font size**, and **Label font size** values.



- **Properties –** All the properties of widgets are listed in this section. Different widgets have different properties. A few examples are listed below.
  - **Line** widget :
    - **Label –** Choose to use the default datastream name or give a new name.
    - **Style** – Set **Line type**, **Line style** to the number of decimals to display.
    - **Y-axis –** Select values to default style, show Y-axis controls and set the **Min** and **Max** values.
  - **Table** widget :
    - **Label –** Choose to use the default datastream name or give a new name.
  - **Table** widget :
    - **Label –** Choose to use the default datastream name or give a new name.



- **Thresholds –** Add a **Threshold** for a widget. Different widgets have different configurations. A few examples are listed below.
  - **Bar chart** widget :
    - Choose **Add a threshold** to add to the widget.
    - Choose **Operator** and give a **Value** for the threshold. Customize the threshold with a color from the color palette.
    - You can choose to apply the threshold across all data.
  - **Line** widget :
    - Choose **Add a threshold** to add to the widget.
    - Choose **Operator** and give a **Value** for the threshold. Customize the threshold with a color from the color palette.
    - Choose how to **Show thresholds** from the drop down menu.
  - **Gauge** widget :
    - Choose **Add a threshold** to add to the widget.
    - Choose **Operator** and give a **Value** for the threshold. Customize the threshold with a color from the color palette.



Use widgets

You can use widgets in the dashboard individually or by multi-selecting them.

Edit widgets in the dashboard

Choose a single widget and edit it. To edit multiple widgets in the dashboard, **Shift + Left-click** and select all the widgets in the dashboard. Once selected, users can add new data-streams, and modify **Widget title** in the **Style** configuration settings. The title is changed for all the widgets in the dashboard.

Right-click on the canvas, and do the following:

- **Copy** – Add a copy of the widget to the canvas.
- **Delete** – Delete the widget.
- **Bring to front** – Bring the selected widget to the front of the canvas.
- **Send to back** – Send the selected widget to the back of the canvas.

Resize widgets

Re-size widgets individually, or in a group by multi-selecting the widgets in the dashboard.

To change the size of widgets:

- To change the size of a single widget, select the widget, and drag it by a corner to change its size.

- To change the size of multiple widgets, select multiple widgets by **Shift + Left-click**, and drag it by a corner to change its size.

Delete widgets in the dashboard

Delete widgets individually, or in a group by multi-selecting the widgets in the dashboard.

To delete widgets:

- To delete a single widget, select the widget, and **Right-click** and choose **Delete**. You can also select, and click **X** on the right hand top corner to delete the widget.
- To delete multiple widgets, select multiple widgets by **Shift + Left-click**, then **Right-click** and choose **Delete**.

Alarms in widgets

Alarms alert you and 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 that they can fix the issue. Alarms help you quickly identify issues and notify operators to maximize performance of your equipment and processes.

You can find an alarm associated with an asset in the **Modeled** tab of the **Resource explorer**.

- Search for and select an asset.
- Scroll down past the **Data Streams** table to the **Alarm Data Streams** section and expand it.
- Select an alarm in the **Alarms** table and choose **Add**.

Topics

Alarms in different widgets

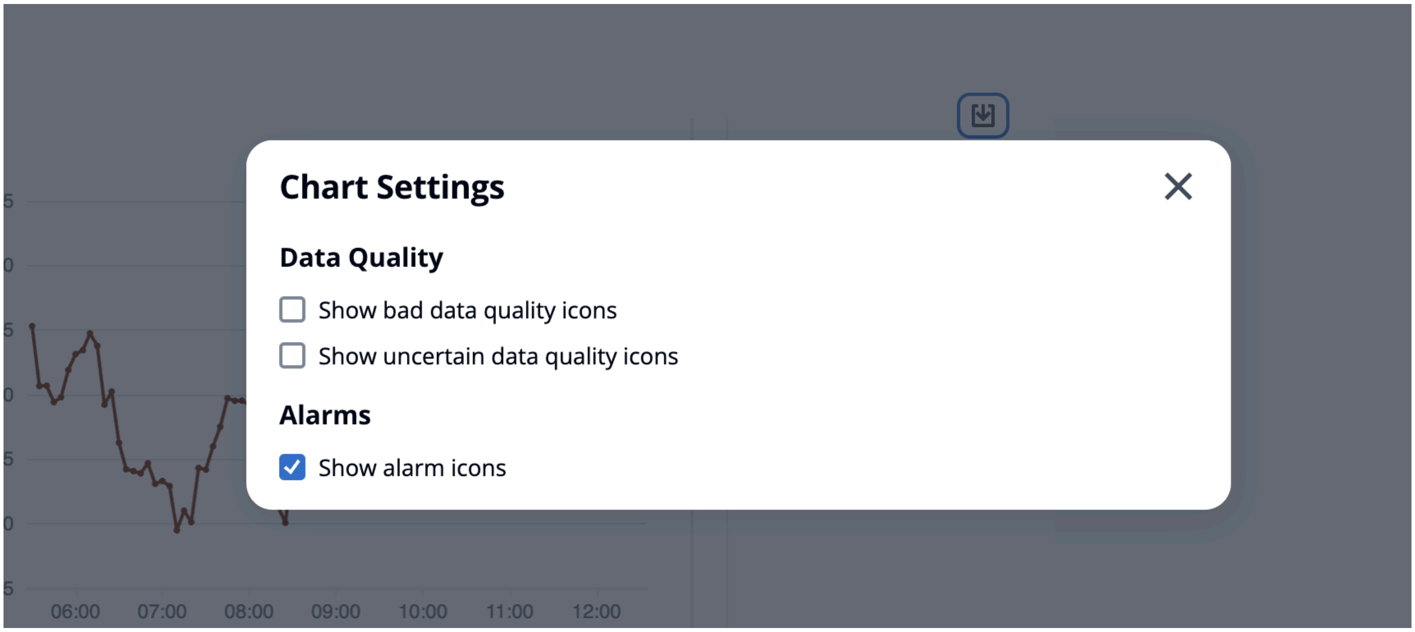
For all widgets:

- Data stream property settings are dependent on what type of property is added to a widget. Data stream properties have full property setting support while alarm properties do not currently allow property setting configurations.

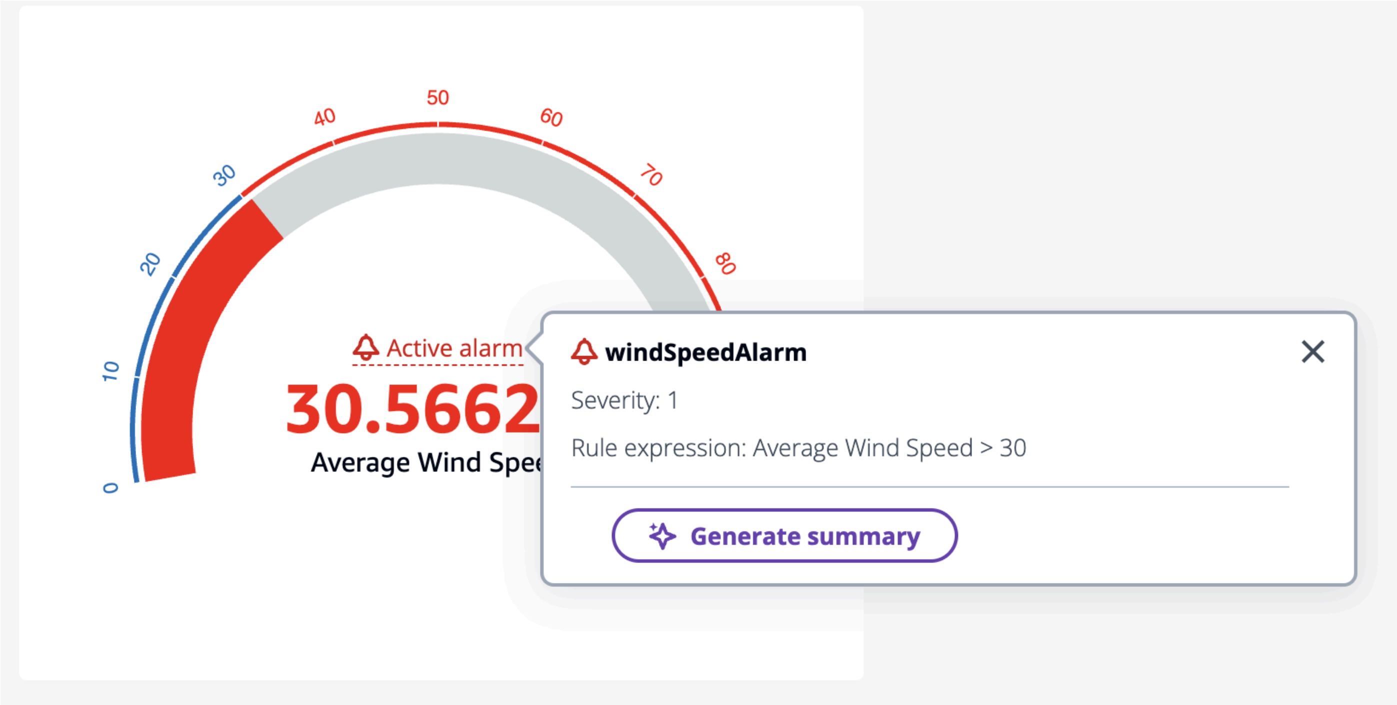
- If you add an alarm data stream, its associated input property data stream is added to the chart as well. If you remove the alarm data stream, then its input property is also removed.
- To individually control the input property data stream of an alarm, you must **add them both** separately.

The below examples state how some widgets use alarms.

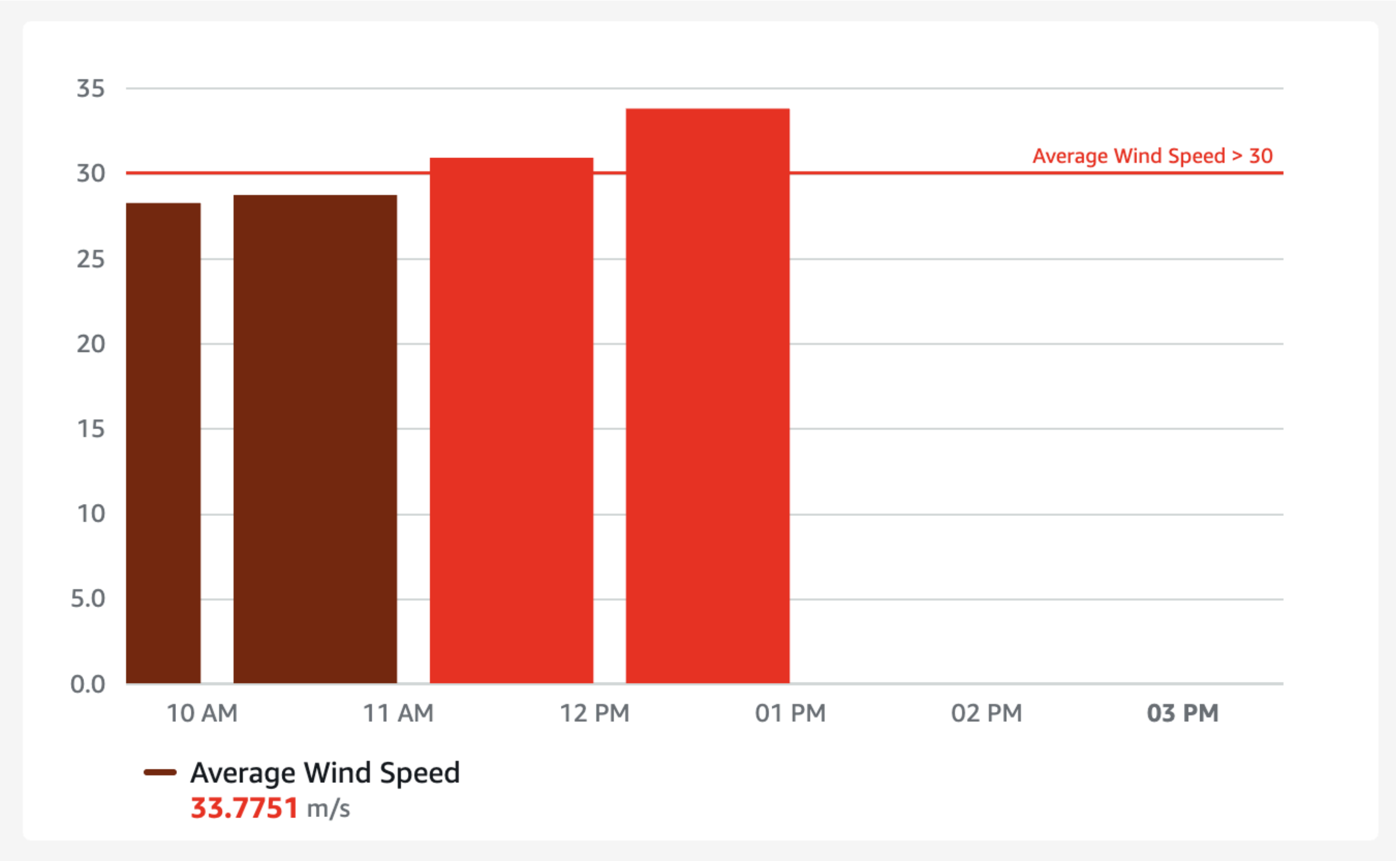
- **Line Chart**
  - The alarm and its input property data stream are added to the chart.
  - You can see alarm state in the chart legend, and as icons hovering over the data stream when the alarm changes state.
  - You can toggle off alarm icons from the chart settings.



- **KPI** and **Gauge**
  - The alarm and its input property data stream are added to the chosen widget.
  - The alarm threshold is added to the widget, which changes color based on its configuration.
  - You can select the alarm state on the widget, see the alarm details, and click **Generate summary** to call the AWS IoT SiteWise to get an alarm summary.



- **Table**
  - The alarm and its input property are added as a row on the table.
- **Bar chart**
  - The alarm is added as a threshold to the chart, which changes the color of any data stream breaching the threshold.
  - You can add any associated data streams separately.
  - You cannot interact with the AWS IoT SiteWise Assistant from the widget.

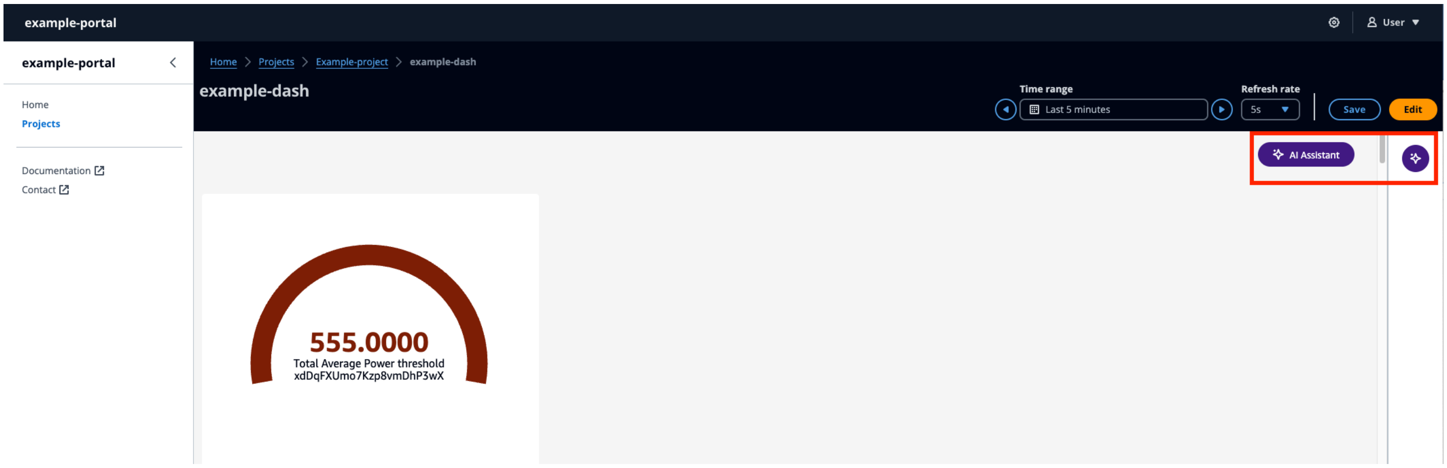


- **Status timeline**
  - The alarm is added as a threshold to the timeline.
  - Adding the alarm state and its input property data to the timeline is work in progress.
  - You cannot interact with the AWS IoT SiteWise Assistant from the widget.

AWS IoT SiteWise Assistant use in widgets

The AWS IoT SiteWise Assistant is a generative AI-powered assistant. It allows users like plant managers, quality engineers, and maintenance technicians to gain insights, solve problems, and take actions directly from their operational and enterprise data. The AWS IoT SiteWise Assistant consolidates information from AWS IoT data, asset models, manuals and documentation into understandable summaries of critical events. It also enables interactive deep dive question and answer sessions for easy diagnosis, root cause explorations and guided recommendations.

The AWS IoT SiteWise Assistant button is on the top right corner of the dashboard. Click on it to activate the Assistant. Can only be used with the **Preview** mode of the dashboard.



Use the AWS IoT SiteWise Assistant in the following scenarios:

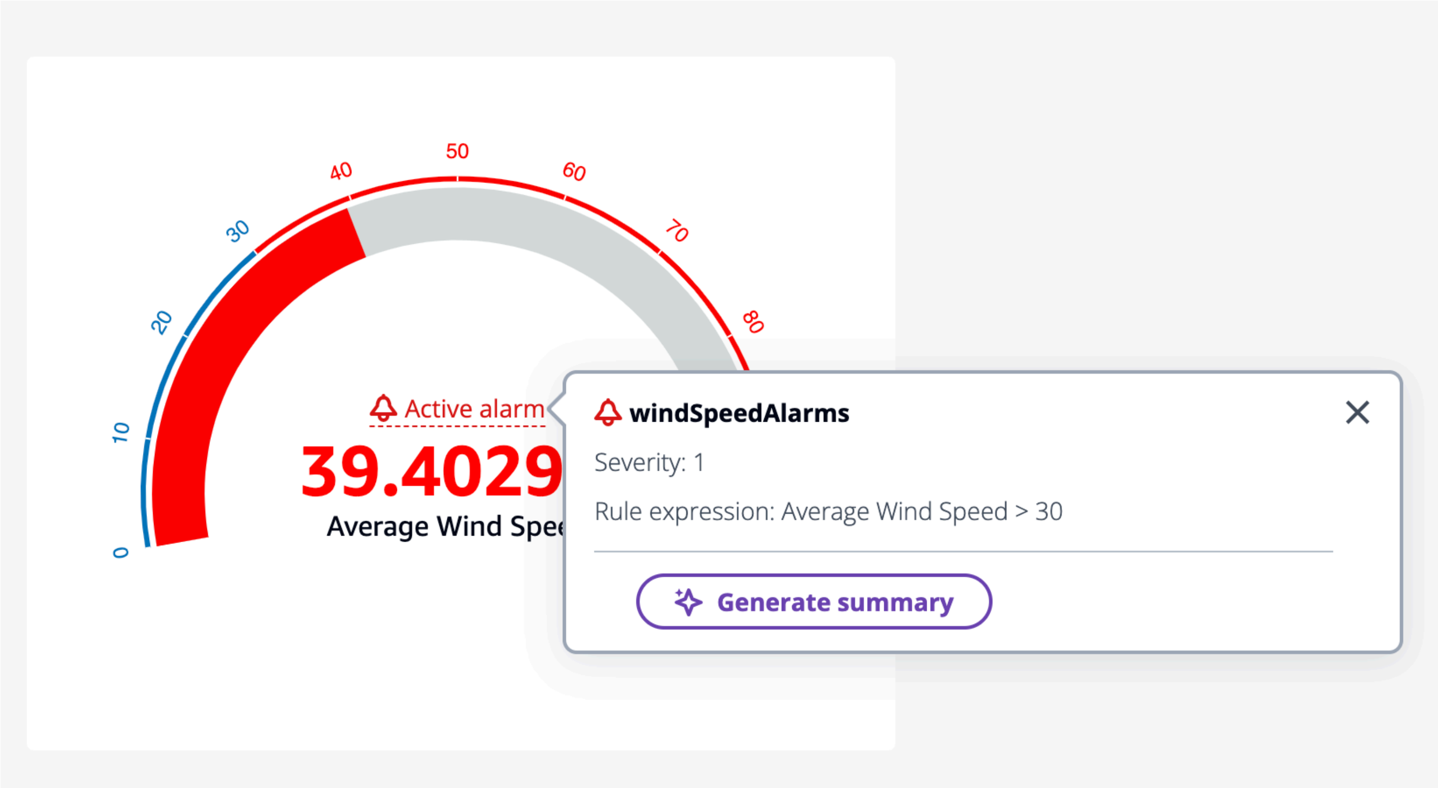
Topics

- Use case - Alarm summaries
- Use case - Situational summaries
- Use case - Deep dive summaries

Use case - Alarm summaries

Summarize the current alarm for a selected panel on the dashboard. Alarms are supported by **Line, KPI, Gauge** and **Table** widgets. Choose a widget with an alarm and summarize it.

- Select **Active alarm** on widget.
- The **Severity** and **Rule expression** is displayed for the alarm.
- Choose **Generate summary** to generate a summary.



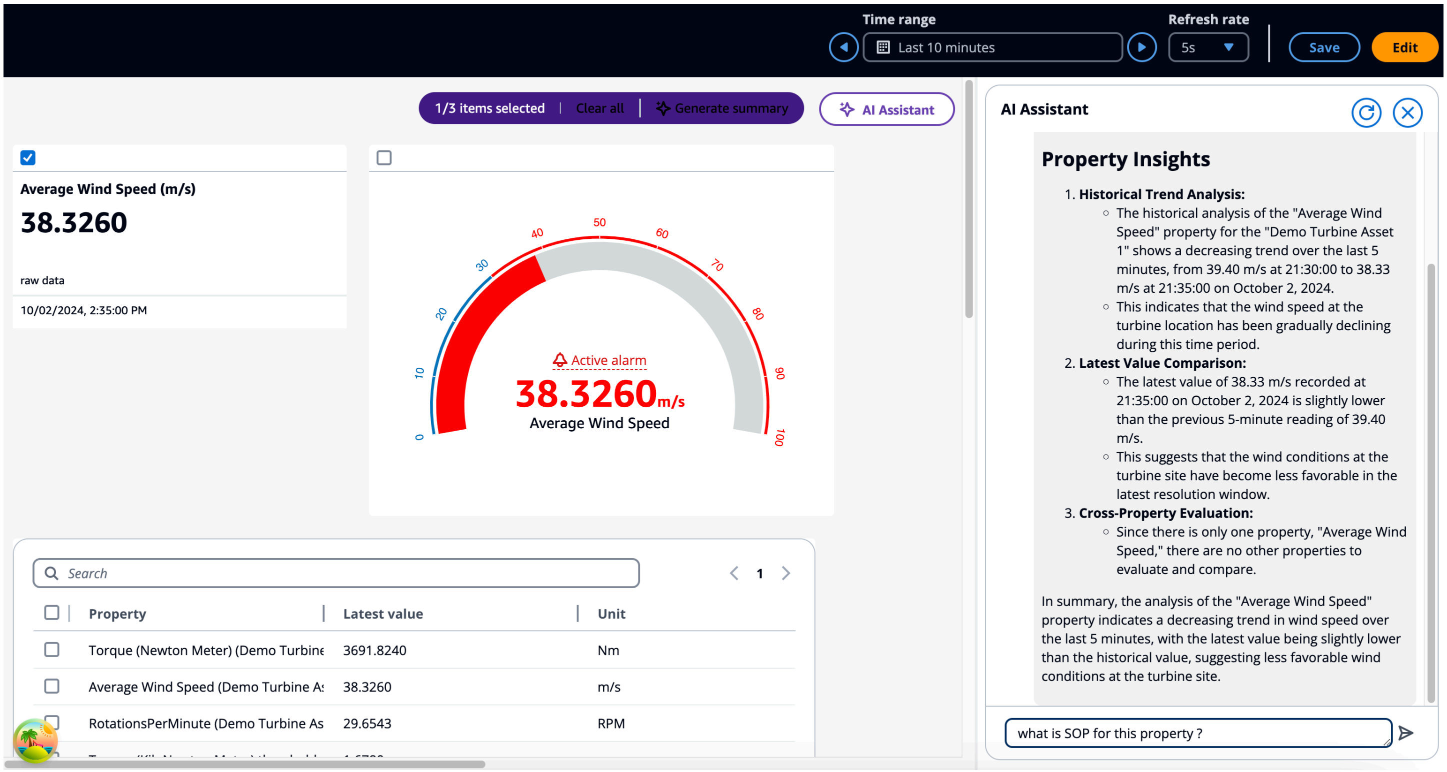
Use case - Situational summaries

Select up to three widgets to summarize. They can be a combination of widgets and properties. If more than three are selected, the Assistant returns an error.

Generate a situation summary with AWS IoT SiteWise Assistant

1. Click on **AI Assistant**. It displays a menu with three options.
   a. **Items selected** – Select only three. You cannot select more than three.
   b. **Clear all** – Clear your selection.
   c. **Generate summary** – Generate a summary about items selected.
2. Choose **Generate summary** to generate the summary about the selected items.

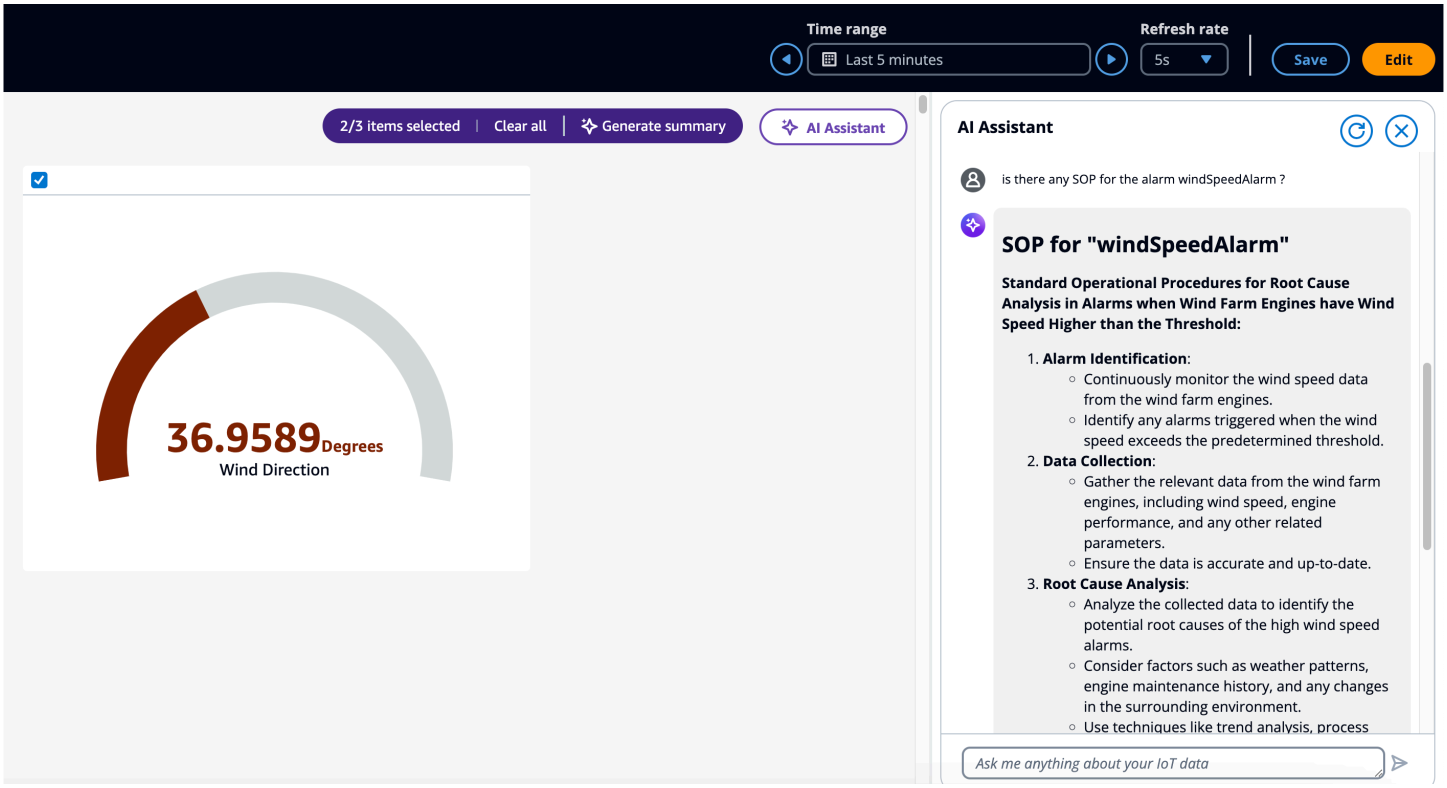
The image below has a widget selected and a summary from the AWS IoT SiteWise Assistant.



Use case - Deep dive summaries

This is the use case where the user can do a deep dive, and access SOPs (Standard Operation Procedure), manuals, documentation, and consider next steps of action. For the example in the previous section, if the user chooses to learn more about the SOP for this property, ask the Assistant about the SOP for this property. This displays the deep dive information about SOP to the user.

The example below displays the answer for "**Is there any SOP for the alarm windSpeedAlarm?**"



Sample questions to ask AWS IoT SiteWise Assistant

> **Note**
>
> - The AWS IoT SiteWise Assistant must use a dataset with an Amazon Kendra index for enterprise level knowledge and guidance. If you do not have a Amazon Kendra index, see Creating an index to create one. Adding a dataset improves the quality of the Assistant's response. See Create a dataset to learn more.
> - Some questions require AWS IoT TwinMaker integration. See Integrate AWS IoT TwinMaker and AWS IoT SiteWise for details.

Some follow up questions to ask the Assistant after getting an alarm summary in the dashboard, as part of the same conversation.

- Show the details of the asset from the summary above?
- What is the hierarchical path from root to the mentioned asset?
- What are the dependent descendant assets of the mentioned asset?
- What are the dependent assets of the mentioned asset that have active alarms?

- Find all assets that have active alarms.

Some follow up questions to ask the Assistant after getting an property summary in the dashboard, as part of the same conversation.

- Perform the same analysis for the last 24 hours.
- Find documentation related to the above mentioned properties.
- Provide the details of asset id 1da67d28-14f8-4f71-a06a-386f0425a21d/asset name Demo Turbine Asset 1.

Invoke the AWS IoT SiteWise Assistant from the API.

- Generate alarm summary for alarm name **windSpeedAlarm** in asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d`.
- Generate alarm summary for the last 12 hours/2 days/1 week for alarm name **windSpeedAlarm** in asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d`.
- Generate property summary for property id `ab187fb7-d74b-44d9-bd9b-f2f19a9137cc` in asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d`
- Generate property summary for the last 12 hours/2 days/1 week for property id `ab187fb7-d74b-44d9-bd9b-f2f19a9137cc` in asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d`.
- Find the assets with asset name Turbine.
- Give me the current property values of property id `5356168c-3390-456f-802c-9f6e047810d4` in asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d`, `3cbb084e-1ded-4b08-9f21-1b47b2fb86fd`.
- What is the relationship between asset id `d591e153-e5cf-4206-96bb-ce3c119d9d2d` and asset id `3cbb084e-1ded-4b08-9f21-1b47b2fb86fd`.
- Find documentation on how to fix wind turbine low RPM issue.
- Generate a property summary for property alias **WindSpeed**.
- What are the pre-operation checks according to my knowledge base?

Query data from AWS IoT SiteWise

You can use the AWS IoT SiteWise API operations to query your asset properties' current values, historical values, and aggregates over specific time intervals. AWS IoT SiteWise provides multiple query interfaces to meet different integration needs:

- **Direct API operations** - Simple, targeted API calls for specific data retrieval needs
- **SQL-like query language** - Powerful, flexible queries for complex data analysis
- **ODBC driver** - Integration with business intelligence tools and applications

Use these query capabilities to:

- Gain real-time insights into operational data
- Analyze historical trends and patterns
- Calculate performance metrics across your industrial assets
- Integrate IoT data with enterprise systems and dashboards
- Build custom applications that leverage industrial data

For example, you can discover all assets with specific property values, build custom representations of your data, 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 Monitor data with AWS IoT SiteWise Monitor.

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`
- `nullValue`

Topics

Query current asset property values in AWS IoT SiteWise

This tutorial shows two ways to get the current value of an asset property. You can use the AWS IoT SiteWise console or use API in the AWS Command Line Interface (AWS CLI).

Topics

Query 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.
4. Choose the arrow icon to expand an asset hierarchy to find your asset.
5. Choose the tab for the type of property. For example, choose **Measurements** to view the current value of a measurement property.

6. Find the property to view. The current value appears in the **Latest value** column.

Query an asset property's current value (AWS 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, specify one of the following:

- The `assetId` and `propertyId` of the asset property that data is sent 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 set property aliases, see Manage data streams for AWS IoT SiteWise.

To get the current value of an asset property (AWS 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.

  ```
  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.

  ```
  {
    "propertyValue": {
      "value": {
        "booleanValue": Boolean,
        "doubleValue": Number,
        "integerValue": Number,
        "stringValue": "String",
        "nullValue": {
            "valueType": "String"
        }
      },
      "timestamp": {
  ```

```
      "timeInSeconds": Number,
      "offsetInNanos": Number
    },
    "quality": "String"
  }
}
```

Query historical asset property values in AWS IoT SiteWise

You can use the AWS IoT SiteWise API [GetAssetPropertyValueHistory]() operation to query the historical values of an asset property.

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

- The `assetId` and `propertyId` of the asset property that data is sent 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 set property aliases, see [Manage data streams for AWS IoT SiteWise]().

Pass 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 (AWS 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.

```
aws iotsitewise get-asset-property-value-history \
  --asset-id asset-id \
  --property-id property-id \
  --start-date 1575216000 \
  --end-date 1575216600
```

The operation returns a response that contains the historical TQVs of the property in the following format:

```
{
  "assetPropertyValueHistory": [
    {
      "value": {
        "booleanValue": Boolean,
        "doubleValue": Number,
        "integerValue": Number,
        "stringValue": "String",
        "nullValue": {
            "valueType": "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.

Query asset property aggregates in AWS IoT SiteWise

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. With metric properties, you define aggregations that are 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 Aggregate data from properties and other assets (metrics).

Topics

Aggregates for an asset property (API)

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, specify one of the following:

- The `assetId` and `propertyId` of the asset property that data is sent 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 set property aliases, see Manage data streams for AWS IoT SiteWise.

You must 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), `15m` (15 minutes), `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.

Aggregates for an asset property (AWS CLI)

To query aggregates for an asset property (AWS 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.

```
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.

```
{
  "aggregatedValues": [
    {
      "timestamp": Number,
      "quality": "String",
      "value": {
        "average": Number,
        "count": Number,
        "maximum": Number,
        "minimum": Number,
        "standardDeviation": Number,
        "sum": Number
      }
    }
  ],
  "nextToken": "String"
}
```

2. If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to the GetAssetPropertyAggregates operation.

Note

If your query range contains a `null` value TQVs, see AssetPropertyValue API. All statistics except count, results in a `null` response, similar to statistics for String TQVs. If your query range contains `Double.NaN` for double type TQVs, all calculations except count will result in a `Double.NaN`.

AWS IoT SiteWise query language

With the AWS IoT SiteWise data retrieval ExecuteQuery API operation, you can retrieve information about declarative structural definitions, and the timeseries data associated with them, from the following:

- models
- assets
- measurements
- metrics
- transforms
- aggregates

This can be done with SQL like query statements, in a single API request.

> **Note**
>
> This feature is available in all Regions where AWS IoT SiteWise is available, except AWS GovCloud (US-West), Canada (Central), China (Beijing) and US East (Ohio).

Topics

Query language reference for AWS IoT SiteWise

AWS IoT SiteWise supports a rich query language for working with your data. The available data types, operators, functions and constructs are described in the following topics.

See Example queries to write queries with the AWS IoT SiteWise query language.

Topics

Query reference views

This section provides information to help you understand the views in AWS IoT SiteWise, such as process metadata and telemetry data.

The following tables provide the view names and descriptions of the views:

Data model

View name	View description
asset	Contains information about the asset and model derivation.
asset_property	Contains information about the asset property's structure.
raw_time_series	Contains the historical data of the time series.
latest_value_time_series	Contains the latest value of the time series.
precomputed_aggregates	Contains the automatically computed aggregated asset property values. They are a set of basic metrics calculated over multiple time intervals.

The following views list the column names and data types of each view.

View:asset

column name	datatype
asset_id	string
asset_name	string
asset_description	string
asset_model_id	string
parent_asset_id	string

column name	datatype
asset_external_id	string
asset_model_external_id	string
hierarchy_id	string

View:asset_property

column name	datatype			
asset_id	string			
property_id	string			
property_name	string			
property_alias	string			
property_external_id	string			
asset_composite_model_id	string			
property_type	string			
property_data_type	string			
int_attribute_value	integer			
double_attribute_value	double			
boolean_attribute_value	boolean			

column name	datatype
string_at tribute_value	string

View:raw_time_series

column name	datatype
asset_id	string
property_id	string
property_alias	string
event_timestamp	timestamp
quality	string
boolean_value	boolean
int_value	integer
double_value	double
string_value	string

View:latest_value_time_series

column name	datatype
asset_id	string
property_id	string
property_alias	string
event_timestamp	timestamp
quality	string

column name	datatype
boolean_value	boolean
int_value	integer
double_value	double
string_value	string

View:precomputed_aggregates

column name	datatype
asset_id	string
property_id	string
property_alias	string
event_timestamp	timestamp
quality	string
resolution	string
sum_value	double
count_value	integer
average_value	double
maximum_value	double
minimum_value	double
stdev_value	double

Supported data types

AWS IoT SiteWise query language supports the following data types.

Scalar value

Data type	Description
`STRING`	A string of maximum length 1024 bytes.
`INTEGER`	A signed 32-bit integer with a range from `-2,147,483,648 to 2,147,483,647` .
`DOUBLE`	A floating point number with range from –`10^100 to 10^100`, or Nan with `IEEE 754` double precision.
`BOOLEAN`	`true` or `false`.
`TIMESTAMP`	ISO-8601 compliant timestamps: • '`yyyy-MM-dd HH:mm:ss[.SSS]` ' • `TIMESTAMP` '`yyyy-MM-dd[\s|T]HH:mm:ss[.SSS][+HH:mm|'Z']` ' • '`yyyy-MM-dd'T'HH:mm:ss[.SSS][+HH:mm|'Z']` ' • '`yyyy-MM-dd'T'HH:mm:ss+[hh:mm]` '

Note

`Null`: A boolean `true` indicating a lack of defined data.

Example

`TIMESTAMP` value examples:

```
TIMESTAMP '2025-12-21 23:59:59.999Z'
TIMESTAMP '2025-12-21 23:59:59+23:59'
```

```
'2025-12-21 23:59:59'
'2025-12-21T23:59:59.123+11:11'
```

Note

The double precision data is not exact. Some values are not converted exactly, and will not represent all real numbers due to limited precision. Floating-point data in the query may not be the same value represented internally. The value is rounded if the precision of an input number is too high.

Supported clauses

The `SELECT` statement is used to retrieve data from one or more views. AWS IoT SiteWise supports the `JOIN` and `INNER JOIN` operations.

Views are joined with an explicit `JOIN` syntax, or with comma-separated notations in the `FROM` clause.

Example

A general `SELECT` statement:

```
SELECT expression [, ...]
  [ FROM table_name AS alias [, ...] ]
  [ WHERE condition ]
  [ GROUP BY expression [, ...] ]
  [ HAVING condition ]
  [ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ] [, ...] ]
  [ LIMIT expression ]
```

Example

A SELECT statement with the different clauses:

```
SELECT
  a.asset_name,
  a.asset_id,
  p.property_type,
  p.property_data_type,
  p.string_attribute_value,
  p.property_name
```

```
FROM asset a, asset_property p
WHERE a.asset_description LIKE '%description%'
AND p.property_type IN ('attribute', 'metric')
OR p.property_id IN (
  SELECT property_id
  FROM raw_time_series
  WHERE event_timestamp BETWEEN TIMESTAMP '2025-01-01 00:00:00' AND TIMESTAMP
 '2025-01-02 00:00:00'
  GROUP BY asset_id, property_id
  HAVING COUNT(*) > 100
 )
GROUP BY p.property_type
HAVING COUNT(*) > 5
ORDER BY a.asset_name ASC
LIMIT 20;
```

Note

An implicit JOIN combines two or more different tables without using the JOIN keyword based on AWS IoT SiteWise's internal schema. This is the equivalent of performing a JOIN on the asset_id and property_id fields between metadata and raw data tables. This pattern allows SiteWise to leverage any given metadata filters in the query, when fetching from raw data tables in a way that results in less overall data scanned.

Example of a query:

```
SELECT a.asset_name, p.property_name, r.event_timestamp
FROM asset a, asset_property p, raw_time_series r
WHERE a.asset_name='my_asset' AND p.property_name='my_property'
```

The above example only scans data from the asset property belonging to the specified metadata names.

Example of a less optimized equivalent of the above query:

```
SELECT a.asset_name, p.property_name, r.event_timestamp
FROM asset a
JOIN asset_property p ON a.asset_id=p.asset_id
JOIN raw_time_series r ON p.asset_id=r.asset_id AND p.property_id=r.property_id
WHERE a.asset_name='my_asset' AND p.property_name='my_property'
```

An explanation of each clause and it's description is listed below:

Clause	Signature	Description
`LIMIT`	`LIMIT { count }`	This clause limits the result set to the specified number of rows. You can use `LIMIT` with or without `ORDER BY` and `OFFSET` clauses. `LIMIT` only works with non-negative integers from [0,2147483647].
`ORDER BY`	`ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]`	The `ORDER BY` clause sorts the result set of a query.
`GROUP BY`	`GROUP BY expression [, ...]`	The `GROUP BY` clause identifies the grouping columns for the query. It is used in conjunction with an aggregate expression.
`HAVING`	`HAVING boolean-expression`	The `HAVING` clause filters group rows created by the GROUP BY clause.
`SUB SELECT`	`SELECT column1, column2 FROM table1 WHERE column3 IN (SELECT column4 FROM table2);`	A `SELECT` statement embedded within another `SELECT` statement.
`JOIN`	`SELECT column1, column2 FROM table1 JOIN table2`	

Clause	Signature	Description
	`ON table1.column1 = table2.column1;`	
`INNER JOIN`	`SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.column;`	An `INNER JOIN` returns all rows from both tables, that match the join condition.
`UNION`	`query { UNION [ ALL ] } another_query`	The `UNION` operator computes the set union of its two arguments, automatically removing duplicate records from the result set.

Logical operators

AWS IoT SiteWise supports the following logical operators.

Operator	Signature	Description
`AND`	a `AND` b	`TRUE` if both values are true
`OR`	a `OR` b	`TRUE` if one value is true
`NOT`	`NOT` expression	`TRUE` if an expression is false, and `FALSE` if an expression is true
`IN`	x `IN` expression	`TRUE` if value in expression
`BETWEEN`	`BETWEEN` a `AND` b	`TRUE` if value between upper and lower limit, and includes both limits
`LIKE`	`LIKE` pattern	`TRUE` if value is in pattern

Operator	Signature	Description
		LIKE supports wildcards. See below for examples: • % substitutes one or more characters in a string. • _ substitutes one character in a string. • ESCAPE is used with a character to designate an escape character in the LIKE pattern.

Examples of all logical operators:

Function	Example
AND	`SELECT a.asset_name FROM asset AS a, latest_value_time_series AS t WHERE t.int_value > 30 AND t.event_timestamp > TIMESTAMP '2025-05-15 00:00:01'`
OR	`SELECT a.asset_name FROM asset AS a WHERE a.asset_name like 'abc' OR a.asset_name like 'pqr'`
NOT	`SELECT ma.asset_id AS a_id FROM asset AS ma WHERE (ma.asset_id NOT LIKE 'some%patterna%' escape 'a') AND ma.asset_id='abc'`

<table>
<tr><th>Function</th><th>Example</th></tr>
<tr><td>IN</td><td>
<pre><code>SELECT a.asset_name
   FROM asset AS a
   WHERE a.asset_name IN ('abc', 'pqr')</code></pre>
</td></tr>
<tr><td>BETWEEN</td><td>
<pre><code>SELECT asset_id, int_value, event_timestamp AS i_v
   FROM raw_time_series
   WHERE event_timestamp BETWEEN TIMESTAMP '2025-04-15 00:00:01' and TIMESTAMP '2025-05-15 00:00:01'</code></pre>
</td></tr>
<tr><td>LIKE</td><td>

% pattern:
<pre><code>SELECT POWER(rw.int_value, 5) AS raised_value
   FROM raw_time_series AS rw
   WHERE rw.asset_id LIKE 'some%pattern%' AND rw.int_value > 30</code></pre>

_ pattern:
<pre><code>SELECT asset_id, property_id
  FROM asset_property
  WHERE string_attribute_value LIKE 'Floor_'</code></pre>

ESCAPE pattern:
<pre><code>SELECT asset_id
  FROM asset
  WHERE asset_name LIKE 'MyAsset/_%' ESCAPE '/'</code></pre>

</td></tr>
</table>

Comparison operators

AWS IoT SiteWise supports the following comparison operators. All comparison operations are available for built in data types and evaluate to a boolean.

Logical operators

Operator	Description
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`=`	Equals
`!=`	Not equal

Comparison operation truth table for non numeric values

Type	Type >= x	Type <= x	Type > x	Type < x	Type = x	Type != x
`NULL`	`FALSE`	`FALSE`	`FALSE`	`FALSE`	`FALSE`	`TRUE`

Some predicates behave like operators but have special syntax. See below:

Comparision predicates

Operator	Description
`IS NULL`	Tests if a value is `NULL`.
`IS NOT NULL`	Tests if a value is not `NULL`.

NaN operators

NaN, or 'Not a Number', is a special value in floating-point arithmetic. Here is a list of NaN comparisions and how they work.

- NaN values must be enclosed within single quotes. For example, 'NaN'.
- NaN values are considered equal to each other.
- NaN is greater than other numeric values.
- In aggregate functions like AVG(), STDDEV(), and SUM(), if any values are NaN, the result is NaN.
- In aggregate functions like MAX() and MIN(), NaN values are included in the calculations.

NaN value comparisions

Comparison	Result
`'NaN' ≥ x`	True
`'NaN' ≤ x`	True if x equals NaN, False otherwise
`'NaN' > x`	False if x equals NaN, True otherwise
`'NaN' < x`	False
`'NaN' = x`	True if x equals NaN, False otherwise
`'NaN' != x`	False if x equals NaN, True otherwise

SQL functions

The function groups supported are:

Topics

- Scalar functions

- Aggregate functions

Scalar functions

Scalar functions take one or more input values and returns a single output value. They are widely used in SQL (Structured Query Language) for data manipulation and retrieval, improving the efficiency of data processing tasks.

Topics

- Null data functions
- String functions
- Math functions
- Date time functions
- Type conversion functions

Null data functions

Null data functions handle or manipulate NULL values, which represent the absence of a value. The functions allow you to replace NULLs with other values, check if a value is NULL, or perform operations that handle NULLs in a specific way.

Function	Signature	Description
`COALESCE`	COALESCE (expression1, expression2, ..., expressionN)	If all expressions evaluate to null, COALESCE returns null. Expressions must be of same type.

Example of a COALESCE function

```
SELECT COALESCE (l.double_value, 100) AS non_double_value FROM latest_value_time_series
 AS l LIMIT 1
```

String functions

String functions are built-in tools used to manipulate and process text data. They enable tasks like concatenation, extraction, formatting, and searching within strings. These functions are essential for cleaning, transforming, and analyzing text-based data within a database.

String functions

Function	Signature	Description
`LENGTH`	LENGTH (string)	Returns the length of the string.
`CONCAT`	CONCAT (string, string)	Concatenates arguments in a string.
`SUBSTR`	• SUBSTR (string, start) • SUBSTR (string, start, length) • SUBSTR (string, regexp)	Returns one of the following: • Returns the substring of the input string starting the specified location and optionally having the specified length. • Returns the first substring of the input string matching the specified regular expression. Uses 1-based indexing for start parameter.
`UPPER`	UPPER (string)	Converts the characters in the input string to uppercase.
`LOWER`	LOWER (string)	Converts the characters in the input string to lowercase.

Function	Signature	Description
`TRIM`	TRIM (string)	Removes any space character s from the beginning, end, or both sides of string.
`LTRIM`	LTRIM (string)	Removes any space characters from the beginning of string.
`RTRIM`	RTRIM (string)	Removes any space characters from the end of string.
`STR_REPLACE`	STR_REPLACE (string, from, to)	Replaces all occurrences of the specified substring with another specified substring.

Examples of all the functions:

Function	Example
LENGTH	`SELECT LENGTH(a.asset_id) AS asset_id_length FROM asset AS a`
CONCAT	`SELECT CONCAT(p.property_id, p.property_name) FROM asset_pro perty AS p`
SUBSTR	• `SELECT SUBSTR(a.asset_name, 1, 3) AS substr-val FROM asset AS a` • `SELECT SUBSTR(p.property_name, 3) AS substr_val1 FROM asset_pro perty AS p` • `SELECT SUBSTR(p.property_name, '@[^.]*') AS substr_val2 FROM asset_property AS p`

Function	Example
UPPER	`SELECT UPPER(d.string_value) AS up_string FROM raw_time_series AS d`
LOWER	`SELECT LOWER(d.string_value) AS low_string FROM raw_time_series AS d`
TRIM	`SELECT TRIM(d.string_value) AS tm_string FROM raw_time_series AS d`
LTRIM	`SELECT LTRIM(d.string_value) AS ltrim_string FROM raw_time_series AS d`
RTRIM	`SELECT RTRIM(d.string_value) AS rtrim_string FROM raw_time_series AS d`
STR_REPLACE	`SELECT STR_REPLACE(d.string_value, 'abc', 'def') AS replaced_string FROM raw_time_series AS d`

Concatenation operator

The concatenation operator `||`, or pipe operator, joins two strings together. It provides an alternative to the `CONCAT` function, and is more readable when combining multiple strings.

Example of the concatenation operator

```
SELECT a.asset_name || ' - ' || p.property_name
  AS full_name
  FROM asset a, asset_property p
```

Math functions

Math functions are pre-defined mathematical operations used within SQL queries to perform calculations on numerical data. They provide ways to manipulate and transform data without needing to extract it from the database and process it separately.

Math functions

Function	Signature	Description
`POWER`	POWER (int|double, int|double)	Returns the value of first argument raised to the power of the second argument.
`ROUND`	• ROUND (int|double, decimal_places_int) • ROUND (int|double)	Rounds to the nearest integer.
`FLOOR`	FLOOR (int|double)	Returns the largest integer not greater than the value given.

Examples of all functions:

Function	Example
POWER	• `POWER (3, 77)` • `POWER (2.3, 3.9)` • `POWER (1.0, 4.2)`
ROUND	`ROUND (32.12435, 3)`
FLOOR	`FLOOR (21.2)`

Date time functions

Date time functions work with dates and times. These functions allow extraction of specific components of a date, perform calculations, and manipulate date values.

The allowed identifiers in these functions are:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

Function	Signature	Description
`NOW`	NOW ( )	Returns the current timestamp with milliseco nd precision. It provides the exact time at the moment it's executed within a query.
`DATE_ADD`	DATE_ADD (identifier, interval_duration, column)	Returns the sum of a date/ time and a number of days/ hours, or of a date/time and date/time interval.
`DATE_SUB`	DATE_SUB (identifier, interval_duration, column)	Returns the difference between a date/time and a number of days/hours, or between a date/time and date/time interval.
`TIMESTAMP_ADD`	TIMESTAMP_ADD (identifier, interval_duration, column)	Adds an interval of time, in the given time units, to a datetime expression.
`TIMESTAMP_SUB`	TIMESTAMP_SUB (identifier, interval_duration, column)	Subtracts an interval of time, in the given time units, from a datetime expression.

Function	Signature	Description
CAST	CAST (expression AS TIMESTAMP FORMAT pattern)	Converts a string expression to a timestamp using the specified format pattern. Common patterns include `'yyyy-MM-dd HH:mm:ss'` for standard datetime format. For example, `SELECT CAST('2023-12-25 14:30:00' AS TIMESTAMP) AS converted_timestamp`

Example of a SQL query using the listed functions:

```
SELECT r.asset_id, r.int_value,
  date_add(DAY, 7, r.event_timestamp) AS date_in_future,
  date_sub(YEAR, 2, r.event_timestamp) AS date_in_past,
  timestamp_add(DAY, 2, r.event_timestamp) AS timestamp_in_future,
  timestamp_sub(DAY, 2, r.event_timestamp) AS timestamp_in_past,
  now() AS time_now
FROM raw_time_series AS r
```

Type conversion functions

Type conversion functions are used to change the data type of a value from one to another. They are essential for ensuring data compatibility and performing operations that require data in a specific format.

Function	Signature	Description
TO_DATE	• TO_DATE (integer) • TO_DATE (expression, format)	• Converts an epoch millisecond integer into a date value.

Function	Signature	Description
		• Converts a character string representation into a date value.
`TO_TIMESTAMP`	• TO_TIMESTAMP (double) • TO_TIMESTAMP (string, format)	• Converts an epoch second integer into a timestamp data type. • Converts a string represent ation of a date and time into a timestamp data type.
`TO_TIME`	• TO_TIME (int) • TO_TIME (string, format)	• Converts an epoch milliseco nd integer into a time value. • Converts a character string representation into a time value.
`CAST`	CAST (<expression> AS <data type>)	Converts an entity, or expression that evaluates to a single value, from one type to another. Supported data types are: • BOOLEAN • INTEGER • INT • TIMESTAMP • DATE • CHAR • CHARACTER • STRING

Example of a SQL query using the listed functions:

```
SELECT TO_TIMESTAMP (100) AS timestamp_value,
  TO_DATE(r.event_timestamp) AS date_value,
  TO_TIME(r.event_timestamp) AS time_value
FROM raw_time_series AS r
```

Aggregate functions

Aggregate functions are database operations that perform calculations across multiple rows of data to produce a single summarized result. These functions analyze data sets to return computed values like sums, averages, counts, or other statistical measures.

Function	Signature	Description
`AVG`	AVG (expression)	Returns the average of a numerical expression.
`COUNT`	COUNT (expression)	Returns the number of rows that match the given criteria.
`MAX`	MAX (expression)	Returns the largest value of the selected expressions.
`MIN`	MIN (expression)	Returns the smallest value of the selected expressions.
`SUM`	SUM (expression)	Returns the sum of a numerical expression.
`STDDEV`	STDDEV (expression)	Returns the sample standard deviation.
`GROUP BY`	GROUP BY expression	Returns a row created by the grouping columns.
`HAVING`	HAVING boolean-expression	Returns group rows filtered by `GROUP BY` clause.

Examples of all functions:

Function	Example
AVG	`SELECT d.asset_id, d.property_id, AVG(d.int_value) FROM raw_time_series AS d`
COUNT	`SELECT COUNT(d.int_value) FROM raw_time_series AS d`
MAX	`SELECT MAX(d.int_value) FROM raw_time_series AS d`
MIN	`SELECT MIN(d.int_value) FROM raw_time_series AS d`
SUM	`SELECT SUM(d.int_value) FROM raw_time_series AS d`
STDDEV	`SELECT STDDEV(d.int_value) FROM raw_time_series AS d`

• GROUP BY
• HAVING
	`SELECT MAX(d.int_value) AS max_int_value, d.asset_id`
	`FROM raw_time_series AS d`
	`GROUP BY d.asset_id`
	`HAVING MAX(d.int_value) > 5`

Example queries

Metadata filtering

The following example is for metadata filtering with a SELECT statement with the AWS IoT SiteWise query language:

```
SELECT a.asset_name, p.property_name
FROM asset a, asset_property p
WHERE a.asset_name LIKE 'Windmill%'
```

Value filtering

The following is an example of value filtering using a SELECT statement with the AWS IoT SiteWise query language:

```
SELECT a.asset_name, r.int_value
FROM asset a, raw_time_series r
WHERE r.int_value > 30
AND r.event_timestamp > TIMESTAMP '2022-01-05 12:15:00'
AND r.event_timestamp < TIMESTAMP '2022-01-05 12:20:00'
```

Query optimization

Metadata filters

When you query metadata or raw data, use the WHERE clause to filter by metadata fields to reduce the amount of data scanned. Use the following operators to limit the metadata scan:

- Equals (=)
- Not equals (!=)
- LIKE
- IN
- AND
- OR

For attribute properties, use the following fields to filter results.:

- `double_attribute_value`
- `int_attribute_value`
- `boolean_attribute_value`
- `string_attribute_value`

These fields provide better performance than the **latest_value_time_series** table for asset properties of attribute type.

Note

Use literals on the right side of operators to properly limit the data scan. For example, the following query performs worse than using a strict string literal:

```
SELECT property_id FROM asset_property WHERE property_name = CONCAT('my',
 'property')
```

Example for metadata filters:

```
SELECT p.property_name FROM asset_property p
WHERE p.property_type = 'attribute' AND p.string_attribute_value LIKE 'my-property-%'
```

Raw data filters

All raw data tables (**raw_time_series**, **latest_value_time_series**, **precomputed_aggregates**) have timestamps associated with their rows. In addition to metadata filters, use `WHERE` clause filters on the `event_timestamp` field to reduce the amount of data scanned. Use the following operations to limit the raw data scan:

- Equals (=)
- Greater than (>)
- Less than (<)
- Greater than or equals (>=)
- Less than or equals (<=)
- BETWEEN

- AND

When querying the **precomputed_aggregates** table, always specify a quality filter in the WHERE clause. This reduces the amount of data that the query scans, especially if you're looking for BAD or UNCERTAIN data. We also highly recommend using a resolution filter (1m, 15m, 1h, or 1d) when querying the **precomputed_aggregates** table. If you don't specify a resolution filter, AWS IoT SiteWise will default to a full table scan across all resolutions, which is inefficient.

Note

Not equals (!=) and OR operators typically don't apply meaningful filters to the raw data scan. Filters on raw data values (string_value, double_value, etc.) also don't limit the raw data scan.

JOIN optimization

AWS IoT SiteWise SQL supports the JOIN keyword to merge two tables together. Only JOINs that actively filter on a field (using the ON keyword) are supported. Full Cartesian joins are prohibited.

AWS IoT SiteWise also supports implicit JOINs without using the JOIN keyword. These are allowed between different metadata tables and between a metadata table and a raw table. For example, this query:

```
SELECT a.asset_name, p.property_name FROM asset a, asset_property p
```

Performs better than this equivalent query:

```
SELECT a.asset_name, p.property_name FROM asset a
JOIN asset_property p ON a.asset_id = p.asset_id
```

The following implicit joins are allowed (O is allowed, X is prohibited):

	asset	asset_property	latest_value_time_series	raw_time_series	precomputed_aggregates	subquery
asset	X	O	O	O	O	X

	asset	asset_property	latest_value_time_series	raw_time_series	precomputed_aggregates	subquery
asset_property	O	X	O	O	O	X
latest_value_time_series	O	O	X	X	X	X
raw_time_series	O	O	X	X	X	X
precomputed_aggregates	O	O	X	X	X	X
subquery	X	X	X	X	X	X

Use implicit `JOIN`s where possible. If you must use the `JOIN` keyword, apply filters on the individual `JOIN`ed tables to minimize data scanned. For example, instead of this query:

```
SELECT level1.asset_id, level2.asset_id, level3.asset_id
FROM asset AS level1
JOIN asset AS level2 ON level2.parent_asset_id = level1.asset_id
JOIN asset AS level3 ON level3.parent_asset_id = level2.asset_id
WHERE level1.asset_name LIKE 'level1%'
AND level2.asset_name LIKE 'level2%'
AND level3.asset_name LIKE 'level3%'
```

Use this more efficient query:

```
SELECT level1.asset_id, level2.asset_id, level3.asset_id
FROM asset AS level1
JOIN (SELECT asset_id, parent_asset_id FROM asset WHERE asset_name LIKE 'level2%') AS
 level2 ON level2.parent_asset_id = level1.asset_id
JOIN (SELECT asset_id, parent_asset_id FROM asset WHERE asset_name LIKE 'level3%') AS
 level3 ON level3.parent_asset_id = level2.asset_id
```

```
WHERE level1.asset_name LIKE 'level1%'
```

By pushing metadata filters into subqueries, you ensure that individual tables in the JOINs are filtered during the scanning process. You can also use the LIMIT keyword in subqueries for the same effect.

Large queries

For queries that produce more rows than the default, set the page size of the ExecuteQuery API to the maximum value of 20000. This improves overall query performance.

Use the LIMIT clause to reduce the amount of data scanned for some queries. Note that aggregate functions and certain table-wide clauses (GROUP BY, ORDER BY, JOIN) require a full scan to complete before applying the LIMIT clause.

> **Note**
>
> AWS IoT SiteWise may scan a minimum amount of data even with the LIMIT clause applied, especially for raw data queries that scan over multiple properties.

ODBC

The open-source ODBC driver for AWS IoT SiteWise provides an SQL-relational interface to AWS IoT SiteWise for developers and enables connectivity from business intelligence (BI) tools such as Power BI Desktop and Microsoft Excel. The AWS IoT SiteWise ODBC driver is currently available on Windows, and supports SSO with Okta and Microsoft Azure Active Directory (AD).

For more information, see AWS IoT SiteWise ODBC driver documentation on GitHub.

Topics

- Connection string syntax and options for the ODBC driver
- Connection string examples for the AWS IoT SiteWise ODBC driver
- Troubleshooting connection with the ODBC driver

Connection string syntax and options for the ODBC driver

The syntax for specifying connection-string options for the ODBC driver is as follows:

```
Driver={AWS IoT SiteWise ODBC Driver};(option)=(value);
```

Available options are as follows:

Driver connection options

- **`Driver`** *(required)* – The driver being used with ODBC.

  The default is AWS IoT SiteWise.

- **`DSN`** – The data source name (DSN) to use for configuring the connection.

  The default is `NONE`.

- **`Auth`** – The authentication mode. This must be one of the following:
  - `AWS_PROFILE` – Use the default credential chain.
  - `IAM` – Use AWS IAM credentials.
  - `AAD` – Use the Azure Active Directory (AD) identity provider.
  - `OKTA` – Use the Okta identity provider.

  The default is `AWS_PROFILE`.

Endpoint configuration options

- **`EndpointOverride`** – The endpoint override for the AWS IoT SiteWise service. This is an advanced option that overrides the region. For example:

  ```
  iotsitewise.us-east-1.amazonaws.com
  ```

- **`Region`** – The signing region for the AWS IoT SiteWise service endpoint.

  The default is `us-east-1`.

Credentials provider option

- **`ProfileName`** – The profile name in the AWS config file.

  The default is `NONE`.

AWS IAM authentication options

- **`UID`** or **`AccessKeyId`** – The AWS user access key id. If both `UID` and `AccessKeyId` are provided in the connection string, the `UID` value will be used unless it is empty.

  The default is `NONE`.
- **`PWD`** or **`SecretKey`** – The AWS user secret access key. If both `PWD` and `SecretKey` are provided in the connection string, the `PWD` value with will be used unless it's empty.

  The default is `NONE`.
- **`SessionToken`** – The temporary session token required to access a database with multi-factor authentication (MFA) enabled. Do not include a trailing `=` in the input.

  The default is `NONE`.

SAML-based authentication options for Okta

- **`IdPHost`** – The hostname of the specified IdP.

  The default is `NONE`.
- **`UID`** or **`IdPUserName`** – The user name for the specified IdP account. If both `UID` and `IdPUserName` are provided in the connection string, the `UID` value will be used unless it's empty.

  The default is `NONE`.
- **`PWD`** or **`IdPPassword`** – The password for the specified IdP account. If both `PWD` and `IdPPassword` are provided in the connection string, the `PWD` value will be used unless it's empty.

  The default is `NONE`.
- **`OktaApplicationID`** – The unique Okta-provided ID associated with the AWS IoT SiteWise application. A place to find the application ID (AppId) is in the `entityID` field provided in the application metadata. An example is:

  ```
  entityID="http://www.okta.com//(IdPAppID)
  ```

  The default is `NONE`.
- **`RoleARN`** – The Amazon Resource Name (ARN) of the role that the caller is assuming.

  The default is `NONE`.

- **`IdPARN`** – The Amazon Resource Name (ARN) of the SAML provider in IAM that describes the IdP.

  The default is `NONE`.

SAML-based authentication options for Azure Active Directory

- **`UID`** or **`IdPUserName`** – The user name for the specified IdP account..

  The default is `NONE`.
- **`PWD`** or **`IdPPassword`** – The password for the specified IdP account.

  The default is `NONE`.
- **`AADApplicationID`** – The unique id of the registered application on Azure AD.

  The default is `NONE`.
- **`AADClientSecret`** – The client secret associated with the registered application on Azure AD used to authorize fetching tokens.

  The default is `NONE`.
- **`AADTenant`** – The Azure AD Tenant ID.

  The default is `NONE`.
- **`RoleARN`** – The Amazon Resource Name (ARN) of the role that the caller is assuming.

  The default is `NONE`.
- **`IdPARN`** – The Amazon Resource Name (ARN) of the SAML provider in IAM that describes the IdP.

  The default is `NONE`.

AWS SDK (advanced) Options

- **`RequestTimeout`** – The time in milliseconds that the AWS SDK waits for a query request before timing out. Any non-positive value disables the request timeout.

  The default is `3000`.

- **`ConnectionTimeout`** – The time in milliseconds that the AWS SDK waits for data to be transferred over an open connection before timing out. A value of 0 disables the connection timeout. This value must not be negative.

  The default is `1000`.

- **`MaxRetryCountClient`** – The maximum number of retry attempts for retryable errors with 5xx error codes in the SDK. The value must not be negative.

  The default is `0`.

- **`MaxConnections`** – The maximum number of allowed concurrently open HTTP connections to the AWS IoT SiteWise service. The value must be positive.

  The default is `25`.

ODBC driver logging Options

- **`LogLevel`** – The log level for driver logging. Must be one of:
  - **0** (OFF).
  - **1** (ERROR).
  - **2** (WARNING).
  - **3** (INFO).
  - **4** (DEBUG).

  The default is `1` (ERROR).

  Warning: personal information could be logged by the driver when using the DEBUG logging mode.

- **`LogOutput`** – Folder in which to store the log file.

  The default is:

  - **Windows:** `%USERPROFILE%`, or if not available, `%HOMEDRIVE%%HOMEPATH%`.
  - **macOS and Linux:** `$HOME`, or if not available, the field `pw_dir` from the function `getpwuid(getuid())` return value.

SDK logging options

The AWS SDK log level is separate from the AWS IoT SiteWise ODBC driver log level. Setting one does not affect the other.

The SDK Log Level is set using the environment variable `SW_AWS_LOG_LEVEL`. Valid values are:

- `OFF`
- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`
- `FATAL`

If `SW_AWS_LOG_LEVEL` is not set, the SDK log level is set to the default, which is `WARN`.

Connecting through a proxy

The ODBC driver supports connecting to AWS IoT SiteWise through a proxy. To use this feature, configure the following environment variables based on your proxy setting:

- **`SW_PROXY_HOST`** – the proxy host.
- **`SW_PROXY_PORT`** – The proxy port number.
- **`SW_PROXY_SCHEME`** – The proxy scheme, either `http` or `https`.
- **`SW_PROXY_USER`** – The user name for proxy authentication.
- **`SW_PROXY_PASSWORD`** – The user password for proxy authentication.
- **`SW_PROXY_SSL_CERT_PATH`** – The SSL Certificate file to use for connecting to an HTTPS proxy.
- **`SW_PROXY_SSL_CERT_TYPE`** – The type of the proxy client SSL certificate.
- **`SW_PROXY_SSL_KEY_PATH`** – The private key file to use for connecting to an HTTPS proxy.
- **`SW_PROXY_SSL_KEY_TYPE`** – The type of the private key file used to connect to an HTTPS proxy.
- **`SW_PROXY_SSL_KEY_PASSWORD`** – The passphrase to the private key file used to connect to an HTTPS proxy.

Connection string examples for the AWS IoT SiteWise ODBC driver

Example of connecting to the ODBC driver with IAM credentials

```
Driver={AWS IoT SiteWise ODBC Driver};Auth=IAM;AccessKeyId=(your access key
 ID);SecretKey=(your secret key);SessionToken=(your session token);Region=us-east-1;
```

Example of connecting to the ODBC driver with a profile

```
Driver={AWS IoT SiteWise ODBC Driver};ProfileName=(the profile name);region=us-east-1;
```

The driver will attempt to connect using the credentials provided in `~/.aws/credentials`, or if a file is specified in the environment variable `AWS_SHARED_CREDENTIALS_FILE`, using the credentials in that file.

Example of connecting to the ODBC driver with Okta

```
Driver={AWS IoT SiteWise ODBC Driver};Auth=OKTA;region=us-east-1;idPHost=(your host at
 Okta);idPUsername=(your user name);idPPassword=(your password);OktaApplicationID=(your
 Okta AppId);roleARN=(your role ARN);idPARN=(your Idp ARN);
```

Example of connecting to the ODBC driver with Azure Active Directory (AAD)

```
Driver={AWS IoT SiteWise ODBC Driver};Auth=AAD;region=us-east-1;idPUsername=(your
 user name);idPPassword=(your password);aadApplicationID=(your AAD
 AppId);aadClientSecret=(your AAD client secret);aadTenant=(your AAD
 tenant);roleARN=(your role ARN);idPARN=(your idP ARN);
```

Example of connecting to the ODBC driver with a specified endpoint and a log level of 2 (WARNING)

```
Driver={AWS IoT SiteWise ODBC Driver};Auth=IAM;AccessKeyId=(your access
 key ID);SecretKey=(your secret key);EndpointOverride=iotsitewise.us-
east-1.amazonaws.com;Region=us-east-1;LogLevel=2;
```

Troubleshooting connection with the ODBC driver

> **Note**
>
> If the username and password is already specified in the DSN, do not specify them again when the ODBC driver manager asks for them.

An error code of `01S02` with a message, `Re-writing` *`(connection string option)`* `(have you specified it several times?)` occurs when a connection string option is passed more than once in the connection string. Specifying an option more than once raises an error. When making a connection with a DSN and a connection string, if a connection option is already specified in the DSN, do not specify it again in the connection string.

Interact 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 Publish property value updates to Amazon DynamoDB.

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 Ingest data to AWS IoT SiteWise using AWS IoT Core rules.

Topics

Understand asset properties in 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*.

Turn on asset property notifications in AWS IoT SiteWise

You can enable property notifications to publish asset data updates to AWS IoT Core, and then run queries on your data. With asset property notifications, AWS IoT SiteWise provides an AWS CloudFormation template that you can use to export AWS IoT SiteWise data to Amazon S3.

> **Note**
>
> Asset data is sent to AWS IoT Core every time it's received by AWS IoT SiteWise, regardless of if the value has changed.

Topics

- Turn on asset property notifications (console)
- Turn on asset property notifications (AWS CLI)

Turn on asset property notifications (console)

By default, AWS IoT SiteWise doesn't publish property value updates. 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.

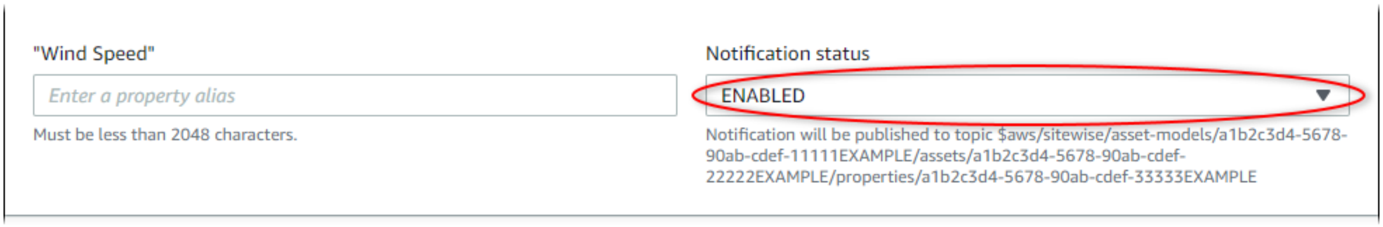
2. In the navigation pane, choose **Assets**.

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**.



   You can also choose **DISABLED** to disable notifications for the asset property.

6. Choose **Save**.

Turn on asset property notifications (AWS CLI)

By default, AWS IoT SiteWise doesn't publish property value updates. 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. You can also use the external ID. If you created an asset and don't know its `assetId`, use the ListAssets API to list all the assets for a specific model. 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.

```
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.

```
{
  "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/properties/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.

```
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.

Query asset property notifications in AWS IoT SiteWise

To query asset property notifications, create AWS IoT Core rules made up of SQL statements.

AWS IoT SiteWise publishes asset property data updates to AWS IoT Core in the following format.

```
{
  "type": "PropertyValueUpdate",
  "payload": {
    "assetId": "String",
    "propertyId": "String",
    "values": [
      {
        "timestamp": {
          "timeInSeconds": Number,
          "offsetInNanos": Number
        },
        "quality": "String",
        "value": {
          "booleanValue": Boolean,
          "doubleValue": Number,
          "integerValue": Number,
          "stringValue": "String",
          "nullValue": {
            "valueType": "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
  - nullValue

nullValue – A structure with the following field denoting the type of the property value with value Null and quality of BAD or UNCERTAIN.

- valueType – Enum of {"B", "D", "S", "I"}

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 Publish property value updates to Amazon DynamoDB 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.

```
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.

```
{
  "windspeed": [
```

```
    26.32020195042838,
    26.282584572975477,
    26.352566977372508,
    26.283084346171442,
    26.571883739599322,
    26.60684140743005,
    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.

```
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.

```
{
  "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.

Export data to Amazon S3 with 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.

To export time series data from AWS IoT SiteWise, enable the cold tier feature to have the data stored in an Amazon S3 bucket. See Manage data storage in AWS IoT SiteWise for details.

To export asset model and asset metadata from AWS IoT SiteWise, use the bulk operations feature to export metadata to an Amazon S3 bucket. See Bulk operations with assets and models for details.

Integrate AWS IoT SiteWise with Grafana

Grafana is a data visualization platform used to visualize and monitor data in dashboards. In Grafana version 10.4.0 and later, use the AWS IoT SiteWise plugin to visualize your AWS IoT SiteWise asset data in Grafana dashboards. Users 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. You can access this demo dashboard on the Grafana Play website.

Integrate AWS IoT SiteWise and AWS IoT TwinMaker

Integrating with AWS IoT TwinMaker grants access to robust functionality in AWS IoT SiteWise, such as AWS IoT SiteWise data retrieval `ExecuteQuery` API and advanced asset search in the AWS IoT SiteWise console. To integrate the services and use these features, you must first enable the integration.

Topics

Enabling the integration

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. For more information about AWS IoT SiteWise supported actions, see Actions defined by AWS IoT SiteWise in the *Service Authorization Reference*.

For more information about AWS IoT TwinMaker service-linked role, see Service-linked roles for AWS IoT TwinMaker in the *AWS IoT TwinMaker User Guide*.

Before you can integrate AWS IoT SiteWise and AWS IoT TwinMaker, you must grant the following permissions that allow AWS IoT SiteWise to integrate with an AWS IoT TwinMaker linked workspace:

- `iotsitewise:EnableSiteWiseIntegration` – Allows AWS IoT SiteWise to integrate with a linked AWS IoT TwinMaker workspace. This integration allows AWS IoT TwinMaker to read all your modeling information in AWS IoT SiteWise through an AWS IoT TwinMaker service-linked role. To enable this permission, add the following policy to your IAM role:

  JSON

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

Integrating AWS IoT SiteWise and AWS IoT TwinMaker

To integrate AWS IoT SiteWise and AWS IoT TwinMaker, you must have the following:

- AWS IoT SiteWise service-linked role set up in your account
- AWS IoT TwinMaker service-linked role set up in your account
- AWS IoT TwinMaker workspace with ID `IoTSiteWiseDefaultWorkspace` in your account in the Region.

To integrate by using the AWS IoT SiteWise console

When you see the **Integration with AWS IoT TwinMaker** banner in the console, choose **Grant permission**. The prerequisites are created in your account.

To integrate by using the AWS CLI

To integrate AWS IoT SiteWise and AWS IoT TwinMaker by using the AWS CLI, enter the following commands:

1. Call `CreateServiceLinkedRole` with an `AWSServiceName` of `iotsitewise.amazonaws.com`.

   ```
   aws iam create-service-linked-role --aws-service-name iotsitewise.amazonaws.com
   ```

2. Call `CreateServiceLinkedRole` with an `AWSServiceName` of `iottwinmaker.amazonaws.com`.

   ```
   aws iam create-service-linked-role --aws-service-name iottwinmaker.amazonaws.com
   ```

3. Call `CreateWorkspace` with an `ID` of `IoTSiteWiseDefaultWorkspace`.

   ```
    aws iottwinmaker create-workspace --workspace-id IoTSiteWiseDefaultWorkspace
   ```

Detect anomalies with Lookout for Equipment

Note

Anomaly detection is only available in the Regions where Amazon Lookout for Equipment is available.

You can integrate AWS IoT SiteWise with Amazon Lookout for Equipment to gain insights about your industrial equipment through anomaly detection and predictive maintenance of industrial equipment. Lookout for Equipment is a machine learning (ML) service for monitoring industrial equipment that detects abnormal equipment behavior and identifies potential failures. With Lookout for Equipment, you can implement predictive maintenance programs and identify suboptimal equipment processes. For more information about Lookout for Equipment, see What is Amazon Lookout for Equipment? in the *Amazon Lookout for Equipment User Guide*.

When you create a prediction to train an ML model to detect anomalous equipment behavior, AWS IoT SiteWise sends asset property values to Lookout for Equipment to train an ML model to detect anomalous equipment behavior. To define a prediction definition on an asset model, you specify the IAM roles needed for Lookout for Equipment to access your data and the properties to send to Lookout for Equipment and send processed data to Amazon S3. For more information, see Create asset models in AWS IoT SiteWise.

To integrate AWS IoT SiteWise and Lookout for Equipment, you'll perform the following high-level steps:

- Add a prediction definition on an asset model that outlines what properties you want to track. The prediction definition is a reusable collection of measurements, transforms, and metrics that is used to create predictions on the assets that are based on that asset model.
- Train the prediction based on historical data that you provide.
- Schedule inference, which tells AWS IoT SiteWise how often to run a specific prediction.

Once inference is scheduled, the Lookout for Equipment model monitors the data it receives from your equipment and looks for anomalies in equipment behavior. You can view and analyze the results in SiteWise Monitor, using the AWS IoT SiteWise GET API operations, or the Lookout for Equipment console. You can also create alarms using alarm detectors from the asset model to alert you about abnormal equipment behavior.

Topics

Add a prediction definition (console)

To begin sending data collected by AWS IoT SiteWise to Lookout for Equipment, you must add an AWS IoT SiteWise prediction definition to an asset model.

To add a prediction definition to an AWS IoT SiteWise asset model

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Models** and select the asset model to which you want to add the prediction definition.
3. Choose **Predictions**.
4. Choose **Add prediction definition**.
5. Define details about the prediction definition.

    a. Enter a unique **Name** and a **Description** for your prediction definition. Choose the name thoughtfully because after you create the prediction definition, you can't change its name.

    b. Create or select an **IAM permissions role** that allows AWS IoT SiteWise to share your asset data with Amazon Lookout for Equipment. The role should have the following IAM and trust policies. For help creating the role, see Creating a role using custom trust policies (console).

    IAM policy

    JSON

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "L4EPermissions",
            "Effect": "Allow",
            "Action": [
                "lookoutequipment:CreateDataset",
                "lookoutequipment:CreateModel",
                "lookoutequipment:CreateInferenceScheduler",
                "lookoutequipment:DescribeDataset",
                "lookoutequipment:DescribeModel",
                "lookoutequipment:DescribeInferenceScheduler",
                "lookoutequipment:ListInferenceExecutions",
                "lookoutequipment:StartDataIngestionJob",
                "lookoutequipment:StartInferenceScheduler",
                "lookoutequipment:UpdateInferenceScheduler",
                "lookoutequipment:StopInferenceScheduler"
            ],
            "Resource": [
                "arn:aws:lookoutequipment:us-
east-1:123456789012:inference-scheduler/IoTSiteWise_*",
                "arn:aws:lookoutequipment:us-east-1:123456789012:model/
IoTSiteWise_*",
                "arn:aws:lookoutequipment:us-east-1:123456789012:dataset/
IoTSiteWise_*"
            ]
        },
        {
            "Sid": "L4EPermissions2",
            "Effect": "Allow",
            "Action": [
                "lookoutequipment:DescribeDataIngestionJob"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::iotsitewise-*"
            ]
        },
        {
            "Sid": "IAMPermissions",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/Role_name"
        }
    ]
}
```

Trust policy

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iotsitewise.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:iotsitewise:us-
east-1:123456789012:asset/*"
                }
            }
        },
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "lookoutequipment.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:lookoutequipment:us-east-1:123456789012:*"
                }
            }
        }
    ]
}
```

c. Choose **Next**.

6. Select data attributes (measurements, transforms, and metrics) that you want to send to Lookout for Equipment.

   a. (Optional) Select measurements.

   b. (Optional) Select transforms.

   c. (Optional) Select metrics.

   d. Choose **Next**.

7. Review your selections. To add the prediction definition to the asset model, on the summary page, choose **Add prediction definition**.

You can also **Edit** or **Delete** an existing prediction definition that has active predictions attached.

Train a prediction (console)

After you've added a prediction definition to an asset model, you can train the predictions that are on your assets.

To train a prediction in AWS IoT SiteWise

1. Navigate to the [AWS IoT SiteWise console](https://console.aws.amazon.com/iotsitewise/).

2. In the navigation pane, choose **Assets**, and select the asset you want to monitor.

3. Choose **Predictions**.
4. Select the predictions that you want to train.
5. Under **Actions**, choose **Start training**, and do the following:

    a. Under **Prediction details**, select an IAM permissions role that allows AWS IoT SiteWise to share your asset data with Lookout for Equipment. If you need to create a new role, choose **Create a new role**.

    b. For **Training data settings**, enter a **Training data time range** to select which data to use to train the prediction.

    c. (Optional) Select sampling rate of the data after post processing.

    d. (Optional) For **Data labels**, provide an Amazon S3 bucket and prefix that holds your labeling data. For more information about labeling data, see Labeling your data in the *Amazon Lookout for Equipment User Guide*.

    e. Choose **Next**.

6. (Optional) If you want the prediction to be active as soon as it has completed training, under **Advanced settings**, select **Automatically activate the prediction after training**, and then do the following:

    a. Under **Input data**, for **Data upload frequency**, define how often data is uploaded, and for **Offset delay time**, define how much of a buffer to use.

    b. Choose **Next**.

7. Review the details of the prediction and choose **Save and start**.

Start or stop inference on a prediction (console)

> **Note**
>
> Lookout for Equipment charges apply to scheduled inferences with the data transferred between AWS IoT SiteWise and Lookout for Equipment. For more information, see Amazon Lookout for Equipment pricing.

If you added the prediction `lookoutequipment:CreateDataset`, but did not choose to activate it after training, you must activate it to start monitoring your assets.

To start inference for a prediction

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**, and select the asset the prediction is added to.
3. Choose **Predictions**.
4. Select the predictions that you want to activate.
5. Under **Actions**, choose **Start inference**, and do the following:

    a. Under **Input data**, for **Data upload frequency**, define how often data is uploaded, and for **Offset delay time**, define how much of a buffer to use.

    b. Choose **Save and start**.

To stop inference for a prediction

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Assets**, and select the asset the prediction is added to.
3. Choose **Predictions**.
4. Select the predictions that you want to stop.
5. Under **Actions**, choose **Stop inference**.

Add a prediction definition (CLI)

To define a prediction definition on a new or existing asset model, you can use the AWS Command Line Interface (AWS CLI). After you define the prediction definition on the asset model, you train, and schedule inference for, a prediction on an asset in AWS IoT SiteWise to do anomaly detection with Lookout for Equipment.

Prerequisites

To complete these steps, you must have an asset model and at least one asset created. For more information, see Create an asset model (AWS CLI) and Create an asset (AWS CLI).

If you are new to AWS IoT SiteWise, you must call the `CreateBulkImportJob` API operation to import asset property values into AWS IoT SiteWise, which will be used to train the model. For more information, see Create an AWS IoT SiteWise bulk import job (AWS CLI).

To add a prediction definition

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.

   - For more information about how to create an asset model, see Create an asset model (AWS CLI)
   - For more information about how to update an existing asset model, see Update an asset model, component model, or interface (AWS CLI)

2. Add a Lookout for Equipment composite model (`assetModelCompositeModels`) to the asset model by adding the following code.

   - Replace *`Property`* with the ID of the properties that you want to include. To get those IDs, call `DescribeAssetModel`.
   - Replace *`RoleARN`* with the ARN of an IAM role that allows Lookout for Equipment to access your AWS IoT SiteWise data.

```
{
  ...
  "assetModelCompositeModels": [
    {
      "name": "L4Epredictiondefinition",
      "type": "AWS/L4E_ANOMALY",
      "properties": [
          {
            "name": "AWS/L4E_ANOMALY_RESULT",
            "dataType": "STRUCT",
            "dataTypeSpec": "AWS/L4E_ANOMALY_RESULT",
            "unit": "none",
            "type": {
              "measurement": {}
            }
          },
          {
            "name": "AWS/L4E_ANOMALY_INPUT",
            "dataType": "STRUCT",
            "dataTypeSpec": "AWS/L4E_ANOMALY_INPUT",
            "type": {
               "attribute": {
```

```
                  "defaultValue": "{\"properties\": [\"Property1\", \"Property2\"]}"
                }
              }
            },
            {
              "name": "AWS/L4E_ANOMALY_PERMISSIONS",
              "dataType": "STRUCT",
              "dataTypeSpec": "AWS/L4E_ANOMALY_PERMISSIONS",
              "type": {
                "attribute": {
                  "defaultValue": "{\"roleArn\": \"RoleARN\"}"
                }
              }
            },
            {
              "name": "AWS/L4E_ANOMALY_DATASET",
              "dataType": "STRUCT",
              "dataTypeSpec": "AWS/L4E_ANOMALY_DATASET",
              "type": {
                  "attribute": {}
              }
            },
            {
              "name": "AWS/L4E_ANOMALY_MODEL",
              "dataType": "STRUCT",
              "dataTypeSpec": "AWS/L4E_ANOMALY_MODEL",
              "type": {
                "attribute": {}
              }
            },
            {
              "name": "AWS/L4E_ANOMALY_INFERENCE",
              "dataType": "STRUCT",
              "dataTypeSpec": "AWS/L4E_ANOMALY_INFERENCE",
              "type": {
                "attribute": {}
              }
            },
            {
              "name": "AWS/L4E_ANOMALY_TRAINING_STATUS",
              "dataType": "STRUCT",
              "dataTypeSpec": "AWS/L4E_ANOMALY_TRAINING_STATUS",
              "type": {
                "attribute": {
```

```
                "defaultValue": "{}"
              }
            }
          },
          {
            "name": "AWS/L4E_ANOMALY_INFERENCE_STATUS",
            "dataType": "STRUCT",
            "dataTypeSpec": "AWS/L4E_ANOMALY_INFERENCE_STATUS",
            "type": {
              "attribute": {
                "defaultValue": "{}"
              }
            }
          }
    ]
}
```

3. Create the asset model or update the existing asset model. Do one of the following:

   - To create the asset model, run the following command:

     ```
     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 that you want to update.

     ```
     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.

Train a prediction and starting inference (CLI)

Now that the prediction definition is defined, you can train assets based on it and start inference. If you want to train your prediction but not start inference, skip to Train a prediction (CLI). To train the prediction and start inference on the asset, you'll need the `assetId` of the target resource.

To train and start inference of the prediction

1. Run the following command to find the `assetModelCompositeModelId` under `assetModelCompositeModelSummaries`. Replace *`asset-model-id`* with the ID of the asset model that you created in [Update an asset model, component model, or interface (AWS CLI)](#).

```
aws iotsitewise describe-asset-model \
  --asset-model-id asset-model-id \
```

2. Run the following command to find the `actionDefinitionId` of the `TrainingWithInference` action. Replace *`asset-model-id`* with the ID used in previous step and replace *`asset-model-composite-model-id`* with the ID returned in the previous step.

```
aws iotsitewise describe-asset-model-composite-model \
  --asset-model-id asset-model-id \
  --asset-model-composite-model-id asset-model-composite-model-id \
```

3. Create a file called `train-start-inference-prediction.json` and add the following code, replacing the following:

   - *`asset-id`* with the ID of the target asset
   - *`action-definition-id`* with the ID of the TrainingWithInference action
   - *`StartTime`* with the start of the training data, provided in epoch seconds
   - *`EndTime`* with the end of the training data, provided in epoch seconds
   - *`TargetSamplingRate`* with the sampling rate of the data after post processing by Lookout for Equipment. Allowed values are: `PT1S | PT5S | PT10S | PT15S | PT30S | PT1M | PT5M | PT10M | PT15M | PT30M | PT1H`.

```
{
  "targetResource": {
    "assetId": "asset-id"
  },
  "actionDefinitionId": "action-definition-Id",
  "actionPayload":{
    "stringValue": "{\"l4ETrainingWithInference\":{\"trainingWithInferenceMode
\":\"START\",\"trainingPayload\":{\"exportDataStartTime\":StartTime,
```

```
\"exportDataEndTime\":EndTime},\"targetSamplingRate\":\"TargetSamplingRate\"},
\"inferencePayload\":{\"dataDelayOffsetInMinutes\":0,\"dataUploadFrequency\":\"PT5M
\"}}}"
  }
}
```

4. Run the following command to start training and inference:

```
aws iotsitewise execute-action --cli-input-json file://train-start-inference-
prediction.json
```

Train a prediction (CLI)

Now that the prediction definition is defined, you can train assets based on it. To train the prediction on the asset, you'll need the `assetId` of the target resource.

To train the prediction

1. Run the following command to find the `assetModelCompositeModelId` under `assetModelCompositeModelSummaries`. Replace *`asset-model-id`* with the ID of the asset model that you created in [Update an asset model, component model, or interface (AWS CLI)](#).

```
aws iotsitewise describe-asset-model \
  --asset-model-id asset-model-id \
```

2. Run the following command to find the `actionDefinitionId` of the `Training` action. Replace *`asset-model-id`* with the ID used in previous step and replace *`asset-model-composite-model-id`* with the ID returned in the previous step.

```
aws iotsitewise describe-asset-model-composite-model \
  --asset-model-id asset-model-id \
  --asset-model-composite-model-id asset-model-composite-model-id \
```

3. Create a file called `train-prediction.json` and add the following code, replacing the following:

   - *`asset-id`* with the ID of the target asset
   - *`action-definition-id`* with the ID of the training action

- *StartTime* with the start of the training data, provided in epoch seconds
- *EndTime* with the end of the training data, provided in epoch seconds
- (Optional) *BucketName* with the name of the Amazon S3 bucket that holds your label data
- (Optional) *Prefix* with the prefix associated with the Amazon S3 bucket.
- *TargetSamplingRate* with the sampling rate of the data after post processing by Lookout for Equipment. Allowed values are: `PT1S | PT5S | PT10S | PT15S | PT30S | PT1M | PT5M | PT10M | PT15M | PT30M | PT1H`.

> **Note**
>
> Include both the bucket name and prefix or neither.

```
{
  "targetResource": {
    "assetId": "asset-id"
  },
  "actionDefinitionId": "action-definition-Id",
  "actionPayload":{ "stringValue": "{\"l4ETraining\": {\"trainingMode\":
\"START\",\"exportDataStartTime\": StartTime, \"exportDataEndTime\": EndTime,
 \"targetSamplingRate\":\"TargetSamplingRate\"}, \"labelInputConfiguration\":
 {\"bucketName\": \"BucketName\", \"prefix\": \"Prefix\"}}}"
}
}
```

4. Run the following command to start training:

```
aws iotsitewise execute-action --cli-input-json file://train-prediction.json
```

Before you can start inference, training must be completed. To check the status of the training, do one of the following:

- From the console, navigate to the asset the prediction is on.
- From the AWS CLI, call `BatchGetAssetPropertyValue` using the `propertyId` of the `trainingStatus` property.

Start or stop inference on a prediction (CLI)

Once the prediction is trained, you can start inference to tell Lookout for Equipment to start monitoring your assets. To start or stop inference, you'll need the `assetId` of the target resource.

To start inference

1. Run the following command to find the `assetModelCompositeModelId` under `assetModelCompositeModelSummaries`. Replace *`asset-model-id`* with the ID of the asset model that you created in [Update an asset model, component model, or interface (AWS CLI)](#).

```
aws iotsitewise describe-asset-model \
  --asset-model-id asset-model-id \
```

2. Run the following command to find the `actionDefinitionId` of the `Inference` action. Replace *`asset-model-id`* with the ID used in previous step and replace *`asset-model-composite-model-id`* with the ID returned in the previous step.

```
aws iotsitewise describe-asset-model-composite-model \
  --asset-model-id asset-model-id \
  --asset-model-composite-model-id asset-model-composite-model-id \
```

3. Create a file called `start-inference.json` and add the following code, replacing the following:

   - *`asset-id`* with the ID of the target asset
   - *`action-definition-id`* with the ID of the start inference action
   - *`Offset`* with the amount of buffer to use
   - *`Frequency`* with how often data is uploaded

```
{
  "targetResource": {
    "assetId": "asset-id"
  },
  "actionDefinitionId": "action-definition-Id",
  "actionPayload":{ "stringValue": "{\"l4EInference\": {\"inferenceMode\":\"START
\",\"dataDelayOffsetInMinutes\": Offset, \"dataUploadFrequency\": \"Frequency\"}}"
```

```
}}
```

4. Run the following command to start inference:

```
aws iotsitewise execute-action --cli-input-json file://start-inference.json
```

To stop inference

1. Run the following command to find the `assetModelCompositeModelId` under `assetModelCompositeModelSummaries`. Replace *`asset-model-id`* with the ID of the asset model that you created in [Update an asset model, component model, or interface (AWS CLI)](#).

```
aws iotsitewise describe-asset-model \
  --asset-model-id asset-model-id \
```

2. Run the following command to find the `actionDefinitionId` of the `Inference` action. Replace *`asset-model-id`* with the ID used in previous step and replace *`asset-model-composite-model-id`* with the ID returned in the previous step.

```
aws iotsitewise describe-asset-model-composite-model \
  --asset-model-id asset-model-id \
  --asset-model-composite-model-id asset-model-composite-model-id \
```

3. Create a file called `stop-inference.json` and add the following code, replacing the following:

   - *`asset-id`* with the ID of the target asset
   - *`action-definition-id`* with the ID of the start inference action

```
{
  "targetResource": {
    "assetId": "asset-id"
  },
  "actionDefinitionId": "action-definition-Id",
  "actionPayload":{ "stringValue": "{\"l4EInference\":{\"inferenceMode\":\"STOP
\"}}"
}}
```

4. Run the following command to stop inference:

```
aws iotsitewise execute-action --cli-input-json file://stop-inference.json
```

Native anomaly detection

AWS IoT SiteWise native anomaly detection is a machine learning (ML) feature for monitoring industrial equipment that detects abnormal equipment behavior and identifies potential failures. With native anomaly detection, you can implement predictive maintenance programs and identify suboptimal equipment processes.

AWS IoT SiteWise native anomaly detection doesn't require extensive ML knowledge or experience. You simply select the properties to train a custom ML model that finds potential failures for you. AWS IoT SiteWise native anomaly detection automatically creates the best model to learn your equipment's normal operating conditions. The model is optimized to find abnormal equipment behavior that occurred in the historical data. Using either the AWS IoT SiteWise console or the SDK, you run the model to process new time-series data according to your desired schedule.

To use AWS IoT SiteWise native anomaly detection, you do the following:

- Select the properties and the time period that you would like to train against.
- Add the periods of historical failures shown in the data (label data), if it exists.
- Train your ML model using AWS IoT SiteWise native anomaly detection.
- Setup your inference schedule to test your live data streams against your trained model.

Native anomaly detection monitors fixed and stationary industrial equipment that operates with limited variability in operating conditions. Supported equipment includes rotating machinery such as pumps, compressors, motors, computer numerical control (CNC) machines, and turbines. Process industry applications include heat exchangers, boilers, and inverters. Native anomaly detection is a back-end analytics service integrated into AWS IoT SiteWise and supplements existing maintenance systems.

Topics

- Review inference results
- Trigger custom actions on anomalous behavior (AWS Management Console)
- Best practices

Native anomaly detection features

The AWS IoT SiteWise platform offers a range of powerful features that can transform your industrial operations:

- **Predictive maintenance**: Detect failures early, and integrates with work order systems to act on insights.
- **Tight integration with AWS IoT SiteWise**: Leverage the tight integration into your existing infrastructure, eliminating the need to move your data to an external service.
- **Automatic model selection and training**: Benefit from this support from AWS IoT SiteWise, without requiring any machine learning expertise.
- **Customize your inference scheduling**: Schedule inference jobs to align with your operational needs and shift timings.
- **Labeled data**: Improve accuracy with known failure intervals during model training.
- **Model evaluation with pointwise diagnostics**: Evaluate model performance at the event level.
- **Scalable across assets**: Create computation models for individual assets or across multiple assets to achieve scalability.
- **Sensor-level insights**: Gain detailed diagnostics that identify the specific sensor(s) contributing to an anomaly.
- **Faster time to value**: Move from sensor modeling to anomaly detection in a matter of hours, not weeks.

Prerequisites

To complete these steps, you must have an asset model and at least one asset created. For more information, see Create an asset model (AWS CLI), and Create an asset (AWS CLI). We do not support external IDs at this time.

If you are new to AWS IoT SiteWise (and do not have historical data), you must call the CreateBulkImportJob API to import asset property values into AWS IoT SiteWise. This is used to train the model. For more information, see Create an AWS IoT SiteWise bulk import job (AWS CLI).

Setup AWS CLI for Computation Model APIs

Follow these steps to update your AWS CLI configuration, and access the computation model APIs:

- Install the latest awscli version `aws-cli`.
- Verify the installation by checking for the new APIs:

```
aws iotsitewise help
```

The command output displays the complete list of AWS IoT SiteWise APIs, including the newly added computation model operations.

Property requirements

To set up anomaly detection, you must have the following requirements and the UpdateAssetModel (AWS CLI):

- At least one input property that is of either `DOUBLE` or `INTEGER` data type. It is either a measurement or transform property, and is used to train the model.
- A result property of `STRING` data type. It must be a measurement property, and stores the anomaly detection results.

Labeling prerequisites

- Upload your data labels to an Amazon S3 bucket.
- Update the bucket policy of this bucket to allow AWS IoT SiteWise to read your labels.

  On console, go to **Permissions -> Bucket policy**. Paste the following policy and replace **bucket-arn** with ARN of your bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SiteWiseReadAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "iotsitewise.amazonaws.com"
```

```
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "bucket-arn",
                "bucket-arn/*"
            ]
        }
    ]
}
```

Model evaluation prerequisites

- Model evaluation generates pointwise model diagnostics in the Amazon S3 bucket location provided by you.
- In order for the pointwise diagnostic results to be written to your Amazon S3 bucket, update the bucket policy of this bucket to allow AWS IoT SiteWise to write the results.

  On console, go to `Permissions -> Bucket policy`. Paste the following policy and replace: `bucket-arn` with ARN of your bucket

```
{
      "Version": "2012-10-17",
      "Statement": [
          {
              "Sid": "SiteWiseWriteAccess",
              "Effect": "Allow",
              "Principal": {
                  "Service": "iotsitewise.amazonaws.com"
              },
              "Action": [
                  "s3:GetObject",
                  "s3:ListBucket",
                  "s3:PutObject"
              ],
              "Resource": [
                  "bucket-arn",
                  "bucket-arn/*"
              ]
```

```
        }
    ]
}
```

Enable anomaly detection on sensors of an asset

Create a computation model (AWS CLI)

To create a computation model, use the AWS Command Line Interface (AWS CLI). After you define the computation model, train the model, and schedule inference to do anomaly detection on an asset in AWS IoT SiteWise.

- Create a file `anomaly-detection-computation-model-payload.json` with the following content:

```
{
    "computationModelName": "anomaly-detection-computation-model-name",
    "computationModelConfiguration": {
        "anomalyDetection": {
            "inputProperties": "${input_properties}",
            "resultProperty": "${result_property}"
        }
    },
    "computationModelDataBinding": {
        "input_properties": {
            "list": [{
                    "assetModelProperty": {
                        "assetModelId": "asset-model-id",
                        "propertyId": "input-property-id-1"
                    }
                },
                {
                    "assetModelProperty": {
                        "assetModelId": "asset-model-id",
                        "propertyId": "input-property-id-2"
                    }
                }
            ]
        },
        "result_property": {
            "assetModelProperty": {
```

```
                "assetModelId": "asset-model-id",
                "propertyId": "results-property-id"
            }
        }
    }
}
```

- Run the following command to create a computation model:

```
aws iotsitewise create-computation-model \
    --cli-input-json file://anomaly-detection-computation-model-payload.json
```

ExecuteAction API payload preparation

The next steps to execute training and inference is performed with the ExecuteAction API. Both training and inference are configured with a JSON action payload configuration. When invoking the ExecuteAction API, the action payload must be provided as a value with a `stringValue` payload.

The payload must strictly adhere to the API requirements. Specifically, the value must be a **flat string**, with no **control characters** (for example, newlines, tabs, or carriage returns).

The following options provide two reliable ways to supply a valid action-payload:

Option 1: Use a clean payload file

The following procedure describes the steps for a clean payload file:

1. Clean the file to remove control characters.

```
tr -d '\n\r\t' < original-action-payload.json > training-or-inference-action-
payload.json
```

2. Execute the action with the file `@=file://....`

```
aws iotsitewise execute-action \
    --target-resource computationModelId=<MODEL_ID> \
    --action-definition-id <ACTION_DEFINITION_ID> \
    --resolve-to assetId=<ASSET_ID> \
    --action-payload stringValue@=file://training-or-inference-action-payload.json
```

Option 2: Inline string with escaped quotes

The following steps describes the steps to supply the payload inline, and avoid intermediary files:

- Use escaped double quotes (\") inside the JSON string.
- Wrap the entire `StringValue=..` expression within double quotes.

Example of an escaped action payload:

```
aws iotsitewise execute-action \
    --target-resource computationModelId=<MODEL_ID> \
    --action-definition-id <ACTION_DEFINITION_ID> \
    --resolve-to assetId=<ASSET_ID> \
    --action-payload "stringValue={\"exportDataStartTime\":1717225200,
\"exportDataEndTime\":1722789360,\"targetSamplingRate\":\"PT1M\"}"
```

Train the AWS CLI

With a computation model created, you can train a model against the assets. Follow the below steps to train a model for an asset:

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_TRAINING` action. Replace `computation-model-id` with the ID returned in the previous step.

   ```
   aws iotsitewise describe-computation-model \
       --computation-model-id computation-model-id
   ```

2. Create a file called `anomaly-detection-training-payload.json` and add the following values:

   > **Note**
   >
   > The payload must conform to Option 1: Use a clean payload file.

   a. `StartTime` with the start of the training data, provided in epoch seconds.

   b. `EndTime` with the end of the training data, provided in epoch seconds.

c. You can optionally configure [Advanced training configurations](), to improve the model performance.

   i. (Optional) `TargetSamplingRate` with the sampling rate of the data.

   ii. (Optional) `LabelInputConfiguration` to specify time periods when anomalous behavior occurred for improved model training.

   iii. (Optional) `ModelEvaluationConfiguration` to evaluate model performance by running inference on a specified time range after training completes.

```
{
  "exportDataStartTime": StartTime,
  "exportDataEndTime": EndTime
}
```

Example of a training payload example:

```
{
  "exportDataStartTime": 1717225200,
  "exportDataEndTime": 1722789360
}
```

3. Run the following command to start training. Replace the following parameters in the command:

   a. `computation-model-id` with the ID of the target computation model.

   b. `asset-id` with the ID of the asset against which you'll train the model.

   c. `training-action-definition-id` with the ID of the `AWS/ANOMALY_DETECTION_TRAINING` action from Step 1.

```
aws iotsitewise execute-action \
    --target-resource computationModelId=computation-model-id \
    --resolve-to assetId=asset-id \
    --action-definition-id training-action-definition-id \
    --action-payload stringValue@=file://anomaly-detection-training-payload.json
```

Example of an execute action:

```
aws iotsitewise execute-action --target-resource computationModelId=27cb824c-
fd84-45b0-946b-0a5b0466d890 --resolve-to assetId=cefd4b68-481b-4735-
b466-6a4220cd19ee --action-definition-id e54cea94-5d1c-4230-a59e-4f54dcbc972d --
action-payload stringValue@=file://anomaly-detection-training-payload.json
```

4. Run the following command to check for status of the model training process. The latest execution summary shows the execution status (`RUNNING`/`COMPLETED`/`FAILED`).

```
aws iotsitewise list-executions \
    --target-resource-type COMPUTATION_MODEL \
    --target-resource-id computation-model-id\
    --resolve-to-resource-type ASSET \
    --resolve-to-resource-id asset-id
```

5. Run the following command to check the configuration of the latest trained model. This command produces an output only if atleast one model was trained successfully.

```
aws iotsitewise describe-computation-model-execution-summary \
    --computation-model-id computation-model-id \
    --resolve-to-resource-type ASSET \
    --resolve-to-resource-id asset-id
```

6. When a `ComputationModel` is using AssetModelProperty, use the ListComputationModelResolveToResources API to identify the assets with executed actions.

```
aws iotsitewise list-computation-model-resolve-to-resources \
    --computation-model-id computation-model-id
```

Start and stop inference (AWS CLI)

After training the model, start the inference. This instructs AWS IoT SiteWise to actively monitor your industrial assets for anomalies.

Start inference

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_INFERENCE` action. Replace `computation-model-id` with the actual ID of computation model created earlier.

```
aws iotsitewise describe-computation-model \
    --computation-model-id computation-model-id
```

2. Create a file `anomaly-detection-start-inference-payload.json` and add the following values:

> **Note**
>
> The payload must conform to Option 1: Use a clean payload file.

```
"inferenceMode": "START",
"dataUploadFrequency": "DataUploadFrequency"
```

   a. `DataUploadFrequency`: Configure the frequency at which the inference schedule runs to perform anomaly detection. Allowed values are: `PT5M, PT10M, PT15M, PT30M, PT1H, PT2H..PT12H, PT1D`.

   b. (Optional) `DataDelayOffsetInMinutes` with the delay offset in minutes. Set this value between 0 and 60 minutes.

   c. (Optional) `TargetModelVersion` with the model version to activate.

   d. (Optional) Configure the `weeklyOperatingWindow` with a shift configuration.

   e. You can optionally configure Advanced inference configurations.

      i. High frequency inferencing (5 minutes – 1 hour).

      ii. Low frequency inferencing (2 hours – 1 day).

      iii. Flexible scheduling.

3. Run the following command to start inference. Replace the following parameters in the payload file.

   a. `computation-model-id` with the ID of the target computation model.

b. `asset-id` with the ID of the asset against which the model was trained.

c. `inference-action-definition-id` with the ID of the `AWS/ANOMALY_DETECTION_INFERENCE` action from Step 1.

```
aws iotsitewise execute-action \
    --target-resource computationModelId=computation-model-id \
    --resolve-to assetId=asset-id \
    --action-definition-id inference-action-definition-id \
    --action-payload stringValue@=file://anomaly-detection-inference-payload.json
```

4. Run the following command to check if inference is still running. The `inferenceTimerActive` field is set to `TRUE` when inference is active.

```
aws iotsitewise describe-computation-model-execution-summary \
    --computation-model-id computation-model-id \
    --resolve-to-resource-type ASSET \
    --resolve-to-resource-id asset-id
```

5. The following command lists all the inference executions:

```
aws iotsitewise list-executions \
   --target-resource-type COMPUTATION_MODEL \
   --target-resource-id computation-model-id \
   --resolve-to-resource-type ASSET \
   --resolve-to-resource-id asset-id
```

6. Run the following command to describe an individual execution. Replace `execution-id` with the id from previous Step 5.

```
aws iotsitewise describe-execution \
    --execution-id execution-id
```

Stop inference

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_INFERENCE` action. Replace `computation-model-id` with the actual ID of computation model created earlier.

```
aws iotsitewise describe-computation-model \
    --computation-model-id computation-model-id
```

2. Create a file `anomaly-detection-stop-inference-payload.json` and add the following code.

```
{
    "inferenceMode": "STOP"
}
```

> **Note**
>
> The payload must conform to Option 1: Use a clean payload file.

3. Run the following command to stop inference. Replace the following parameter in the payload file:

   a. `computation-model-id` with the ID of the target computation model.

   b. `asset-id` with the ID of the asset against which the model was trained.

   c. `inference-action-definition-id` with the ID of the `AWS/ANOMALY_DETECTION_INFERENCE` action from Step 1.

   Example of the stop inference command:

```
aws iotsitewise execute-action \
    --target-resource computationModelId=computation-model-id \
    --resolve-to assetId=asset-id \
    --action-definition-id inference-action-definition-id \
    --action-payload stringValue@=file://anomaly-detection-stop-inference-
payload.json
```

Find computation models that uses a given resource in data binding

To list computation models which are bound to a given resource:

- **asset model** (fetch all computation models where any of this asset model's properties are bound).

- **asset** (fetch all computation models where any of this asset's properties are bound)
- **asset model property** (fetch all computation models where this property is bound)
- **asset property** (fetch all computation models where this property is bound. This could be for informational purposes, or required when user tries to bind this property to another computation model but it is already bound somewhere else)

Use [ListComputationModelDataBindingUsages](#) API to fetch a list of `ComputationModelIds` that take the asset (property) or asset model (property) as data binding.

Prepare a `request.json` with the following information:

```
{
  "dataBindingValueFilter": {
    "asset": {
      "assetId": "<string>"
    }
    // OR
    "assetModel": {
      "assetModelId": "<string>"
    }
    // OR
    "assetProperty": {
      "assetId": "<string>",
      "propertyId": "<string>"
    }
    // OR
    "assetModelProperty": {
      "assetModelId": "<string>",
      "propertyId": "<string>"
    }
  },
  "nextToken": "<string>",
  "maxResults": "<number>"
}
```

Use the `list-computation-model-data-binding-usages` command to retrieve the models with assets or asset models as data bindings.

```
aws iotsitewise list-computation-model-data-binding-usages \
--cli-input-json file://request.json
```

Enable anomaly detection on sensors across assets

Create a computation model (AWS CLI)

To create a computation model, use the AWS Command Line Interface (AWS CLI). After you define the computation model, train the model and schedule inference to do anomaly detection across assets in AWS IoT SiteWise.

The following steps explain this process:

1. To set up anomaly detection, use the UpdateAssetModel (AWS CLI), and meet the following requirements:

   a. At least one input property that is of either `DOUBLE` or `INTEGER` data type. It is either a measurement or transform property, and is used to train the model.

   b. A result property of `STRING` data type. It must be a measurement property, and stores the anomaly detection results.

2. Create a file `anomaly-detection-computation-model-payload.json` with the following content:

   Note

   Create a computation model by directly providing `assetProperty` as the data source.

```
{
    "computationModelName": "name of ComputationModel",
    "computationModelConfiguration": {
        "anomalyDetection": {
            "inputProperties": "${properties}",
            "resultProperty": "${p3}"
        }
    },
    "computationModelDataBinding": {
        "properties": {
            "list": [
                {
                    "assetProperty": {
                        "assetId": "asset-id",
                        "propertyId": "input-property-id-1"
```

```
                    }
                },
                {
                    "assetProperty": {
                        "assetId": "asset-id",
                        "propertyId": "input-property-id-2"
                    }
                }
            ]
        },
        "p3": {
            "assetProperty": {
                "assetId": "asset-id",
                "propertyId": "results-property-id"
            }
        }
    }
}
```

3. Run the following command to create a computation model:

```
aws iotsitewise create-computation-model \
    --cli-input-json file://anomaly-detection-computation-model-payload.json
```

ExecuteAction API payload preparation

The next steps to execute training and inference is performed with the ExecuteAction API. Both training and inference are configured with a JSON action payload configuration. When invoking the ExecuteAction API, the action payload must be provided as a value with a `stringValue` payload.

The payload must strictly adhere to the API requirements. Specifically, the value must be a **flat string** with no **control characters** (for example, newlines, tabs, or carriage returns). The following options provides two reliable ways to supply a valid action-payload.

Option 1: Use a clean payload file

The following procedure describes the steps for a clean payload file:

1. Clean the file to remove control characters.

```
tr -d '\n\r\t' < original-action-payload.json > training-or-inference-action-
payload.json
```

2. Execute the action with the file `@=file://....`

```
aws iotsitewise execute-action \
    --target-resource computationModelId=<MODEL_ID> \
    --action-definition-id <ACTION_DEFINITION_ID> \
    --action-payload stringValue@=file://training-or-inference-action-payload.json
```

Option 2: Inline string with escaped quotes

The following steps describes the steps to supply the payload inline, and avoid intermediary files:

- Use escaped double quotes (\") inside the JSON string.
- Wrap the entire `StringValue=..` expression within double quotes.

Example of an escaped action payload:

```
aws iotsitewise execute-action \
    --target-resource computationModelId=<MODEL_ID> \
    --action-definition-id <ACTION_DEFINITION_ID> \
    --action-payload "stringValue={\"exportDataStartTime\":1717225200,
\"exportDataEndTime\":1722789360,\"targetSamplingRate\":\"PT1M\"}"
```

Train the AWS CLI

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_TRAINING` action. Replace `computation-model-id` with the ID returned in the previous step.

```
aws iotsitewise describe-computation-model \
    --computation-model-id computation-model-id
```

2. Create a file called `anomaly-detection-training-payload.json` and add the following values:

> **Note**
>
> The payload must conform to Option 1: Use a clean payload file.

a. `StartTime` with the start of the training data, provided in epoch seconds.

b. `EndTime` with the end of the training data, provided in epoch seconds.

c. You can optionally configure Advanced inference configurations.

   i. (Optional) `TargetSamplingRate` with the sampling rate of the data.

   ii. (Optional) `LabelInputConfiguration` to specify time periods when anomalous behavior occurred for improved model training.

   iii. (Optional) `ModelEvaluationConfiguration` to evaluate model performance by running inference on a specified time range after training completes.

```
{
  "exportDataStartTime": StartTime,
  "exportDataEndTime": EndTime
}
```

Example of a training payload example:

```
{
  "exportDataStartTime": 1717225200,
  "exportDataEndTime": 1722789360
}
```

3. Run the following command to start training (without providing asset as a target resource). Replace the following parameters in the command:

```
aws iotsitewise execute-action \
    --target-resource computationModelId=computation-model-id \
    --action-definition-id training-action-definition-id \
    --action-payload stringValue@=file://anomaly-detection-training-payload.json
```

4. Run the following command to check for status of the model training process. The latest execution summary shows the execution status (`RUNNING`/`COMPLETED`/`FAILED`).

```
aws iotsitewise list-executions \
    --target-resource-type COMPUTATION_MODEL \
    --target-resource-id computation-model-id
```

5. Run the following command to check the configuration of the latest trained model. This command produces an output only if at least one model has completed training successfully.

```
aws iotsitewise describe-computation-model-execution-summary \
    --computation-model-id computation-model-id
```

Start and stop inference (AWS CLI)

After training the model, start the inference, which instructs AWS IoT SiteWise to begin monitoring your industrial assets for anomalies.

Start inference

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_INFERENCE` action. Replace `computation-model-id` with the actual ID of computation model created earlier.

```
aws iotsitewise describe-computation-model \
    --computation-model-id computation-model-id
```

2. Create a file `anomaly-detection-start-inference-payload.json` and add the following code. Replace the following parameters as described:

> **Note**
>
> The payload must conform to Option 1: Use a clean payload file.

   a. `DataUploadFrequency`: Configure the frequency at which the inference schedule runs to perform anomaly detection. Allowed values are: `PT5M, PT10M, PT15M, PT30M, PT1H, PT2H..PT12H, PT1D`.

```
"inferenceMode": "START",
```

```
"dataUploadFrequency": "DataUploadFrequency"
```

b. (Optional) `DataDelayOffsetInMinutes` with the delay offset in minutes. Set this value between 0 and 60 minutes.

c. (Optional) `TargetModelVersion` with the model version to activate.

d. (Optional) Configure the `weeklyOperatingWindow` with a shift configuration.

e. You can optionally configure Advanced inference configurations.

   i. High frequency inferencing (5 minutes – 1 hour).

   ii. Low frequency inferencing (2 hours – 1 day).

   iii. Flexible scheduling.

3. Run the following command to start inference. Replace the following parameters in the payload file.

   a. `computation-model-id` with the ID of the target computation model.

   b. `inference-action-definition-id` with the ID of the `AWS/ANOMALY_DETECTION_INFERENCE` action from Step 1.

```
aws iotsitewise execute-action \
    --target-resource computationModelId=computation-model-id \
    --action-definition-id inference-action-definition-id \
    --action-payload stringValue@=file://anomaly-detection-inference-payload.json
```

4. Run the following command to check if inference is still running. The `inferenceTimerActive` field is set to `TRUE` when inference is active.

```
aws iotsitewise describe-computation-model-execution-summary \
    --computation-model-id computation-model-id
```

5. The following command lists all the inference executions:

```
aws iotsitewise list-executions \
    --target-resource-type COMPUTATION_MODEL \
    --target-resource-id computation-model-id
```

6. Run the following command to describe an individual execution. Replace `execution-id` with the id from previous Step 5.

```
aws iotsitewise describe-execution \
    --execution-id execution-id
```

Stop inference

1. Run the following command to find the `actionDefinitionId` of the `AWS/ANOMALY_DETECTION_INFERENCE` action. Replace `computation-model-id` with the actual ID of computation model created earlier.

```
aws iotsitewise describe-computation-model \
    --computation-model-id computation-model-id
```

2. Create a file `anomaly-detection-stop-inference-payload.json` and add the following code.

```
{
    "inferenceMode": "STOP"
}
```

> **Note**
>
> The payload must conform to Option 1: Use a clean payload file.

3. Run the following command to stop inference. Replace the following parameter in the payload file:

   a. `computation-model-id` with the ID of the target computation model.

   b. `inference-action-definition-id` with the ID of the `AWS/ANOMALY_DETECTION_INFERENCE` action from Step 1.

Example of the stop inference command:

```
aws iotsitewise execute-action \
--target-resource computationModelId=computation-model-id \
--action-definition-id inference-action-definition-id \
--action-payload stringValue@=file://anomaly-detection-stop-inference-payload.json
```

Advanced training configurations

Sample rate configuration

The **sample rate** defines how frequently sensor readings are recorded (for example, once every second, or once every minute). This setting directly impacts the **granularity** of the training data, and influences the model's ability to capture short-term variations in sensor behavior.

Visit Sampling for high-frequency data and consistency between training and inference to learn about best practices.

Configure target sampling rate

You can optionally specify a `TargetSamplingRate` in your training configuration, to control the frequency at which data is sampled. Supported values are:

```
PT1S | PT5S | PT10S | PT15S | PT30S | PT1M | PT5M | PT10M | PT15M | PT30M | PT1H
```

These are ISO 8601 duration formats, representing the following time formats:

- `PT1S` = 1 second
- `PT1M` = 1 minute
- `PT1H` = 1 hour

Choose a sampling rate that strikes the right balance between **data resolution**, and **training efficiency**. The following rates are available:

- **Higher sampling rates** (`PT1S`) offer finer detail but may increase data volume and training time.
- **Lower sampling rates** (`PT10M`, `PT1H`) reduce data size and cost but may miss short-lived anomalies.

Handling timestamp misalignment

AWS IoT SiteWise automatically compensates for **timestamp misalignment** across multiple data streams during training. This ensures consistent model behavior even if input signals are not perfectly aligned in time.

Visit Sampling for high-frequency data and consistency between training and inference to learn about best practices.

Enable sampling

Add the following code to `anomaly-detection-training-payload.json`.

Configure sampling by adding `TargetSamplingRate` in the training action payload, with the sampling rate of the data. The allowed values are: `PT1S | PT5S | PT10S | PT15S | PT30S | PT1M | PT5M | PT10M | PT15M | PT30M | PT1H`.

```
{
    "exportDataStartTime": StartTime,
    "exportDataEndTime": EndTime,
    "targetSamplingRate": "TargetSamplingRate"
}
```

Example of a sample rate configuration:

```
{
    "exportDataStartTime": 1717225200,
    "exportDataEndTime": 1722789360,
    "targetSamplingRate": "PT1M"
}
```

Label your data

When labeling your data, you must define time intervals that represent periods of abnormal equipment behavior. This labeling information is provided as a CSV file, where each row specifies a time range during which the equipment was not operating correctly.

Each row contains two timestamps:

- The **start time**, indicating when abnormal behavior is believed to have begun.
- The **end time**, representing when the failure or issue was first observed.

This CSV file is stored in an Amazon S3 bucket and is used during model training to help the system learn from known examples of abnormal behavior. The following example shows how your label data should appear as a `.csv` file. The file has no header.

Example of a CSV file:

```
2024-06-21T00:00:00.000000,2024-06-21T12:00:00.000000
2024-07-11T00:00:00.000000,2024-07-11T12:00:00.000000
2024-07-31T00:00:00.000000,2024-07-31T12:00:00.000000
```

Row 1 represents a maintenance event on **June 21, 2024**, with a **12-hour window** (from `2024-06-21T00:00:00.000000Z` to `2024-06-21T12:00:00.000000Z`) for AWS IoT SiteWise to look for abnormal behavior.

Row 2 represents a maintenance event on **July 11, 2024**, with a **12-hour window** (from `2024-07-11T00:00:00.000000Z` to `2024-07-11T12:00:00.000000Z`) for AWS IoT SiteWise to look for abnormal behavior.

Row 3 represents a maintenance event on **July 31, 2024**, with a **12-hour window** (from `2024-07-31T00:00:00.000000Z` to `2024-07-31T12:00:00.000000Z`) for AWS IoT SiteWise to look for abnormal behavior.

AWS IoT SiteWise uses all of these time windows to train and evaluate models that can identify abnormal behavior around these events. Note that not all events are detectable, and results are highly dependent on the quality and characteristics of the underlying data.

For details about best practices for sampling, see Best practices.

Data labeling steps

- Configure your Amazon S3 bucket according to the labeling prerequisites at Labeling data prerequisites.
- Upload the file to your labeling bucket.
- Add the following to `anomaly-detection-training-payload.json`.
  - Provide the locations in the `labelInputConfiguration` section of the file. Replace `labels-bucket` with bucket name and `files-prefix` with file(s) path or any part of prefix. All files at the location are parsed, and (on success) used as label files.

```
{
    "exportDataStartTime": StartTime,
    "exportDataEndTime": EndTime,
    "labelInputConfiguration":
```

```
        {
         "bucketName": "label-bucket",
         "prefix": "files-prefix"
        }
}
```

Example of a label configuration:

```
{
    "exportDataStartTime": 1717225200,
    "exportDataEndTime": 1722789360,
    "labelInputConfiguration": {
      "bucketName": "anomaly-detection-customer-data-278129555252-iad",
      "prefix": "Labels/model=b2d8ab3e-73af-48d8-9b8f-a290bef931b4/
asset[d3347728-4796-4c5c-afdb-ea2f551ffe7a]/Lables.csv"
    }
}
```

Evaluate your model

Pointwise model diagnostics for an AWS IoT SiteWise training model is an evaluation of the model performance at the individual events. During training, AWS IoT SiteWise generates an anomaly score, and sensor contribution diagnostics for each row in the input dataset. A higher anomaly score indicates a higher likelihood of an abnormal event.

Pointwise diagnostics are available, when you train a model with ExecuteAction API, and `AWS/ANOMALY_DETECTION_TRAINING` action type.

To configure model evaluation,

- Configure your Amazon S3 bucket according to the labelling prerequisites at Labeling data prerequisites.
- Add the following to `anomaly-detection-training-payload.json`.
  - Provide the `evaluationStartTime` and `evaluationEndTime` (both in epoch seconds) for the data in the window used to evaluate the performance of the model.
  - Provide the Amazon S3 bucket location (`resultDestination`) in order for the the evaluation diagnostics to be written to.

Note

The model evaluation interval (`dataStartTime` to `dataEndtime`) must either overlap, or be contiguous to the training interval. No gaps are permitted.

```
{
  "exportDataStartTime": StartTime,
  "exportDataEndTime": EndTime,
  "modelEvaluationConfiguration": {
    "dataStartTime": evaluationStartTime,
    "dataEndTime": evaluationEndTime
    "resultDestination": {
      "bucketName": "s3BucketName",
      "prefix": "bucketPrefix"
    }
  }
}
```

Example of a model evaluation configuration:

```
{
  "exportDataStartTime": 1717225200,
  "exportDataEndTime": 1722789360,
  "modelEvaluationConfiguration": {
    "dataStartTime": 1722789360,
    "dataEndTime": 1725174000,
    "resultDestination": {
      "bucketName": "anomaly-detection-customer-data-278129555252-iad",
      "prefix": "Evaluation/asset[d3347728-4796-4c5c-afdb-ea2f551ffe7a]/1747681026-
evaluation_results.jsonl"
    }
  }
}
```

Advanced inference configurations

AWS IoT SiteWise allows customers to configure model inference schedules tailored to their operational needs.

Inference scheduling is broadly categorized into three modes:

- High frequency inferencing (5 minutes – 1 hour)
- Low frequency inferencing (2 hours – 1 day)
- Flexible scheduling

High frequency inferencing (5 minutes – 1 hour)

This mode is ideal for processes that operate continuously, or have a high rate of change in sensor values. In this configuration, inference runs frequently as often as every 5 minutes.

Use cases:

- It is used in monitoring fast-changing equipment like compressors or conveyors.
- It is helpful in catching short-lived anomalies that require immediate response.
- It's an always-on operation where data is consistently flowing.

Conditional offset support:

You can define a **conditional offset** (0 - 60 minutes) to delay inference after data ingestion. This ensures late-arriving data is still included in the analysis window.

To configure high frequency inferencing:

- Configure `AWS/ANOMALY_DETECTION_INFERENCE` action payload value with `DataUploadFrequency` with values: `PT5M, PT10M, PT15M, PT30M, PT1H` while starting inference.
- (Optional) Configure `DataDelayOffsetInMinutes` with the delay offset in minutes. Set this value between 0 and 60 minutes.

```
{
```

```
    "inferenceMode": "START",
    "dataDelayOffsetInMinutes": "DataDelayOffsetInMinutes",
    "dataUploadFrequency": "DataUploadFrequency"
}
```

Example of high frequency inference configuration:

```
{
    "inferenceMode": "START",
    "dataDelayOffsetInMinutes": "2",
    "dataUploadFrequency": "PT5M"
}
```

Low frequency inferencing (2 hours – 1 day)

This mode is suited for slower-moving processes or use cases where daily evaluations are sufficient. Customers configure inference to run hourly or once per day.

Start time support for 1-day interval:

For daily inference, optionally specify a **`startTime`** (8 AM every day), along with timezone awareness.

Timezone support:

When a `startTime` is provided, AWS IoT SiteWise uses Time Zone Database, maintained by the Internet Assigned Numbers Authority (IANA). This ensures your inference aligns with local working hours even across regions.

Conditional offset support:

As with other modes, a conditional offset of 0 – 60 minutes is configured.

Use cases:

- Daily health checks for batch processes or shift-based operations.
- Avoids inference during maintenance or downtime.
- It's helpful in resource-constrained environments, where compute usage must be minimized.

To configure low frequency inferencing:

- Configure `AWS/ANOMALY_DETECTION_INFERENCE` action payload value with `DataUploadFrequency` with values: `PT2H..PT12H`.
  - In the case of 1 day, `DataUploadFrequency` is `P1D`.
- (Optional) Configure `DataDelayOffsetInMinutes` with the delay offset in minutes. Set this value between 0 and 60 minutes.

Example of low frequency inference configuration:

```
{
    "inferenceMode": "START",
    "dataUploadFrequency": "P1D",
    "inferenceStartTime": "13:00",
    "inferenceTimeZone": "America/Chicago"
}
```

Flexible scheduling

Flexible scheduling allows customers to define **specific days and time ranges**, during which inference is run. This gives customers complete control over scheduling based on production hours, shift timings, and planned downtimes.

The `weeklyOperatingWindow` helps when:

- The equipment runs only during specific hours (8 AM – 4 PM).
- There is no production on weekends.
- Daily maintenance is scheduled during known time blocks.

Timezone support:

When a `startTime` is provided, AWS IoT SiteWise uses [Time Zone Database](), maintained by the Internet Assigned Numbers Authority (IANA). This ensures the inference aligns with local working hours even across regions.

Conditional offset support:

As with other modes, a conditional offset of 0 – 60 minutes can be configured.

Benefits of `weeklyOperatingWindow`:

- It avoids inference during idle or maintenance periods, reducing false positives.
- It aligns anomaly detection with operational priorities and shift-based workflows.

To configure flexible scheduling:

- Configure `AWS/ANOMALY_DETECTION_INFERENCE` action payload value with `DataUploadFrequency`.
- (Optional) `DataDelayOffsetInMinutes` with the delay offset in minutes. Set this value between 0 and 60 minutes.
- Configure `weeklyOperatingWindow` with a shift configuration:
  - Keys for the `weeklyOperatingWindow` are days of the week: `monday|tuesday|wednesday|thursday|friday|saturday|sunday`.
  - Each time range must be in **24-hour format** as `"HH:MM-HH:MM"` (`"08:00-16:00"`).
  - Multiple ranges can be specified per day.

Example of flexible scheduling configuration:

```
{
    "inferenceMode": "START",
    "dataUploadFrequency": "PT5M",
    "weeklyOperatingWindow": {
      "tuesday": ["11:00-13:00"],
      "monday": ["10:00-11:00", "13:00-15:00"]
    }
}
```

Model version activation

When starting inference, you can optionally activate a specific model version to use for anomaly detection. This feature allows you to select a particular trained model version, roll back to previous versions, or override automatic model promotion decisions.

Use cases:

- **Production rollback**: Quickly revert to a stable model version when the current version shows degraded performance or unexpected behavior.

- **A/B testing**: Compare performance between different model versions by switching between them during controlled time periods.
- **Manual model selection**: Override automatic promotion decisions, and manually select your preferred model version based on business requirements.
- **Staged deployment**: Test newer model versions in non-critical time windows before promoting them to full production use.
- **Performance optimization**: Select model versions that perform better for specific operational conditions, or seasonal patterns.
- **Rollback during maintenance**: Use older, well-tested model versions during system maintenance, or upgrades to ensure stability.

Model version selection behavior

When `targetModelVersion` is specified:

- The system activates the requested model version for inference.
- Validates that the specified model version exists.
- Overrides any automatic promotion settings.

When `targetModelVersion` is not specified:

- Activates the last active model version if inference was previously started.
- If inference was never activated, uses the latest trained model version.

To activate a specific model version:

- Configure the inference action payload, with targetModelVersion set to your desired model version number.
- The specified model version is validated and activated if it exists.

Example of model version activation:

```
{
    "inferenceMode": "START",
    "dataUploadFrequency": "PT15M",
    "targetModelVersion": 2
```

```
}
```

Checking model versions

To verify the currently active model version:

- Use the [DescribeComputationModelExecutionSummary] API, which includes the active model version in the response.

To view all available model versions:

- Use the ListExecutions API to retrieve a complete list of historical model versions.
- Use the Use the DescribeExecution API to retrieve trained model information including export data time range, computation model version, and billable duration in minutes.

Model version characteristics

- Model version numbers are assigned sequentially starting from 1.
- You can activate any previously trained model versions.
- The activated model version persists until explicitly changed.
- Model version activation works with all inference scheduling modes (high-frequency, low-frequency, and flexible).
- If the specified model version doesn't exist, the inference action fails with an error.

Review inference results

Retrieve inference results

Latest inference results

Run the following command to fetch the most recent inference result for an asset property. For more information, see get-asset-property-value in the *AWS CLI Command Reference Guide*.

```
aws iotsitewise get-asset-property-value \
    —asset-id asset-id \
    —property-id result-property-id
```

Inference results history

Run the following command to fetch the history of inference results for a specified time window. For more information, see get-asset-property-value-history in the *AWS CLI Command Reference Guide*.

```
aws iotsitewise get-asset-property-value-history \
    --asset-id asset-id \
    --property-id result-property-id \
    --start-date start-time \
    --end-date end-time
```

Example response

Example of an inference result response:

```
{
  "value": {
    "stringValue": "{\"timestamp\": \"2025-02-10T22:42:00.000000\", \"prediction\": 0,
 \"prediction_reason\": \"NO_ANOMALY_DETECTED\", \"diagnostics\": [{\"name\": \"asset-
id\\\\property-id\", \"value\": 0.53528}]}"
  },
  "timestamp": {
    "timeInSeconds": 1739227320,
    "offsetInNanos": 0
  },
  "quality": "GOOD"
}
```

Response fields

- **value.stringValue** – A JSON string containing the inference result with the following fields:
  - **timestamp** – The timestamp of the TQV against which inference is performed.
  - **prediction** – The prediction result (0 for no anomaly, 1 for anomaly detected).
  - **prediction_reason** – The reason for the prediction (`NO_ANOMALY_DETECTED` or `ANOMALY_DETECTED`).
  - **diagnostics** – An array of diagnostic information showing contributing factors.
- **timestamp** – The timestamp when the result is recorded in AWS IoT SiteWise.
- **quality** – The quality of the data point (typically `GOOD`).

Understand inference results

An inference result returned by AWS IoT SiteWise anomaly detection includes key information about the model's prediction at a specific timestamp, including whether an anomaly was detected and which sensors contributed to the anomaly.

Example of a detailed inference result:

```
{
    "timestamp": "2021-03-11T22:25:00.000000",
    "prediction": 1,
    "prediction_reason": "ANOMALY_DETECTED",
    "anomaly_score": 0.72385,
    "diagnostics": [
      { "name": "asset_id_1\\\\property_id_1", "value": 0.02346 },
      { "name": "asset_id_2\\\\property_id_2", "value": 0.10011 },
      { "name": "asset_id_3\\\\property_id_3", "value": 0.11162 }
    ]
}
```

The `diagnostics` field is useful for interpreting *why* the model makes a certain prediction. Each entry in the list includes:

- `name`: The sensor that contributed to the prediction (format: `asset_id\\\\property_id`).
- `value`: A floating-point number between 0 and 1, representing the **relative weight or importance**, of that sensor at that point in time.

User benefits:

- Understand which sensors had the strongest impact on an anomaly.
- Correlate high-weight sensors with physical equipment behavior.
- Inform root cause analysis.

Note

Even when `prediction = 0` (normal behavior), the diagnostics list is returned. This helps assess which sensors are currently influencing the model's decisions, even in healthy states.

Trigger custom actions on anomalous behavior (AWS Management Console)

You can enable **custom actions in response to anomalous behavior** by using **AWS IoT SiteWise MQTT notifications** in combination with **AWS IoT Core**.

Follow these steps to configure MQTT notifications in AWS IoT SiteWise, and trigger a custom action in AWS IoT Core based on inference results:

- Locate the asset in AWS IoT SiteWise, where the inference runs.
- Identify the property you used as the `resultProperty` when creating the computation model. Enable **MQTT Notification** for this property.
- Once you enable **MQTT Notification**, copy the **Notification Topic** that AWS IoT SiteWise generates.
- Navigate to AWS IoT Core. Under MQTT test client, subscribe to the copied **Notification Topic** to monitor the incoming messages.
- Create an **AWS IoT Core Rule** to **Process Inference Results**. (This ensures that actions are triggered only if the system detects an anomaly). Replace `notification-topic` with the **Notification Topic** that AWS IoT SiteWise generates.

```
SELECT * FROM "notification-topic"
  WHERE indexof(get(get(payload.values, 0).value, 'stringValue'),
 "NO_ANOMALY_DETECTED") < 0
```

Configure the rule to trigger any of the actions that AWS IoT Core supports. Learn more about the actions supported by AWS IoT Core.

Best practices

Understand the minimum date range

Use a minimum of 14 days for training data duration. However, we recommend that you include a longer period of data in many cases.

Ensure that your training dataset spans a timeframe during which the asset operated under all of its normal operating modes. This approach helps AWS IoT SiteWise accurately distinguish between expected behavior and true anomalies.

If your training data doesn't represent all typical operating modes, AWS IoT SiteWise might incorrectly flag unfamiliar but normal patterns as anomalies, which increases false positives.

Sampling for high-frequency data and consistency between training and inference

If your sensors generate data at a frequency higher than 1 Hz (more than one reading per second), apply sampling during training. Sampling reduces data volume while preserving essential trends, which enables efficient processing and improves model generalization by minimizing the impact of noise or transient fluctuations.

AWS IoT SiteWise native anomaly detection currently doesn't support data ingested at rates below 1 Hz. Verify that your data meets this minimum frequency requirement before you configure anomaly detection.

Additionally, AWS IoT SiteWise uses the sampling rate that you configure during training for inference as well. To maintain consistency and ensure accurate anomaly detection results, choose a sampling rate that aligns with both your operational needs and the behavior of your sensor data.

Find more details about how to set sampling rate at Sample rate configuration.

Labeling recommendations

Accurate and consistent labeling of anomalies is essential for effective model evaluation and continuous improvement. Consider the following best practices when you label anomalies:

- **Consolidate related anomalies:** Don't label closely occurring anomalies as separate events, if they're part of the same underlying issue. For example, if anomalies occur within 1–2 days of each other and the same root cause drives them, treat them as a single anomaly window. This approach helps the model better learn the pattern of abnormal behavior, and reduces noise in your evaluation data.
- **Label anomaly windows, not just points:** Instead of marking individual data points as anomalous, label the entire window that reflects abnormal behavior from deviation onset to recovery. This approach provides clearer boundaries and improves model alignment with actual operational issues.

- **Exclude uncertain periods:** If you're unsure whether a period is anomalous, leave it unlabeled. Ambiguous labels can confuse the model and degrade its accuracy over time.

Find more details about how to add labels at Label your data.

Manage data storage in AWS IoT SiteWise

You can configure AWS IoT SiteWise to save your data in the following storage tiers:

Hot tier

The hot storage tier is an AWS IoT SiteWise managed time series storage. Hot tier is most effective for frequently accessed data, with low write-to-read latency. Data stored in the hot tier is used by industrial applications that need quick access to the latest values of measurements in your equipment. This includes applications that visualize real-time metrics with an interactive dashboard, or applications that monitor operations and launch alarms to identify performance issues.

By default, data ingested into AWS IoT SiteWise is stored in the hot tier. You can define a retention period for the hot tier, after which AWS IoT SiteWise moves data in the hot tier to either warm or cold tier storage, based on your configuration. For best performance and cost efficiency, set your hot tier retention period to be longer than the time taken to retrieve data often. This is used for real time metrics, alarms, and monitoring scenarios. If a retention period is not set, your data is stored indefinitely in the hot tier.

Warm tier

The warm storage tier is an AWS IoT SiteWise managed tier that's effective for cost-efficient storage of historical data. It's best used to retrieve large volumes of data with medium write-to-read latency characteristics. Use the warm tier to store historical data that's needed for large workloads. For example, it's used for data retrieval for analytics, business intelligence applications (BI), reporting tools, and training of machine learning (ML) models. If you enable the cold storage tier, you can define a warm tier retention period. After the retention period ends, AWS IoT SiteWise deletes data from the warm tier.

Cold tier

The cold storage tier uses an Amazon S3 bucket to store data that's rarely used. With cold tier enabled, AWS IoT SiteWise replicates the time series, including measurements, metrics, transforms and aggregates, and asset model definitions every 6 hours. Cold tier is used to store data that tolerates high read latency for historical reports and backups.

Topics

- Configure storage settings in AWS IoT SiteWise
- Troubleshoot storage settings for AWS IoT SiteWise
- File paths and schemas of data saved in the cold tier

Configure storage settings in AWS IoT SiteWise

You can configure storage settings to opt in to service managed warm tier storage, and also to replicate data to the cold tier. To learn more about the retention period for the warm and hot tier, see Data retention impact. While configuring the storage settings, do the following:

- **Hot tier retention** — Set a retention period for how long your data is stored in the hot tier before it's deleted, and moved to the service managed warm tier storage or cold tier storage based on your storage settings. AWS IoT SiteWise will delete any data in the hot tier that existed before the retention period ends. If you don't set a retention period, your data is stored indefinitely in the hot tier.
- **Warm tier retention** — Set a retention period for how long your data is stored in the warm tier before it's deleted from AWS IoT SiteWise storage and moved to the customer managed cold tier storage. AWS IoT SiteWise deletes any data from the warm tier that existed before the retention period ends. If a retention period is not set, your data is stored indefinitely in the warm tier.

> **Note**
>
> To improve query performance, set a hot tier retention period with warm tier storage.

Impact of data retention in hot and warm tier storage

- When you decrease the retention period of the hot tier storage, data is permanently moved from the hot tier to the warm or cold tier. When you decrease the retention period of the warm tier, data is moved to the cold tier, and permanently deleted from the warm tier.
- When you increase the retention period of the hot or warm tier storage, the change affects data that's sent to AWS IoT SiteWise from then on. AWS IoT SiteWise does not retrieve data from the warm or cold storage to populate the hot tier. For example, if the retention period of the hot tier storage is initially set for 30 days and then increased to 60 days, it takes 30 days for the hot tier storage to contain 60 days worth of data.

Topics

Configure storage settings for warm tier (console)

The following procedure shows you how to configure the storage settings to replicate data to the warm tier 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:
5. For **Hot tier settings**, do the following:
   - If you want to set a retention period for how long your data is stored in the hot tier before it's deleted, and moved to the service managed warm tier storage, choose **Enable retention period**.
   - To configure a retention period, enter a whole number and choose a unit. The retention period must be greater than or equal to 30 days.

   AWS IoT SiteWise deletes any data in the hot tier that's older than the retention period. If you don't set a retention period, your data is stored indefinitely.
6. (Recommended) For **Warm tier settings**, do the following:
   - To opt in to warm tier storage, select **I confirm to the opt-in of warm tier storage** to opt in for the warm tier storage.
   - (Optional) To configure a retention period, enter a whole number and choose a unit. The retention period must be greater than or equal to 365 days.

AWS IoT SiteWise deletes data in the warm tier that existed earlier than the retention period. If you don't set a retention period, your data is stored indefinitely.

> **Note**
>
> - When you opt in for warm tier, the configuration displays once only.
> - To set hot tier retention, you must have either warm or cold tier storage. For cost efficiency and historical data retrieval, AWS IoT SiteWise recommends that you store long term data in the warm tier.
> - To set warm tier retention, you must have cold tier storage.

7. Choose **Save** to save your storage settings.

In the **AWS IoT SiteWise storage** section, the **Warm tier storage** is in one of these states:

- **Enabled** – If your data existed before the hot tier retention period, AWS IoT SiteWise moves the data to the warm tier."
- **Disabled** – The warm tier storage is disabled.

Configure storage settings for warm tier (AWS CLI)

You can configure storage settings to move data to the warm tier by using the AWS CLI and the following commands.

To prevent overriding the existing configuration, retrieve the current storage configuration information by running the following command:

```
aws iotsitewise describe-storage-configuration
```

Example response without existing cold tier configuration

```
{
          "storageType": "SITEWISE_DEFAULT_STORAGE",
          "disassociatedDataStorage": "ENABLED",
          "configurationStatus": {
```

```
                "state": "ACTIVE"
            },
            "lastUpdateDate": "2021-10-14T15:53:35-07:00",
            "warmTier": "DISABLED"
}
```

Example response with existing cold tier configuration

```
{
      "storageType": "MULTI_LAYER_STORAGE",
          "multiLayerStorage": {
            "customerManagedS3Storage": {
            "s3ResourceArn": "arn:aws:s3:::amzn-s3-demo-bucket/prefix/",
            "roleArn": "arn:aws:iam::aws-account-id:role/role-name"
            }
          },
      "disassociatedDataStorage": "ENABLED",
      "retentionPeriod": {
      "numberOfDays": retention-in-days
      },
       "configurationStatus": {
       "state": "ACTIVE"
      },
      "lastUpdateDate": "2023-10-25T15:59:46-07:00",
      "warmTier": "DISABLED"
}
```

Configure storage settings for warm tier with AWS CLI

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.

```
aws iotsitewise put-storage-configuration --cli-input-json file://file-name.json
```

Example AWS IoT SiteWise configuration with hot and warm tier

```
{
             "storageType": "SITEWISE_DEFAULT_STORAGE",
             "disassociatedDataStorage": "ENABLED",
             "warmTier": "ENABLED",
             "retentionPeriod": {
```

```
                "numberOfDays": hot-tier-retention-in-days
            }

}
```

hot-tier-retention-in-days must be a whole number greater than or equal to 30 days.

Example response

```
{
            "storageType": "SITEWISE_DEFAULT_STORAGE",
            "configurationStatus": {
            "state": "UPDATE_IN_PROGRESS"
            }
}
```

If you have cold tier storage enabled, see Configure storage settings with AWS CLI and existing cold tier.

Configure storage settings with AWS CLI and existing cold tier

Configure storage settings using AWS CLI with existing cold tier storage

- 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.

  ```
  aws iotsitewise put-storage-configuration --cli-input-json file://file-name.json
  ```

  #### Example AWS IoT SiteWise storage configuration

  - Replace *amzn-s3-demo-bucket* with your Amazon S3 bucket name.
  - Replace *prefix* with your Amazon S3 prefix.
  - Replace *aws-account-id* with your AWS account ID.
  - Replace *role-name* with the name of the Amazon S3 access role that allows AWS IoT SiteWise to send data to Amazon S3.
  - Replace *hot-tier-retention-in-days* with a whole number greater than or equal to 30 days.
  - Replace *warm-tier-retention-in-days* with a whole number greater than or equal to 365 days.

> **Note**
>
> AWS IoT SiteWise will delete any data in the warm tier that's older than the retention period of the cold tier. If you don't set a retention period, your data is stored indefinitely.

```
{
      "storageType": "MULTI_LAYER_STORAGE",
        "multiLayerStorage": {
          "customerManagedS3Storage": {
              "s3ResourceArn": "arn:aws:s3:::amzn-s3-demo-bucket/prefix/",
              "roleArn": "arn:aws:iam::aws-account-id:role/role-name"
              }
          },
    "disassociatedDataStorage": "ENABLED",
    "retentionPeriod": {
      "numberOfDays": hot-tier-retention-in-days
    },
    "warmTier": "ENABLED",
    "warmTierRetentionPeriod": {
      "numberOfDays": warm-tier-retention-in-days
    }
}
```

Example response

```
{
      "storageType": "MULTI_LAYER_STORAGE",
      "configurationStatus": {
        "state": "UPDATE_IN_PROGRESS"
       }
}
```

Configure storage settings for cold tier (console)

The following procedure shows you how to configure the storage settings to replicate data to the cold tier 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 **Enable cold tier storage**. The cold tier storage is disabled by default.

    b. For **S3 bucket location**, enter the name of an existing Amazon S3 bucket and a prefix.

    > **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:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

Replace amzn-s3-demo-bucket with the name of your Amazon S3 bucket.

- If the Amazon S3 bucket is encrypted using a customer managed KMS key, the KMS key must have an access policy with an IAM role for `kms:Decrypt` and `kms:GenerateDataKey` operations.

d. To setup hot tier, see Step 5 in Configure storage settings for warm tier (console).

e. (Optional) For **AWS IoT Analytics integration**, do the following.

> **Note**
>
> End of support notice: On December 15, 2025, AWS will end support for AWS IoT Analytics. After December 15, 2025, you will no longer be able to access the AWS IoT Analytics console or AWS IoT Analytics resources. For more information, see AWS IoT Analytics end of support.

i. If you want to use AWS IoT Analytics to query your data, choose **Enabled AWS IoT Analytics data store**.

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*.

f. Choose **Save**.

In the **AWS IoT SiteWise storage** section, the **Cold tier storage** can be one of the following values:

- **Enabled** – AWS IoT SiteWise replicates your data to the specified Amazon S3 bucket.
- **Enabling** – AWS IoT SiteWise is processing your request to enable the cold tier storage. This process can take several minutes to complete.
- **Enable_Failed** – AWS IoT SiteWise couldn't process your request to enable the cold tier storage. 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 Monitor with Amazon CloudWatch Logs.
- **Disabled** – The cold tier storage is disabled.

Configure storage settings for cold tier (AWS CLI)

The following procedure shows you how to configure the storage settings to replicate data to the cold tier 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.

```
aws iotsitewise put-storage-configuration --cli-input-json file://file-name.json
```

Example AWS IoT SiteWise storage configuration

- Replace *amzn-s3-demo-bucket* with your Amazon S3 bucket name.
- Replace *prefix* with your Amazon S3 prefix.
- Replace *aws-account-id* with your AWS account ID.
- Replace *role-name* with the name of the Amazon S3 access role that allows AWS IoT SiteWise to send data to Amazon S3.

- Replace *retention-in-days* with a whole number than is greater than or equal to 30 days.

```
{
      "storageType": "MULTI_LAYER_STORAGE",
      "multiLayerStorage": {
          "customerManagedS3Storage": {
              "s3ResourceArn": "arn:aws:s3:::amzn-s3-demo-bucket/prefix/",
              "roleArn": "arn:aws:iam::aws-account-id:role/role-name"
          }
      },
      "retentionPeriod": {
          "numberOfDays": retention-in-days,
          "unlimited": false
      }
  }
```

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:::amzn-s3-demo-bucket",
                    "arn:aws:s3:::amzn-s3-demo-bucket/*"
                ]
            }
        ]
    }
```

Replace amzn-s3-demo-bucket with the name of your Amazon S3 bucket.

- If the Amazon S3 bucket is encrypted using a customer managed KMS key, the KMS key must have an access policy with an IAM role for `kms:Decrypt` and `kms:GenerateDataKey` operations.

Example response

```
{
    "storageType": "MULTI_LAYER_STORAGE",
    "retentionPeriod": {
        "numberOfDays": 100,
        "unlimited": false
    },
    "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

```
{
      "storageType": "MULTI_LAYER_STORAGE",
      "multiLayerStorage": {
```

```
            "customerManagedS3Storage": {
                "s3ResourceArn": "arn:aws:s3:::amzn-s3-demo-bucket/torque/",
                "roleArn": "arn:aws:iam::123456789012:role/SWAccessS3Role"
            }
        },
        "retentionPeriod": {
            "numberOfDays": 100,
            "unlimited": false
        },
        "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, your data is only stored in the hot tier of AWS IoT SiteWise.

Example response

```
{
        "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

```
{
```

```
        "storageType": "SITEWISE_DEFAULT_STORAGE",
        "configurationStatus": {
            "state": "ACTIVE"
        },
        "lastUpdateDate": "2021-03-30T15:57:14-07:00"
    }
```

(Optional) Create an AWS IoT Analytics data store (AWS CLI)

> **Note**
>
> End of support notice: On December 15, 2025, AWS will end support for AWS IoT Analytics. After December 15, 2025, you will no longer be able to access the AWS IoT Analytics console or AWS IoT Analytics resources. For more information, see AWS IoT Analytics end of support.

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 the *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 *amzn-s3-demo-bucket* with your AWS IoT Analytics data store name and Amazon S3 bucket name.

```
{
      "datastoreName": "data-store-name",
      "datastoreStorage": {
          "iotSiteWiseMultiLayerStorage": {
              "customerManagedS3Storage": {
                  "bucket": "amzn-s3-demo-bucket"
              }
          }
      },
      "retentionPeriod": {
          "numberOfDays": 90
      }
  }
```

Example response

```
{
      "datastoreName": "datastore_IoTSiteWise_demo",
      "datastoreArn": "arn:aws:iotanalytics:us-west-2:123456789012:datastore/
datastore_IoTSiteWise_demo",
      "retentionPeriod": {
          "numberOfDays": 90,
          "unlimited": false
      }
  }
```

Troubleshoot storage settings for AWS IoT SiteWise

Use the following information to troubleshoot and resolve issues with the storage configuration.

Issues

- Error: Bucket doesn't exist
- Error: Access denied to Amazon S3 path
- Error: Role ARN can't be assumed

- Error: Failed to access cross-Region Amazon S3 bucket

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 Amazon 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:::amzn-s3-demo-bucket",
                  "arn:aws:s3:::amzn-s3-demo-bucket/*"
              ]
          }
      ]
  }
```

Replace amzn-s3-demo-bucket 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 Amazon 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 of data saved in the cold tier

AWS IoT SiteWise stores your data in the cold tier by replicating time series, including measurements, metrics, transforms and aggregates, and also asset and asset model definitions. The following describes the file paths and schemas of data that is sent to the cold tier.

Topics

Equipment data (measurements)

AWS IoT SiteWise exports equipment data (measurements) to the cold tier once every six hours. Raw data is saved in the cold tier in the Apache AVRO (`.avro`) format.

File path

AWS IoT SiteWise stores equipment data (measurements) in the cold tier using 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.

Path component	Description
`keyPrefix`	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.
`raw`	The folder that stores time series data from equipment (measurements). The `raw` folder is saved in the prefix folder.
`seriesBucket`	A hexadecimal number between 00 and ff. This number is derived from `timeSeriesId` . This partition is used to increase throughpu t when AWS IoT SiteWise writes to the cold tier. When you use Amazon Athena to run queries, you can use the partition for fine-grai n 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.

Path component	Description
	• 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 the cold tier

```
keyPrefix/raw/startYear=2021/startMonth=1/startDay=2/seriesBucket=a2/
raw_7020c8e2-e6db-40fa-9845-ed0dddd4c77d_95e63da7-d34e-43e1-
bc6f-1b490154b07a_1609577700_GOOD.avro
```

Fields

The schema of raw data that is exported to the cold tier contains the following fields.

AWS IoT SiteWise advises customers to implement support for schema evolution on systems that read raw data from cold tier, as there may be additional fields introduced in the future.

Null data is represented as all value fields being null. However, customers will still receive the correct data type when querying with AWS IoT SiteWise APIs.

Field name	Supported types	Default type	Description
`seriesId`	`string`	N/A	The ID that identifies the time series data from equipment (measurements). You can use this field to join raw data and

Field name	Supported types	Default type	Description
			asset metadata in queries.
`timeInSeconds`	`long`	N/A	The timestamp date, in seconds, in the Unix epoch format. Fractional nanosecond data is provided by `offsetInNanos` .
`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).

Field name	Supported types	Default type	Description
`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 the cold tier

```
  {"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,"re
  {"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,"re
  {"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,"re
  {"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,"re
  {"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,"re
```

Metrics, transforms, and aggregates

AWS IoT SiteWise exports metrics, transforms, and aggregates to the cold tier once every six hours. Metrics, transforms, and aggregates are saved in the cold tier in the Apache AVRO (`.avro`) format.

File path

AWS IoT SiteWise stores metrics, transforms, and aggregates in the cold tier using the following template.

```
{keyPrefix}/agg/startYear={startYear}/startMonth={startMonth}/startDay={startDay}/
seriesBucket={seriesBucket}/agg_{timeseriesId}_{startTimestamp}_{quality}.avro
```

Every file path to metrics, transforms, and aggregates in Amazon S3 contains the following components.

Path component	Description
`keyPrefix`	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.
`agg`	The folder that stores time series data from metrics. The `agg` folder is saved in the prefix folder.
`seriesBucket`	A hexadecimal number between 00 and ff. This number is derived from `timeSeriesId` . This partition is used to increase throughpu t when AWS IoT SiteWise writes to the cold tier. When you use Amazon Athena to run queries, you can use the partition for fine-grai n 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.

Path component	Description
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 informati on, 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 metrics in the cold tier

```
keyPrefix/agg/startYear=2021/startMonth=1/startDay=2/seriesBucket=a2/
agg_7020c8e2-e6db-40fa-9845-ed0dddd4c77d_95e63da7-d34e-43e1-
bc6f-1b490154b07a_1609577700_GOOD.avro
```

Fields

The schema of metrics, transforms, and aggregates that are exported to the cold tier contains the following fields.

Field name	Supported types	Default type	Description
seriesId	string	N/A	The ID that identifie s the time series data from equipment, metrics, or transform s. You can use this

Field name	Supported types	Default type	Description
			field to join raw data and asset metadata in queries.
`timeInSeconds`	`long`	N/A	The timestamp date, in seconds, in the Unix epoch format. Fractional nanosecond data is provided by `offsetInNanos` .
`offsetInNanos`	`long`	N/A	The nanosecond offset from `timeInSeconds` .
`quality`	`string`	N/A	The quality by which to filter asset data.
`resolution`	`string`	N/A	The time interval over which to aggregate data.
`count`	`double` or `null`	`null`	The total number of data points for the given variables over the current time interval.
`average`	`double` or `null`	`null`	The mean of the given variables ' values over the current time interval.
`min`	`double` or `null`	`null`	The minimum of the given variables ' values over the current time interval.

Field name	Supported types	Default type	Description
`max`	`boolean` or `null`	`null`	The maximum of the given variables ' values over the current time interval.
`sum`	`string` or `null`	`null`	The sum of the given variables' values over the current time interval.
`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 Metric data in the cold tier

```
{"seriesId":"f74c2828-5317-4df3-
ba16-6d41b5bcb531","timeInSeconds":1637334060,"offsetInNanos":0,"quality":"GOOD","resolution":"
{"double":16.0},"min":{"double":1.0},"max":{"double":31.0},"sum":
{"double":496.0},"recordVersion":null}
  {"seriesId":"f74c2828-5317-4df3-
ba16-6d41b5bcb531","timeInSeconds":1637334120,"offsetInNanos":0,"quality":"GOOD","resolution":"
{"double":46.0},"min":{"double":32.0},"max":{"double":60.0},"sum":
{"double":1334.0},"recordVersion":null}
  {"seriesId":"f74c2828-5317-4df3-
ba16-6d41b5bcb531","timeInSeconds":1637334540,"offsetInNanos":0,"quality":"GOOD","resolution":"
{"double":16.0},"min":{"double":1.0},"max":{"double":31.0},"sum":
{"double":496.0},"recordVersion":null}
  {"seriesId":"f74c2828-5317-4df3-
ba16-6d41b5bcb531","timeInSeconds":1637334600,"offsetInNanos":0,"quality":"GOOD","resolution":"
{"double":46.0},"min":{"double":32.0},"max":{"double":60.0},"sum":
{"double":1334.0},"recordVersion":null}
```

```
  {"seriesId":"f74c2828-5317-4df3-
ba16-6d41b5bcb531","timeInSeconds":1637335020,"offsetInNanos":0,"quality":"GOOD","resolution":"
{"double":16.0},"min":{"double":1.0},"max":{"double":31.0},"sum":
{"double":496.0},"recordVersion":null}
```


Asset metadata

When you enable AWS IoT SiteWise to export data to the cold tier for the first time, asset metadata is exported to the cold tier. After the initial configuration, AWS IoT SiteWise exports asset metadata to the tier only when you change asset model definitions or asset definitions. Asset metadata is saved in the cold tier in the newline delimited JSON (`.ndjson`) format.

File path

AWS IoT SiteWise stores asset metadata in the cold tier using the following template.

```
{keyPrefix}/asset_metadata/asset_{assetId}.ndjson
```

Every file path to asset metadata in the cold tier contains the following components.

Path component	Description
`keyPrefix`	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.
`asset_metadata`	The folder that stores asset metadata. The `asset_metadata` folder is saved in the prefix folder.
`fileName`	The file name uses the underscore (_) character as a delimiter to separate the following: • The `asset` prefix. • The `assetId` value.

Path component	Description
	The file is saved in the `.ndjson` format.

Example file path to asset metadata in the colder tier

`keyPrefix/asset_metadata/asset_35901915-d476-4dca-8637-d9ed4df939ed.ndjson`

Fields

The schema of asset metadata that is exported to the cold tier contains the following fields.

Field name	Description
`assetId`	The ID of the asset.
`assetName`	The name of the asset.
`assetExternalId`	The external ID of the asset.
`assetModelId`	The ID of the asset model used to create this asset.
`assetModelName`	The name of the asset model.
`assetModelExternalId`	The external ID of the asset model.
`assetPropertyId`	The ID of the asset property.
`assetPropertyName`	The name of the asset property.
`assetPropertyExternalId`	The external ID of the asset property.
`assetPropertyDataType`	The data type of the asset property.
`assetPropertyUnit`	The unit of the asset property (for example, `Newtons` and `RPM`).
`assetPropertyAlias`	The alias that identifies the asset property, such as an OPC UA server data stream path

Field name	Description
	(for example, `/company/windfarm/3/turbine/7/temperature`).
`timeSeriesId`	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.
`timeSeriesBucket`	A hexadecimal number between 00 and ff. This number is derived from `timeSeriesId` . This partition is used to increase throughpu t when AWS IoT SiteWise writes to the cold tier. When you use Amazon Athena to run queries, you can use the partition for fine-grai n partitioning to improve query performance. `timeSeriesBucket` and `seriesBucket` in the file path to raw data are the same number.
`assetCompositeModelId`	The ID of the composite model.
`assetCompositeModelExternalId`	The external ID of the composite model.
`assetCompositeModelDescription`	The description of the composite model.
`assetCompositeModelName`	The name of the composite model.
`assetCompositeModelType`	The type of the composite model. For alarm composite models, this type is `AWS/ALARM` .
`assetCreationDate`	The date the asset was created, in Unix epoch time.
`assetLastUpdateDate`	The date the asset was last updated, in Unix epoch time.
`assetStatusErrorCode`	The error code.

Field name	Description
assetStatusErrorMessage	The error message.
assetStatusState	The current status of the asset.

Example asset metadata in the cold tier


```
 {"assetId":"7020c8e2-e6db-40fa-9845-
ed0dddd4c77d","assetExternalId":null,"assetName":"Wind Turbine Asset
 2","assetModelId":"ec1d924f-f07d-444f-b072-
e2994c165d35","assetModelExternalId":null,"assetModelName":"Wind
 Turbine Asset Model","assetPropertyId":"95e63da7-d34e-43e1-
bc6f-1b490154b07a","assetPropertyExternalId":null,"assetPropertyName":"Temperature","assetPrope
Washington/Seattle/WT2/temp","timeSeriesId":"7020c8e2-e6db-40fa-9845-
ed0dddd4c77d_95e63da7-d34e-43e1-
bc6f-1b490154b07a","timeSeriesBucket":"f6","assetArn":null,"assetCompositeModelDescription":nul
 {"assetId":"7020c8e2-e6db-40fa-9845-
ed0dddd4c77d","assetExternalId":null,"assetName":"Wind Turbine Asset
 2","assetModelId":"ec1d924f-f07d-444f-b072-
e2994c165d35","assetModelExternalId":null,"assetModelName":"Wind Turbine Asset
 Model","assetPropertyId":"c706d54d-4c11-42dc-9a01-63662fc697b4","assetPropertyExternalId":null
Washington/Seattle/WT2/pressure","timeSeriesId":"7020c8e2-e6db-40fa-9845-
ed0dddd4c77d_c706d54d-4c11-42dc-9a01-63662fc697b4","timeSeriesBucket":"1e","assetArn":null,"ass
 {"assetId":"7020c8e2-e6db-40fa-9845-
ed0dddd4c77d","assetExternalId":null,"assetName":"Wind Turbine Asset
 2","assetModelId":"ec1d924f-f07d-444f-b072-
e2994c165d35","assetModelExternalId":null,"assetModelName":"Wind
 Turbine Asset Model","assetPropertyId":"8cf1162f-dead-4fbe-b468-
c8e24cde9f50","assetPropertyExternalId":null,"assetPropertyName":"Max
 Temperature","assetPropertyDataType":"DOUBLE","assetPropertyUnit":null,"assetPropertyAlias":nu
e6db-40fa-9845-ed0dddd4c77d_8cf1162f-dead-4fbe-b468-
c8e24cde9f50","timeSeriesBucket":"d7","assetArn":null,"assetCompositeModelDescription":null,"as

 {"assetId":"3a5f2a22-3b37-4332-9c1c-404ea1d73fab","assetExternalId":null,"assetName":"BatchAss
ebc75e75e827","assetModelExternalId":null,"assetModelName":"FlashTestAssetModelDouble","assetPr
b410-
ab401a9176ed","assetPropertyExternalId":null,"assetPropertyName":"measurementProperty","assetPr
ae89-
ff316f5ff8aa","timeSeriesBucket":"af","assetArn":null,"assetCompositeModelDescription":null,"as
```

Asset hierarchy metadata

When you enable AWS IoT SiteWise to save data the in cold tier for the first time, asset hierarchy metadata is exported to the cold tier. After the initial configuration, AWS IoT SiteWise exports asset hierarchy metadata to the cold tier only when you make changes to asset model or asset definitions. Asset hierarchy metadata is saved in the cold tier in the newline delimited JSON (`.ndjson`) format.

An external identifier for the hierarchy, target asset, or source asset is retrieved by calling the DescribeAsset API.

File path

AWS IoT SiteWise stores asset hierarchy metadata in the cold tier using the following template.

```
{keyPrefix}/asset_hierarchy_metadata/{parentAssetId}_{hierarchyId}.ndjson
```

Every file path to asset hierarchy metadata in the cold tier contains the following components.

Path component	Description
`keyPrefix`	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.
`asset_hierarchy_metadata`	The folder that stores asset hierarchy metadata. The `asset_hierarchy_metadata` folder is saved in the prefix folder.
`fileName`	The file name uses the underscore (_) character as a delimiter to separate the following: • The `parentAssetId` value. • The `hierarchyId` value.

Path component	Description
	The file is saved in the `.ndjson` format.

Example file path to asset hierarchy metadata in the cold tier

`keyPrefix/asset_hierarchy_metadata/35901915-d476-4dca-8637-d9ed4df939ed_c5b3ced8-589a-48c7-9998-cdccfc9747a0.ndjson`

Fields

The schema of asset hierarchy metadata that is exported to the cold tier contains the following fields.

Field name	Description
`sourceAssetId`	The ID of the source asset in this asset relationship.
`targetAssetId`	The ID of the target asset in this asset relationship.
`hierarchyId`	The ID of the hierarchy.
`associationType`	The association type of this asset relationship. The value must be `CHILD`. The target asset is a child asset of the source asset.

Example asset hierarchy metadata in the cold tier

```
{"sourceAssetId":"80388e72-2284-44fb-9c89-
bfbaf0dfedd2","targetAssetId":"2b866c25-0c74-4750-bdf5-
b73683c8a2a2","hierarchyId":"bbed9f59-0412-4585-
a61d-6044db526aee","associationType":"CHILD"}
  {"sourceAssetId":"80388e72-2284-44fb-9c89-
bfbaf0dfedd2","targetAssetId":"6b51246e-984d-460d-
bc0b-470ea47d1e31","hierarchyId":"bbed9f59-0412-4585-
a61d-6044db526aee","associationType":"CHILD"}
```

To view your data in the cold tier

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**.

Storage data index files

AWS IoT SiteWise uses these files to optimize data query performance. They appear in your Amazon S3 bucket, but you don't need to use them.

File path

AWS IoT SiteWise stores data index files in the cold tier using the following template.

```
keyPrefix/index/series=timeseriesId/startYear=startYear/startMonth=startMonth/
startDay=startDay/index_timeseriesId_startTimestamp_quality
```

Example file path to data storage index file

```
keyPrefix/index/series=7020c8e2-e6db-40fa-9845-ed0dddd4c77d_95e63da7-
d34e-43e1-bc6f-1b490154b07a/startYear=2022/startMonth=02/startDay=03/
index_7020c8e2-e6db-40fa-9845-ed0dddd4c77d_95e63da7-d34e-43e1-
bc6f-1b490154b07a_1643846400_GOOD
```

Code examples for AWS IoT SiteWise using AWS SDKs

The following code examples show how to use AWS IoT SiteWise with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](). This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello AWS IoT SiteWise

The following code examples show how to get started using AWS IoT SiteWise.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository]().

```
public class HelloSitewise {
    private static final Logger logger =
 LoggerFactory.getLogger(HelloSitewise.class);
    public static void main(String[] args) {
         fetchAssetModels();
    }

    /**
     * Fetches asset models using the provided {@link IoTSiteWiseAsyncClient}.
     */
```

```
    public static void fetchAssetModels() {
        IoTSiteWiseAsyncClient siteWiseAsyncClient =
 IoTSiteWiseAsyncClient.create();
        ListAssetModelsRequest assetModelsRequest =
 ListAssetModelsRequest.builder()
            .assetModelTypes(AssetModelType.ASSET_MODEL)
            .build();

        // Asynchronous paginator - process paginated results.
        ListAssetModelsPublisher listModelsPaginator =
 siteWiseAsyncClient.listAssetModelsPaginator(assetModelsRequest);
        CompletableFuture<Void> future = listModelsPaginator.subscribe(response -
> {
            response.assetModelSummaries().forEach(assetSummary ->
                logger.info("Asset Model Name: {} ", assetSummary.name())
            );
        });

        // Wait for the asynchronous operation to complete
        future.join();
    }
}
```

- For API details, see ListAssetModels in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  paginateListAssetModels,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
```

```
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- For API details, see ListAssetModels in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3
```

```
def hello_iot_sitewise(iot_sitewise_client):
    """
    Use the AWS SDK for Python (Boto3) to create an AWS IoT SiteWise
    client and list the asset models in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param iot_sitewise_client: A Boto3 AWS IoT SiteWise Client object. This
 object wraps
                                the low-level AWS IoT SiteWise service API.
    """
    print("Hello, AWS IoT SiteWise! Let's list some of your asset models:\n")
    paginator = iot_sitewise_client.get_paginator("list_asset_models")
    page_iterator = paginator.paginate(PaginationConfig={"MaxItems": 10})

    asset_model_names: [str] = []
    for page in page_iterator:
        for asset_model in page["assetModelSummaries"]:
            asset_model_names.append(asset_model["name"])

    print(f"{len(asset_model_names)} asset model(s) retrieved.")
    for asset_model_name in asset_model_names:
        print(f"\t{asset_model_name}")


if __name__ == "__main__":
    hello_iot_sitewise(boto3.client("iotsitewise"))
```

- For API details, see ListAssetModels in *AWS SDK for Python (Boto3) API Reference*.

Code examples

    - Use CreateGateway with an AWS SDK or CLI
    - Use CreatePortal with an AWS SDK or CLI
    - Use DeleteAsset with an AWS SDK or CLI
    - Use DeleteAssetModel with an AWS SDK or CLI
    - Use DeleteGateway with an AWS SDK or CLI
    - Use DeletePortal with an AWS SDK or CLI
    - Use DescribeAssetModel with an AWS SDK or CLI
    - Use DescribeGateway with an AWS SDK or CLI
    - Use DescribePortal with an AWS SDK or CLI
    - Use GetAssetPropertyValue with an AWS SDK or CLI
    - Use ListAssetModels with an AWS SDK or CLI

Basic examples for AWS IoT SiteWise using AWS SDKs

The following code examples show how to use the basics of AWS IoT SiteWise with AWS SDKs.

Examples

- Hello AWS IoT SiteWise
- Learn the basics of AWS IoT SiteWise with an AWS SDK
- Actions for AWS IoT SiteWise using AWS SDKs
    - Use BatchPutAssetPropertyValue with an AWS SDK or CLI
    - Use CreateAsset with an AWS SDK or CLI
    - Use CreateAssetModel with an AWS SDK or CLI
    - Use CreateGateway with an AWS SDK or CLI
    - Use CreatePortal with an AWS SDK or CLI
    - Use DeleteAsset with an AWS SDK or CLI
    - Use DeleteAssetModel with an AWS SDK or CLI
    - Use DeleteGateway with an AWS SDK or CLI
    - Use DeletePortal with an AWS SDK or CLI
    - Use DescribeAssetModel with an AWS SDK or CLI
    - Use DescribeGateway with an AWS SDK or CLI

- Use DescribePortal with an AWS SDK or CLI
- Use GetAssetPropertyValue with an AWS SDK or CLI
- Use ListAssetModels with an AWS SDK or CLI

Hello AWS IoT SiteWise

The following code examples show how to get started using AWS IoT SiteWise.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
public class HelloSitewise {
    private static final Logger logger =
 LoggerFactory.getLogger(HelloSitewise.class);
    public static void main(String[] args) {
         fetchAssetModels();
    }

    /**
     * Fetches asset models using the provided {@link IoTSiteWiseAsyncClient}.
     */
    public static void fetchAssetModels() {
        IoTSiteWiseAsyncClient siteWiseAsyncClient =
 IoTSiteWiseAsyncClient.create();
        ListAssetModelsRequest assetModelsRequest =
 ListAssetModelsRequest.builder()
            .assetModelTypes(AssetModelType.ASSET_MODEL)
            .build();

        // Asynchronous paginator - process paginated results.
        ListAssetModelsPublisher listModelsPaginator =
 siteWiseAsyncClient.listAssetModelsPaginator(assetModelsRequest);
```

```
        CompletableFuture<Void> future = listModelsPaginator.subscribe(response -
> {
            response.assetModelSummaries().forEach(assetSummary ->
                logger.info("Asset Model Name: {} ", assetSummary.name())
            );
        });

        // Wait for the asynchronous operation to complete
        future.join();
    }
}
```

- For API details, see ListAssetModels in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  paginateListAssetModels,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
```

```
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- For API details, see ListAssetModels in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3


def hello_iot_sitewise(iot_sitewise_client):
    """
    Use the AWS SDK for Python (Boto3) to create an AWS IoT SiteWise
    client and list the asset models in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param iot_sitewise_client: A Boto3 AWS IoT SiteWise Client object. This
 object wraps
                                the low-level AWS IoT SiteWise service API.
```

```
    """
    print("Hello, AWS IoT SiteWise! Let's list some of your asset models:\n")
    paginator = iot_sitewise_client.get_paginator("list_asset_models")
    page_iterator = paginator.paginate(PaginationConfig={"MaxItems": 10})

    asset_model_names: [str] = []
    for page in page_iterator:
        for asset_model in page["assetModelSummaries"]:
            asset_model_names.append(asset_model["name"])

    print(f"{len(asset_model_names)} asset model(s) retrieved.")
    for asset_model_name in asset_model_names:
        print(f"\t{asset_model_name}")


if __name__ == "__main__":
    hello_iot_sitewise(boto3.client("iotsitewise"))
```

- For API details, see ListAssetModels in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn the basics of AWS IoT SiteWise with an AWS SDK

The following code examples show how to:

- Create an AWS IoT SiteWise Asset Model.
- Create an AWS IoT SiteWise Asset.
- Retrieve the property ID values.
- Send data to an AWS IoT SiteWise Asset.
- Retrieve the value of the AWS IoT SiteWise Asset property.
- Create an AWS IoT SiteWise Portal.
- Create an AWS IoT SiteWise Gateway.
- Describe the AWS IoT SiteWise Gateway.
- Delete the AWS IoT SiteWise Assets.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/iotsitewise#code-examples).

Run an interactive scenario demonstrating AWS IoT SiteWise features.

```
public class SitewiseScenario {

    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    private static final Logger logger =
 LoggerFactory.getLogger(SitewiseScenario.class);
    static Scanner scanner = new Scanner(System.in);

    private static final String ROLES_STACK = "RoleSitewise";

    static SitewiseActions sitewiseActions = new SitewiseActions();

    public static void main(String[] args) throws Throwable {
        Scanner scanner = new Scanner(System.in);
        String contactEmail = "user@mydomain.com"; // Change email address.
        String assetModelName = "MyAssetModel1";
        String assetName = "MyAsset1" ;
        String portalName = "MyPortal1" ;
        String gatewayName = "MyGateway1" ;
        String myThing =  "MyThing1" ;

        logger.info("""
            AWS IoT SiteWise is a fully managed software-as-a-service (SaaS)
 that
            makes it easy to collect, store, organize, and monitor data from
 industrial equipment and processes.
            It is designed to help industrial and manufacturing organizations
 collect data from their equipment and
            processes, and use that data to make informed decisions about their
 operations.
```

```
            One of the key features of AWS IoT SiteWise is its ability to connect
to a wide range of industrial
            equipment and systems, including programmable logic controllers
(PLCs), sensors, and other
            industrial devices. It can collect data from these devices and
organize it into a unified data model,
            making it easier to analyze and gain insights from the data. AWS IoT
SiteWise also provides tools for
            visualizing the data, setting up alarms and alerts, and generating
reports.

            Another key feature of AWS IoT SiteWise is its ability to scale to
handle large volumes of data.
            It can collect and store data from thousands of devices and process
millions of data points per second,
            making it suitable for large-scale industrial operations.
Additionally, AWS IoT SiteWise is designed
            to be secure and compliant, with features like role-based access
controls, data encryption,
            and integration with other AWS services for additional security and
compliance features.

            Let's get started...
            """);

        waitForInputToContinue(scanner);
        logger.info(DASHES);

        try {
            runScenario(assetModelName, assetName, portalName, contactEmail,
gatewayName, myThing);
        } catch (RuntimeException e) {
           logger.info(e.getMessage());
        }
    }

    public static void runScenario(String assetModelName, String assetName,
String portalName, String contactEmail, String gatewayName, String myThing)
throws Throwable {
        logger.info("Use AWS CloudFormation to create an IAM role that is
required for this scenario.");
        CloudFormationHelper.deployCloudFormationStack(ROLES_STACK);
```

```
        Map<String, String> stackOutputs = 
CloudFormationHelper.getStackOutputsAsync(ROLES_STACK).join();
        String iamRole = stackOutputs.get("SitewiseRoleArn");
        logger.info("The ARN of the IAM role is {}",iamRole);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("1. Create an AWS SiteWise Asset Model");
        logger.info("""
             An AWS IoT SiteWise Asset Model is a way to represent the physical 
assets, such as equipment,
             processes, and systems, that exist in an industrial environment. 
This model provides a structured and
             hierarchical representation of these assets, allowing users to 
define the relationships and properties
             of each asset.

             This scenario creates two asset model properties: temperature and 
humidity.
            """);
        waitForInputToContinue(scanner);
        String assetModelId = null;
        try {
            CreateAssetModelResponse response = 
sitewiseActions.createAssetModelAsync(assetModelName).join();
            assetModelId = response.assetModelId();
            logger.info("Asset Model successfully created. Asset Model ID: {}. ", 
assetModelId);
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof ResourceAlreadyExistsException) {
                try {
                    assetModelId = 
sitewiseActions.getAssetModelIdAsync(assetModelName).join();
                    logger.info("The Asset Model {} already exists. The id of the 
existing model is {}. Moving on...", assetModelName, assetModelId);
                } catch (CompletionException cex) {
                    logger.error("Exception thrown acquiring the asset model id: 
{}", cex.getCause().getCause(), cex);
                    return;
                }
            } else {
                logger.info("An unexpected error occurred: " + 
cause.getMessage(), cause);
```

```
                return;
            }
        }
        waitForInputToContinue(scanner);

        logger.info(DASHES);
        logger.info("2. Create an AWS IoT SiteWise Asset");
        logger.info("""
             The IoT SiteWise model that we just created defines the structure
and metadata for your physical assets.
             Now we create an asset from the asset model.

            """);
        logger.info("Let's wait 30 seconds for the asset to be ready.");
        countdown(30);
        waitForInputToContinue(scanner);
        String assetId;
        try {
            CreateAssetResponse response =
sitewiseActions.createAssetAsync(assetName, assetModelId).join();
            assetId = response.assetId();
            logger.info("Asset created with ID: {}", assetId);
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof ResourceNotFoundException) {
                logger.info("The asset model id was not found: {}",
cause.getMessage(), cause);
            } else {
                logger.info("An unexpected error occurred: {}",
cause.getMessage(), cause);
            }
            return;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("3. Retrieve the property ID values");
        logger.info("""
             To send data to an asset, we need to get the property ID values. In
this scenario, we access the
             temperature and humidity property ID values.
            """);
        waitForInputToContinue(scanner);
```

```
        Map<String, String>  propertyIds = null;
        try {
            propertyIds = sitewiseActions.getPropertyIds(assetModelId).join();
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof IoTSiteWiseException) {
                logger.error("IoTSiteWiseException occurred: {}",
 cause.getMessage(), ce);
            } else {
                logger.error("An unexpected error occurred: {}",
 cause.getMessage(), ce);
            }
            return;
        }
        String humPropId =  propertyIds.get("Humidity");
        logger.info("The Humidity property Id is {}", humPropId);
        String tempPropId = propertyIds.get("Temperature");
        logger.info("The Temperature property Id is {}", tempPropId);

        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("4. Send data to an AWS IoT SiteWise Asset");
        logger.info("""
            By sending data to an IoT SiteWise Asset, you can aggregate data
 from
            multiple sources, normalize the data into a standard format, and
 store it in a
            centralized location. This makes it easier to analyze and gain
 insights from the data.

            In this example, we generate sample temperature and humidity data and
 send it to the AWS IoT SiteWise asset.

            """);
        waitForInputToContinue(scanner);
        try {
            sitewiseActions.sendDataToSiteWiseAsync(assetId, tempPropId,
 humPropId).join();
            logger.info("Data sent successfully.");
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof ResourceNotFoundException) {
```

```
                logger.error("The AWS resource was not found: {}",
cause.getMessage(), cause);
            } else {
                logger.error("An unexpected error occurred: {}",
cause.getMessage(), cause);
            }
            return;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("5. Retrieve the value of the IoT SiteWise Asset property");
        logger.info("""
            IoT SiteWise is an AWS service that allows you to collect, process,
and analyze industrial data
            from connected equipment and sensors. One of the key benefits of
reading an IoT SiteWise property
            is the ability to gain valuable insights from your industrial data.

            """);
        waitForInputToContinue(scanner);
        try {
            Double assetVal = sitewiseActions.getAssetPropValueAsync(tempPropId,
assetId).join();
            logger.info("The property name is: {}", "Temperature");
            logger.info("The value of this property is: {}", assetVal);

            waitForInputToContinue(scanner);

            assetVal = sitewiseActions.getAssetPropValueAsync(humPropId,
assetId).join();
            logger.info("The property name is: {}", "Humidity");
            logger.info("The value of this property is: {}", assetVal);
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
                if (cause instanceof ResourceNotFoundException) {
                    logger.info("The AWS resource was not found: {}",
cause.getMessage(), cause);
                } else {
                    logger.info("An unexpected error occurred: {}",
cause.getMessage(), cause);
                }
                return;
```

```
            }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("6. Create an IoT SiteWise Portal");
        logger.info("""
             An IoT SiteWise Portal allows you to aggregate data from multiple
industrial sources,
             such as sensors, equipment, and control systems, into a centralized
platform.
            """);
        waitForInputToContinue(scanner);
        String portalId;
        try {
            portalId = sitewiseActions.createPortalAsync(portalName, iamRole,
contactEmail).join();
            logger.info("Portal created successfully. Portal ID {}", portalId);
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof IoTSiteWiseException siteWiseEx) {
                logger.error("IoT SiteWise error occurred: Error message: {},
Error code {}",
                        siteWiseEx.getMessage(),
siteWiseEx.awsErrorDetails().errorCode(), siteWiseEx);
            } else {
                logger.error("An unexpected error occurred: {}",
cause.getMessage());
            }
            return;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("7. Describe the Portal");
        logger.info("""
             In this step, we get a description of the portal and display the
portal URL.
            """);
        waitForInputToContinue(scanner);
        try {
            String portalUrl =
sitewiseActions.describePortalAsync(portalId).join();
```

```
            logger.info("Portal URL: {}", portalUrl);
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof ResourceNotFoundException notFoundException) {
                logger.error("A ResourceNotFoundException occurred: Error
 message: {}, Error code {}",
                        notFoundException.getMessage(),
 notFoundException.awsErrorDetails().errorCode(), notFoundException);
            } else {
                logger.error("An unexpected error occurred: {}",
 cause.getMessage());
            }
            return;
        }
        waitForInputToContinue(scanner);
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("8. Create an IoT SiteWise Gateway");
        logger.info(
            """
                IoT SiteWise Gateway serves as the bridge between industrial
 equipment, sensors, and the
                cloud-based IoT SiteWise service. It is responsible for securely
 collecting, processing, and
                transmitting data from various industrial assets to the IoT
 SiteWise platform,
                enabling real-time monitoring, analysis, and optimization of
 industrial operations.

                """);
        waitForInputToContinue(scanner);
        String gatewayId = "";
        try {
            gatewayId = sitewiseActions.createGatewayAsync(gatewayName,
 myThing).join();
            logger.info("Gateway creation completed successfully. id is {}",
 gatewayId );
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof IoTSiteWiseException siteWiseEx) {
                logger.error("IoT SiteWise error occurred: Error message: {},
 Error code {}",
```

```
                        siteWiseEx.getMessage(),
siteWiseEx.awsErrorDetails().errorCode(), siteWiseEx);
            } else {
                logger.error("An unexpected error occurred: {}",
cause.getMessage());
            }
            return;
        }
        logger.info(DASHES);
        logger.info(DASHES);

        logger.info("9. Describe the IoT SiteWise Gateway");
         waitForInputToContinue(scanner);
        try {
            sitewiseActions.describeGatewayAsync(gatewayId)
                .thenAccept(response -> {
                    logger.info("Gateway Name: {}", response.gatewayName());
                    logger.info("Gateway ARN: {}", response.gatewayArn());
                    logger.info("Gateway Platform: {}",
response.gatewayPlatform());
                    logger.info("Gateway Creation Date: {}",
response.creationDate());
                }).join();
        } catch (CompletionException ce) {
            Throwable cause = ce.getCause();
            if (cause instanceof ResourceNotFoundException notFoundException) {
                logger.error("A ResourceNotFoundException occurred: Error
message: {}, Error code {}",
                        notFoundException.getMessage(),
notFoundException.awsErrorDetails().errorCode(), notFoundException);
            } else {
                logger.error("An unexpected error occurred: {}",
cause.getMessage(), cause);
            }
            return;
        }
        logger.info(DASHES);

        logger.info(DASHES);
        logger.info("10. Delete the AWS IoT SiteWise Assets");
        logger.info(
            """
            Before you can delete the Asset Model, you must delete the assets.
```

```
            """);
        logger.info("Would you like to delete the IoT SiteWise Assets? (y/n)");
        String delAns = scanner.nextLine().trim();
        if (delAns.equalsIgnoreCase("y")) {
            logger.info("You selected to delete the SiteWise assets.");
            waitForInputToContinue(scanner);
            try {
                sitewiseActions.deletePortalAsync(portalId).join();
                logger.info("Portal {} was deleted successfully.", portalId);

            } catch (CompletionException ce) {
                Throwable cause = ce.getCause();
                if (cause instanceof ResourceNotFoundException notFoundException)
 {
                    logger.error("A ResourceNotFoundException occurred: Error
 message: {}, Error code {}",
                            notFoundException.getMessage(),
 notFoundException.awsErrorDetails().errorCode(), notFoundException);
                } else {
                    logger.error("An unexpected error occurred: {}",
 cause.getMessage());
                }
            }

            try {
                sitewiseActions.deleteGatewayAsync(gatewayId).join();
                logger.info("Gateway {} was deleted successfully.", gatewayId);
            } catch (CompletionException ce) {
                Throwable cause = ce.getCause();
                if (cause instanceof ResourceNotFoundException notFoundException)
 {
                    logger.error("A ResourceNotFoundException occurred: Error
 message: {}, Error code {}",
                            notFoundException.getMessage(),
 notFoundException.awsErrorDetails().errorCode(), notFoundException);
                } else {
                    logger.error("An unexpected error occurred: {}",
 cause.getMessage());
                }
            }

            try {
                sitewiseActions.deleteAssetAsync(assetId).join();
```

```
                logger.info("Request to delete asset {} sent successfully",
assetId);
            } catch (CompletionException ce) {
                Throwable cause = ce.getCause();
                if (cause instanceof ResourceNotFoundException notFoundException)
{
                    logger.error("A ResourceNotFoundException occurred: Error
message: {}, Error code {}",
                            notFoundException.getMessage(),
notFoundException.awsErrorDetails().errorCode(), notFoundException);
                } else {
                    logger.error("An unexpected error occurred: {}",
cause.getMessage());
                }
            }
            logger.info("Let's wait 1 minute for the asset to be deleted.");
            countdown(60);
            waitForInputToContinue(scanner);
            logger.info("Delete the AWS IoT SiteWise Asset Model");
            try {
                sitewiseActions.deleteAssetModelAsync(assetModelId).join();
                logger.info("Asset model deleted successfully.");
            } catch (CompletionException ce) {
                Throwable cause = ce.getCause();
                if (cause instanceof ResourceNotFoundException notFoundException)
{
                    logger.error("A ResourceNotFoundException occurred: Error
message: {}, Error code {}",
                            notFoundException.getMessage(),
notFoundException.awsErrorDetails().errorCode(), notFoundException);
                } else {
                    logger.error("An unexpected error occurred: {}",
cause.getMessage());
                }
            }
            waitForInputToContinue(scanner);

        } else {
            logger.info("The resources will not be deleted.");
        }
        logger.info(DASHES);

        logger.info(DASHES);
        CloudFormationHelper.destroyCloudFormationStack(ROLES_STACK);
```

```
        logger.info("This concludes the AWS IoT SiteWise Scenario");
        logger.info(DASHES);
    }

    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            logger.info("");
            logger.info("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input.trim().equalsIgnoreCase("c")) {
                logger.info("Continuing with the program...");
                logger.info("");
                break;
            } else {
                logger.info("Invalid input. Please try again.");
            }
        }
    }

    public static void countdown(int totalSeconds) throws InterruptedException {
        for (int i = totalSeconds; i >= 0; i--) {
            int displayMinutes = i / 60;
            int displaySeconds = i % 60;
            System.out.printf("\r%02d:%02d", displayMinutes, displaySeconds);
            Thread.sleep(1000); // Wait for 1 second
        }
        System.out.println(); // Move to the next line after countdown
        logger.info("Countdown complete!");
    }
}
```

A wrapper class for AWS IoT SiteWise SDK methods.

```
public class SitewiseActions {

    private static final Logger logger =
 LoggerFactory.getLogger(SitewiseActions.class);

    private static IoTSiteWiseAsyncClient ioTSiteWiseAsyncClient;

    private static IoTSiteWiseAsyncClient getAsyncClient() {
```

```
        if (ioTSiteWiseAsyncClient == null) {
            SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
                .maxConcurrency(100)
                .connectionTimeout(Duration.ofSeconds(60))
                .readTimeout(Duration.ofSeconds(60))
                .writeTimeout(Duration.ofSeconds(60))
                .build();

            ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
                .apiCallTimeout(Duration.ofMinutes(2))
                .apiCallAttemptTimeout(Duration.ofSeconds(90))
                .retryStrategy(RetryMode.STANDARD)
                .build();

            ioTSiteWiseAsyncClient = IoTSiteWiseAsyncClient.builder()
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return ioTSiteWiseAsyncClient;
    }


    /**
     * Creates an asset model.
     *
     * @param name the name of the asset model to create.
     * @return a {@link CompletableFuture} that represents a {@link
CreateAssetModelResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps it
     *         available to the calling code as a {@link CompletionException}. By
calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<CreateAssetModelResponse>
createAssetModelAsync(String name) {
        PropertyType humidity = PropertyType.builder()
```

```
            .measurement(Measurement.builder().build())
            .build();

        PropertyType temperaturePropertyType = PropertyType.builder()
            .measurement(Measurement.builder().build())
            .build();

        AssetModelPropertyDefinition temperatureProperty =
AssetModelPropertyDefinition.builder()
            .name("Temperature")
            .dataType(PropertyDataType.DOUBLE)
            .type(temperaturePropertyType)
            .build();

        AssetModelPropertyDefinition humidityProperty =
AssetModelPropertyDefinition.builder()
            .name("Humidity")
            .dataType(PropertyDataType.DOUBLE)
            .type(humidity)
            .build();

        CreateAssetModelRequest createAssetModelRequest =
CreateAssetModelRequest.builder()
            .assetModelName(name)
            .assetModelDescription("This is my asset model")
            .assetModelProperties(temperatureProperty, humidityProperty)
            .build();

        return getAsyncClient().createAssetModel(createAssetModelRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create asset model: {} ",
exception.getCause().getMessage());
                }
            });
    }


    /**
     * Creates an asset with the specified name and asset model Id.
     *
     * @param assetName    the name of the asset to create.
     * @param assetModelId the Id of the asset model to associate with the asset.
```

```
     * @return a {@link CompletableFuture} that represents a {@link
 CreateAssetResponse} result. The calling code can
     *         attach callbacks, then handle the result or exception by calling
 {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps it
     *         available to the calling code as a {@link CompletionException}. By
 calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<CreateAssetResponse> createAssetAsync(String
 assetName, String assetModelId) {
        CreateAssetRequest createAssetRequest = CreateAssetRequest.builder()
            .assetModelId(assetModelId)
            .assetDescription("Created using the AWS SDK for Java")
            .assetName(assetName)
            .build();

        return getAsyncClient().createAsset(createAssetRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create asset: {}",
 exception.getCause().getMessage());
                }
            });
    }

    /**
     * Sends data to the SiteWise service.
     *
     * @param assetId        the ID of the asset to which the data will be sent.
     * @param tempPropertyId the ID of the temperature property.
     * @param humidityPropId the ID of the humidity property.
     * @return a {@link CompletableFuture} that represents a {@link
 BatchPutAssetPropertyValueResponse} result. The
     *         calling code can attach callbacks, then handle the result or
 exception by calling
     *         {@link CompletableFuture#join()} or {@link
 CompletableFuture#get()}.
     *         <p>
```

```
    *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps it
    *         available to the calling code as a {@link CompletionException}. By
calling
    *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
    */
   public CompletableFuture<BatchPutAssetPropertyValueResponse>
sendDataToSiteWiseAsync(String assetId, String tempPropertyId, String
humidityPropId) {
       Map<String, Double> sampleData = generateSampleData();
       long timestamp = Instant.now().toEpochMilli();

       TimeInNanos time = TimeInNanos.builder()
           .timeInSeconds(timestamp / 1000)
           .offsetInNanos((int) ((timestamp % 1000) * 1000000))
           .build();

       BatchPutAssetPropertyValueRequest request =
BatchPutAssetPropertyValueRequest.builder()
           .entries(Arrays.asList(
               PutAssetPropertyValueEntry.builder()
                   .entryId("entry-3")
                   .assetId(assetId)
                   .propertyId(tempPropertyId)
                   .propertyValues(Arrays.asList(
                       AssetPropertyValue.builder()
                           .value(Variant.builder()
                               .doubleValue(sampleData.get("Temperature"))
                               .build())
                           .timestamp(time)
                           .build()
                   ))
                   .build(),
               PutAssetPropertyValueEntry.builder()
                   .entryId("entry-4")
                   .assetId(assetId)
                   .propertyId(humidityPropId)
                   .propertyValues(Arrays.asList(
                       AssetPropertyValue.builder()
                           .value(Variant.builder()
                               .doubleValue(sampleData.get("Humidity"))
                               .build())
                           .timestamp(time)
```

```
                            .build()
                    ))
                    .build()
            ))
            .build();

        return getAsyncClient().batchPutAssetPropertyValue(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("An exception occurred: {}",
exception.getCause().getMessage());
                }
            });
    }

    /**
     * Fetches the value of an asset property.
     *
     * @param propId  the ID of the asset property to fetch.
     * @param assetId the ID of the asset to fetch the property value for.
     * @return a {@link CompletableFuture} that represents a {@link Double}
result. The calling code can attach
     *         callbacks, then handle the result or exception by calling {@link
CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<Double> getAssetPropValueAsync(String propId, String
assetId) {
        GetAssetPropertyValueRequest assetPropertyValueRequest =
GetAssetPropertyValueRequest.builder()
                .propertyId(propId)
                .assetId(assetId)
                .build();

        return getAsyncClient().getAssetPropertyValue(assetPropertyValueRequest)
                .handle((response, exception) -> {
                    if (exception != null) {
```

```
                        logger.error("Error occurred while fetching property
value: {}.", exception.getCause().getMessage());
                        throw (CompletionException) exception;
                    }
                    return response.propertyValue().value().doubleValue();
                });
   }

   /**
    * Retrieves the property IDs associated with a specific asset model.
    *
    * @param assetModelId the ID of the asset model that defines the properties.
    * @return a {@link CompletableFuture} that represents a {@link Map} result
that associates the property name to the
    *         propert ID. The calling code can attach callbacks, then handle the
result or exception by calling
    *         {@link CompletableFuture#join()} or {@link
CompletableFuture#get()}.
    *         <p>
    *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
    *         it available to the calling code as a {@link CompletionException}.
By calling
    *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
    */
   public CompletableFuture<Map<String, String>> getPropertyIds(String
assetModelId) {
       ListAssetModelPropertiesRequest modelPropertiesRequest =
ListAssetModelPropertiesRequest.builder().assetModelId(assetModelId).build();
       return getAsyncClient().listAssetModelProperties(modelPropertiesRequest)
           .handle((response, throwable) -> {
               if (response != null) {
                   return response.assetModelPropertySummaries().stream()
                       .collect(Collectors
                           .toMap(AssetModelPropertySummary::name,
AssetModelPropertySummary::id));
               } else {
                   logger.error("Error occurred while fetching property IDs:
{}.", throwable.getCause().getMessage());
                   throw (CompletionException) throwable;
               }
           });
   }
```

```
    /**
     * Deletes an asset.
     *
     * @param assetId the ID of the asset to be deleted.
     * @return a {@link CompletableFuture} that represents a {@link
 DeleteAssetResponse} result. The calling code can
     *         attach callbacks, then handle the result or exception by calling
 {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DeleteAssetResponse> deleteAssetAsync(String
 assetId) {
        DeleteAssetRequest deleteAssetRequest = DeleteAssetRequest.builder()
            .assetId(assetId)
            .build();

        return getAsyncClient().deleteAsset(deleteAssetRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("An error occurred deleting asset with id: {}",
 assetId);
                }
            });
    }

    /**
     * Deletes an Asset Model with the specified ID.
     *
     * @param assetModelId the ID of the Asset Model to delete.
     * @return a {@link CompletableFuture} that represents a {@link
 DeleteAssetModelResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
```

```
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<DeleteAssetModelResponse>
deleteAssetModelAsync(String assetModelId) {
        DeleteAssetModelRequest deleteAssetModelRequest =
DeleteAssetModelRequest.builder()
            .assetModelId(assetModelId)
            .build();

        return getAsyncClient().deleteAssetModel(deleteAssetModelRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to delete asset model with ID:{}.",
exception.getMessage());
                }
            });
    }

    /**
     * Creates a new IoT SiteWise portal.
     *
     * @param portalName   the name of the portal to create.
     * @param iamRole      the IAM role ARN to use for the portal.
     * @param contactEmail the email address of the portal contact.
     * @return a {@link CompletableFuture} that represents a {@link String}
result of the portal ID. The calling code
     *         can attach callbacks, then handle the result or exception by
calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<String> createPortalAsync(String portalName, String
iamRole, String contactEmail) {
```

```
        CreatePortalRequest createPortalRequest = CreatePortalRequest.builder()
            .portalName(portalName)
            .portalDescription("This is my custom IoT SiteWise portal.")
            .portalContactEmail(contactEmail)
            .roleArn(iamRole)
            .build();

        return getAsyncClient().createPortal(createPortalRequest)
            .handle((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create portal: {} ",
exception.getCause().getMessage());
                    throw (CompletionException) exception;
                }
                return response.portalId();
            });
    }

    /**
     * Deletes a portal.
     *
     * @param portalId the ID of the portal to be deleted.
     * @return a {@link CompletableFuture} that represents a {@link
DeletePortalResponse}. The calling code can attach
     *         callbacks, then handle the result or exception by calling {@link
CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<DeletePortalResponse> deletePortalAsync(String
portalId) {
        DeletePortalRequest deletePortalRequest = DeletePortalRequest.builder()
            .portalId(portalId)
            .build();

        return getAsyncClient().deletePortal(deletePortalRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
```

```
                    logger.error("Failed to delete portal with ID: {}. Error:
{}", portalId, exception.getCause().getMessage());
                }
            });
    }

    /**
     * Retrieves the asset model ID for the given asset model name.
     *
     * @param assetModelName the name of the asset model for the ID.
     * @return a {@link CompletableFuture} that represents a {@link String}
result of the asset model ID or null if the
     *         asset model cannot be found. The calling code can attach
callbacks, then handle the result or exception
     *         by calling {@link CompletableFuture#join()} or {@link
CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<String> getAssetModelIdAsync(String assetModelName)
{
        ListAssetModelsRequest listAssetModelsRequest =
ListAssetModelsRequest.builder().build();
        return getAsyncClient().listAssetModels(listAssetModelsRequest)
                .handle((listAssetModelsResponse, exception) -> {
                    if (exception != null) {
                        logger.error("Failed to retrieve Asset Model ID: {}",
exception.getCause().getMessage());
                        throw (CompletionException) exception;
                    }
                    for (AssetModelSummary assetModelSummary :
listAssetModelsResponse.assetModelSummaries()) {
                        if (assetModelSummary.name().equals(assetModelName)) {
                            return assetModelSummary.id();
                        }
                    }
                    return null;
                });
    }
```

```
    /**
     * Retrieves a portal's description.
     *
     * @param portalId the ID of the portal to describe.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the portal's start URL
     *         (see: {@link DescribePortalResponse#portalStartUrl()}). The
 calling code can attach callbacks, then handle the
     *         result or exception by calling {@link CompletableFuture#join()} or
 {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<String> describePortalAsync(String portalId) {
        DescribePortalRequest request = DescribePortalRequest.builder()
            .portalId(portalId)
            .build();

        return getAsyncClient().describePortal(request)
            .handle((response, exception) -> {
                if (exception != null) {
                    logger.error("An exception occurred retrieving the portal
 description: {}", exception.getCause().getMessage());
                    throw (CompletionException) exception;
                }
                return response.portalStartUrl();
            });
    }


    /**
     * Creates a new IoT Sitewise gateway.
     *
     * @param gatewayName The name of the gateway to create.
     * @param myThing     The name of the core device thing to associate with the
 gateway.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the gateways ID. The calling code
```

```
     *         can attach callbacks, then handle the result or exception by
calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<String> createGatewayAsync(String gatewayName,
String myThing) {
        GreengrassV2 gg = GreengrassV2.builder()
            .coreDeviceThingName(myThing)
            .build();

        GatewayPlatform platform = GatewayPlatform.builder()
            .greengrassV2(gg)
            .build();

        Map<String, String> tag = new HashMap<>();
        tag.put("Environment", "Production");

        CreateGatewayRequest createGatewayRequest =
CreateGatewayRequest.builder()
            .gatewayName(gatewayName)
            .gatewayPlatform(platform)
            .tags(tag)
            .build();

        return getAsyncClient().createGateway(createGatewayRequest)
            .handle((response, exception) -> {
                if (exception != null) {
                    logger.error("Error creating the gateway.");
                    throw (CompletionException) exception;
                }
                logger.info("The ARN of the gateway is {}" ,
response.gatewayArn());
                return response.gatewayId();
            });
    }

    /**
```

```
     * Deletes the specified gateway.
     *
     * @param gatewayId the ID of the gateway to delete.
     * @return a {@link CompletableFuture} that represents a {@link
 DeleteGatewayResponse} result.. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DeleteGatewayResponse> deleteGatewayAsync(String
 gatewayId) {
        DeleteGatewayRequest deleteGatewayRequest =
 DeleteGatewayRequest.builder()
            .gatewayId(gatewayId)
            .build();

        return getAsyncClient().deleteGateway(deleteGatewayRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to delete gateway: {}",
 exception.getCause().getMessage());
                }
            });
    }

    /**
     * Describes the specified gateway.
     *
     * @param gatewayId the ID of the gateway to describe.
     * @return a {@link CompletableFuture} that represents a {@link
 DescribeGatewayResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
```

```
     *          it available to the calling code as a {@link CompletionException}.
 By calling
     *          {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DescribeGatewayResponse> describeGatewayAsync(String
 gatewayId) {
        DescribeGatewayRequest request = DescribeGatewayRequest.builder()
            .gatewayId(gatewayId)
            .build();

        return getAsyncClient().describeGateway(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("An error occurred during the describeGateway
 method: {}", exception.getCause().getMessage());
                }
            });
    }

    private static Map<String, Double> generateSampleData() {
        Map<String, Double> data = new HashMap<>();
        data.put("Temperature", 23.5);
        data.put("Humidity", 65.0);
        return data;
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
import {
  Scenario,
```

```
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
  //} from "@aws-doc-sdk-examples/lib/scenario/index.js";
} from "../../libs/scenario/index.js";
import {
  IoTSiteWiseClient,
  CreateAssetModelCommand,
  CreateAssetCommand,
  ListAssetModelPropertiesCommand,
  BatchPutAssetPropertyValueCommand,
  GetAssetPropertyValueCommand,
  CreatePortalCommand,
  DescribePortalCommand,
  CreateGatewayCommand,
  DescribeGatewayCommand,
  DeletePortalCommand,
  DeleteGatewayCommand,
  DeleteAssetCommand,
  DeleteAssetModelCommand,
  DescribeAssetModelCommand,
} from "@aws-sdk/client-iotsitewise";
import {
  CloudFormationClient,
  CreateStackCommand,
  DeleteStackCommand,
  DescribeStacksCommand,
  waitUntilStackExists,
  waitUntilStackCreateComplete,
  waitUntilStackDeleteComplete,
} from "@aws-sdk/client-cloudformation";
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { parseArgs } from "node:util";
import { readFileSync } from "node:fs";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
const stackName = "SiteWiseBasicsStack";

/**
 * @typedef {{
 *   iotSiteWiseClient: import('@aws-sdk/client-iotsitewise').IotSiteWiseClient,
```

```
 *   cloudFormationClient: import('@aws-sdk/client-
cloudformation').CloudFormationClient,
 *   stackName,
 *   stack,
 *   askToDeleteResources: true,
 *   asset: {assetName: "MyAsset1"},
 *   assetModel: {assetModelName: "MyAssetModel1"},
 *   portal: {portalName: "MyPortal1"},
 *   gateway: {gatewayName: "MyGateway1"},
 *   propertyIds: [],
 *   contactEmail: "user@mydomain.com",
 *   thing: "MyThing1",
 *   sampleData: { temperature: 23.5, humidity: 65.0}
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `AWS IoT SiteWise is a fully managed industrial software-as-a-service (SaaS)
 that makes it easy to collect, store, organize, and monitor data from industrial
 equipment and processes. It is designed to help industrial and manufacturing
 organizations collect data from their equipment and processes, and use that data
 to make informed decisions about their operations.
One of the key features of AWS IoT SiteWise is its ability to connect to a
 wide range of industrial equipment and systems, including programmable logic
 controllers (PLCs), sensors, and other industrial devices. It can collect data
 from these devices and organize it into a unified data model, making it easier
 to analyze and gain insights from the data. AWS IoT SiteWise also provides tools
 for visualizing the data, setting up alarms and alerts, and generating reports.
Another key feature of AWS IoT SiteWise is its ability to scale to handle large
 volumes of data. It can collect and store data from thousands of devices and
 process millions of data points per second, making it suitable for large-scale
 industrial operations. Additionally, AWS IoT SiteWise is designed to be secure
 and compliant, with features like role-based access controls, data encryption,
 and integration with other AWS services for additional security and compliance
 features.
```

```
Let's get started...`,
  { header: true },
);

const displayBuildCloudFormationStack = new ScenarioOutput(
  "displayBuildCloudFormationStack",
  "This scenario uses AWS CloudFormation to create an IAM role that is required
 for this scenario. The stack will now be deployed.",
);

const sdkBuildCloudFormationStack = new ScenarioAction(
  "sdkBuildCloudFormationStack",
  async (/** @type {State} */ state) => {
    try {
      const data = readFileSync(
        `${__dirname}/../../../../resources/cfn/iotsitewise_basics/SitewiseRoles-
template.yml`,
        "utf8",
      );
      await state.cloudFormationClient.send(
        new CreateStackCommand({
          StackName: stackName,
          TemplateBody: data,
          Capabilities: ["CAPABILITY_IAM"],
        }),
      );
      await waitUntilStackExists(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      await waitUntilStackCreateComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      const stack = await state.cloudFormationClient.send(
        new DescribeStacksCommand({
          StackName: stackName,
        }),
      );
      state.stack = stack.Stacks[0].Outputs[0];
      console.log(`The ARN of the IAM role is ${state.stack.OutputValue}`);
    } catch (caught) {
      console.error(caught.message);
```

```
      throw caught;
    }
  },
);

const displayCreateAWSSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSSiteWiseAssetModel",
  `1. Create an AWS SiteWise Asset Model
An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such
 as equipment, processes, and systems, that exist in an industrial environment.
 This model provides a structured and hierarchical representation of these
 assets, allowing users to define the relationships and properties of each asset.

This scenario creates two asset model properties: temperature and humidity.`,
);

const sdkCreateAWSSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    let assetModelResponse;
    try {
      assetModelResponse = await state.iotSiteWiseClient.send(
        new CreateAssetModelCommand({
          assetModelName: state.assetModel.assetModelName,
          assetModelProperties: [
            {
              name: "Temperature",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
            {
              name: "Humidity",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
          ],
        }),
      );
      state.assetModel.assetModelId = assetModelResponse.assetModelId;
      console.log(
```

```
        `Asset Model successfully created. Asset Model ID:
 ${state.assetModel.assetModelId}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceAlreadyExistsException") {
        console.log(
          `The Asset Model ${state.assetModel.assetModelName} already exists.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateAWSIoTSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSIoTSiteWiseAssetModel",
  `2. Create an AWS IoT SiteWise Asset
The IoT SiteWise model that we just created defines the structure and metadata
 for your physical assets. Now we create an asset from the asset model.

Let's wait 30 seconds for the asset to be ready.`,
);

const waitThirtySeconds = new ScenarioAction("waitThirtySeconds", async () => {
  await wait(30); // wait 30 seconds
  console.log("Time's up! Let's check the asset's status.");
});

const sdkCreateAWSIoTSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSIoTSiteWiseAssetModel",
  async (/** @type {State} */ state) => {
    try {
      const assetResponse = await state.iotSiteWiseClient.send(
        new CreateAssetCommand({
          assetModelId: state.assetModel.assetModelId,
          assetName: state.asset.assetName,
        }),
      );
      state.asset.assetId = assetResponse.assetId;
      console.log(`Asset created with ID: ${state.asset.assetId}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
```

```
        console.log(
          `The Asset ${state.assetModel.assetModelName} was not found.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrievePropertyId = new ScenarioOutput(
  "displayRetrievePropertyId",
  `3. Retrieve the property ID values

To send data to an asset, we need to get the property ID values. In this
 scenario, we access the temperature and humidity property ID values.`,
);

const sdkRetrievePropertyId = new ScenarioAction(
  "sdkRetrievePropertyId",
  async (state) => {
    try {
      const retrieveResponse = await state.iotSiteWiseClient.send(
        new ListAssetModelPropertiesCommand({
          assetModelId: state.assetModel.assetModelId,
        }),
      );
      for (const retrieveResponseKey in
 retrieveResponse.assetModelPropertySummaries) {
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Humidity"
        ) {
          state.propertyIds.Humidity =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Temperature"
        ) {
          state.propertyIds.Temperature =
```

```
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
      }
      console.log(`The Humidity propertyId is ${state.propertyIds.Humidity}`);
      console.log(
        `The Temperature propertyId is ${state.propertyIds.Temperature}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem retrieving the properties: ${caught.message}`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displaySendDataToIoTSiteWiseAsset = new ScenarioOutput(
  "displaySendDataToIoTSiteWiseAsset",
  `4. Send data to an AWS IoT SiteWise Asset

By sending data to an IoT SiteWise Asset, you can aggregate data from multiple
 sources, normalize the data into a standard format, and store it in a
 centralized location. This makes it easier to analyze and gain insights from the
 data.

In this example, we generate sample temperature and humidity data and send it to
 the AWS IoT SiteWise asset.`,
);

const sdkSendDataToIoTSiteWiseAsset = new ScenarioAction(
  "sdkSendDataToIoTSiteWiseAsset",
  async (state) => {
    try {
      const sendResponse = await state.iotSiteWiseClient.send(
        new BatchPutAssetPropertyValueCommand({
          entries: [
            {
              entryId: "entry-3",
```

```
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Humidity,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.humidity,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
            {
              entryId: "entry-4",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Temperature,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.temperature,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
          ],
        }),
      );
      console.log("The data was sent successfully.");
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayRetrieveValueOfIoTSiteWiseAsset = new ScenarioOutput(
```

```
  "displayRetrieveValueOfIoTSiteWiseAsset",
  `5. Retrieve the value of the IoT SiteWise Asset property

IoT SiteWise is an AWS service that allows you to collect, process, and analyze
 industrial data from connected equipment and sensors. One of the key benefits of
 reading an IoT SiteWise property is the ability to gain valuable insights from
 your industrial data.`,
);

const sdkRetrieveValueOfIoTSiteWiseAsset = new ScenarioAction(
  "sdkRetrieveValueOfIoTSiteWiseAsset",
  async (/** @type {State} */ state) => {
    try {
      const temperatureResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Temperature,
        }),
      );
      const humidityResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Humidity,
        }),
      );
      console.log(
        `The property value for Temperature is
 ${temperatureResponse.propertyValue.value.doubleValue}`,
      );
      console.log(
        `The property value for Humidity is
 ${humidityResponse.propertyValue.value.doubleValue}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);
```

```
const displayCreateIoTSiteWisePortal = new ScenarioOutput(
  "displayCreateIoTSiteWisePortal",
  `6. Create an IoT SiteWise Portal

An IoT SiteWise Portal allows you to aggregate data from multiple industrial
 sources, such as sensors, equipment, and control systems, into a centralized
 platform.`,
);

const sdkCreateIoTSiteWisePortal = new ScenarioAction(
  "sdkCreateIoTSiteWisePortal",
  async (/** @type {State} */ state) => {
    try {
      const createPortalResponse = await state.iotSiteWiseClient.send(
        new CreatePortalCommand({
          portalName: state.portal.portalName,
          portalContactEmail: state.contactEmail,
          roleArn: state.stack.OutputValue,
        }),
      );
      state.portal = { ...state.portal, ...createPortalResponse };
      await wait(5); // Allow the portal to properly propagate.
      console.log(
        `Portal created successfully. Portal ID
 ${createPortalResponse.portalId}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the Portal: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayDescribePortal = new ScenarioOutput(
  "displayDescribePortal",
  `7. Describe the Portal

In this step, we get a description of the portal and display the portal URL.`,
```

```
);

const sdkDescribePortal = new ScenarioAction(
  "sdkDescribePortal",
  async (/** @type {State} */ state) => {
    try {
      const describePortalResponse = await state.iotSiteWiseClient.send(
        new DescribePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(`Portal URL: ${describePortalResponse.portalStartUrl}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateIoTSiteWiseGateway = new ScenarioOutput(
  "displayCreateIoTSiteWiseGateway",
  `8. Create an IoT SiteWise Gateway

IoT SiteWise Gateway serves as the bridge between industrial equipment, sensors,
 and the cloud-based IoT SiteWise service. It is responsible for securely
 collecting, processing, and transmitting data from various industrial assets
 to the IoT SiteWise platform, enabling real-time monitoring, analysis, and
 optimization of industrial operations.`,
);

const sdkCreateIoTSiteWiseGateway = new ScenarioAction(
  "sdkCreateIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const createGatewayResponse = await state.iotSiteWiseClient.send(
        new CreateGatewayCommand({
          gatewayName: state.gateway.gatewayName,
          gatewayPlatform: {
            greengrassV2: {
              coreDeviceThingName: state.thing,
```

```
          },
        },
      }),
    );
    console.log(
      `Gateway creation completed successfully. ID is
 ${createGatewayResponse.gatewayId}`,
    );
    state.gateway.gatewayId = createGatewayResponse.gatewayId;
  } catch (caught) {
    if (caught.name === "IoTSiteWiseException") {
      console.log(
        `There was a problem creating the gateway: ${caught.message}.`,
      );
      throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
  }
  },
);

const displayDescribeIoTSiteWiseGateway = new ScenarioOutput(
  "displayDescribeIoTSiteWiseGateway",
  "9. Describe the IoT SiteWise Gateway",
);

const sdkDescribeIoTSiteWiseGateway = new ScenarioAction(
  "sdkDescribeIoTSiteWiseGateway",
  async (/** @type {State} */ state) => {
    try {
      const describeGatewayResponse = await state.iotSiteWiseClient.send(
        new DescribeGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log("Gateway creation completed successfully.");
      console.log(`Gateway Name: ${describeGatewayResponse.gatewayName}`);
      console.log(`Gateway ARN: ${describeGatewayResponse.gatewayArn}`);
      console.log(
        `Gateway Platform:
 ${Object.keys(describeGatewayResponse.gatewayPlatform)}`,
      );
      console.log(
```

```
        `Gateway Creation Date: ${describeGatewayResponse.creationDate}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  `10. Delete the AWS IoT SiteWise Assets

Before you can delete the Asset Model, you must delete the assets.`,
  { type: "confirm" },
);

const displayConfirmDeleteResources = new ScenarioAction(
  "displayConfirmDeleteResources",
  async (/** @type {State} */ state) => {
    if (state.askToDeleteResources) {
      return "You selected to delete the SiteWise assets.";
    }
    return "The resources will not be deleted. Please delete them manually to
 avoid charges.";
  },
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    await wait(10); // Give the portal status time to catch up.
    try {
      await state.iotSiteWiseClient.send(
        new DeletePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(
        `Portal ${state.portal.portalName} was deleted successfully.`,
```

```
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
      } else {
        console.log(`When trying to delete the portal: ${caught.message}`);
      }
    }

    try {
      await state.iotSiteWiseClient.send(
        new DeleteGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log(
        `Gateway ${state.gateway.gatewayName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
      } else {
        console.log(`When trying to delete the gateway: ${caught.message}`);
      }
    }

    try {
      await state.iotSiteWiseClient.send(
        new DeleteAssetCommand({
          assetId: state.asset.assetId,
        }),
      );
      await wait(5); // Allow the delete to finish.
      console.log(`Asset ${state.asset.assetName} was deleted successfully.`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
      } else {
        console.log(`When deleting the asset: ${caught.message}`);
      }
    }

    await wait(30); // Allow asset deletion to finish.
    try {
```

```
      await state.iotSiteWiseClient.send(
        new DeleteAssetModelCommand({
          assetModelId: state.assetModel.assetModelId,
        }),
      );
      console.log(
        `Asset Model ${state.assetModel.assetModelName} was deleted
 successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(
          `The Asset Model ${state.assetModel.assetModelName} was not found.`,
        );
      } else {
        console.log(`When deleting the asset model: ${caught.message}`);
      }
    }

    try {
      await state.cloudFormationClient.send(
        new DeleteStackCommand({
          StackName: stackName,
        }),
      );
      await waitUntilStackDeleteComplete(
        { client: state.cloudFormationClient },
        { StackName: stackName },
      );
      console.log("The stack was deleted successfully.");
    } catch (caught) {
      console.log(
        `${caught.message}. The stack was NOT deleted. Please clean up the
 resources manually.`,
      );
    }
  },
  { skipWhen: (/** @type {{}} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the IoT Sitewise Basics scenario for the AWS Javascript SDK v3.
 Thank you!",
```

```
);

const myScenario = new Scenario(
  "IoTSiteWise Basics",
  [
    greet,
    pressEnter,
    displayBuildCloudFormationStack,
    sdkBuildCloudFormationStack,
    pressEnter,
    displayCreateAWSSiteWiseAssetModel,
    sdkCreateAWSSiteWiseAssetModel,
    displayCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    waitThirtySeconds,
    sdkCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    displayRetrievePropertyId,
    sdkRetrievePropertyId,
    pressEnter,
    displaySendDataToIoTSiteWiseAsset,
    sdkSendDataToIoTSiteWiseAsset,
    pressEnter,
    displayRetrieveValueOfIoTSiteWiseAsset,
    sdkRetrieveValueOfIoTSiteWiseAsset,
    pressEnter,
    displayCreateIoTSiteWisePortal,
    sdkCreateIoTSiteWisePortal,
    pressEnter,
    displayDescribePortal,
    sdkDescribePortal,
    pressEnter,
    displayCreateIoTSiteWiseGateway,
    sdkCreateIoTSiteWiseGateway,
    pressEnter,
    displayDescribeIoTSiteWiseGateway,
    sdkDescribeIoTSiteWiseGateway,
    pressEnter,
    askToDeleteResources,
    displayConfirmDeleteResources,
    sdkDeleteResources,
    goodbye,
  ],
  {
```

```
    iotSiteWiseClient: new IoTSiteWiseClient({}),
    cloudFormationClient: new CloudFormationClient({}),
    asset: { assetName: "MyAsset1" },
    assetModel: { assetModelName: "MyAssetModel1" },
    portal: { portalName: "MyPortal1" },
    gateway: { gatewayName: "MyGateway1" },
    propertyIds: [],
    contactEmail: "user@mydomain.com",
    thing: "MyThing1",
    sampleData: { temperature: 23.5, humidity: 65.0 },
  },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Run an interactive scenario at a command prompt.

```
class IoTSitewiseGettingStarted:
    """
    A scenario that demonstrates how to use Boto3 to manage IoT physical assets
 using
    the AWS IoT SiteWise.
    """

    def __init__(
        self,
        iot_sitewise_wrapper: IoTSitewiseWrapper,
        cloud_formation_resource: ServiceResource,
    ):
        self.iot_sitewise_wrapper = iot_sitewise_wrapper
        self.cloud_formation_resource = cloud_formation_resource
        self.stack = None
        self.asset_model_id = None
        self.asset_id = None
        self.portal_id = None
        self.gateway_id = None

    def run(self) -> None:
        """
        Runs the scenario.
        """
        print(
            """
AWS IoT SiteWise is a fully managed software-as-a-service (SaaS) that
makes it easy to collect, store, organize, and monitor data from industrial
 equipment and processes.
It is designed to help industrial and manufacturing organizations collect data
 from their equipment and
processes, and use that data to make informed decisions about their operations.

One of the key features of AWS IoT SiteWise is its ability to connect to a wide
 range of industrial
equipment and systems, including programmable logic controllers (PLCs), sensors,
 and other
industrial devices. It can collect data from these devices and organize it into a
 unified data model,
making it easier to analyze and gain insights from the data. AWS IoT SiteWise
 also provides tools for
visualizing the data, setting up alarms and alerts, and generating reports.
```

```
Another key feature of AWS IoT SiteWise is its ability to scale to handle large
 volumes of data.
It can collect and store data from thousands of devices and process millions of
 data points per second,
making it suitable for large-scale industrial operations. Additionally, AWS IoT
 SiteWise is designed
to be secure and compliant, with features like role-based access controls, data
 encryption,
and integration with other AWS services for additional security and compliance
 features.

Let's get started...
        """
        )
        press_enter_to_continue()
        print_dashes()
        print(f"")
        print(
            f"Use AWS CloudFormation to create an IAM role that is required for
 this scenario."
        )
        template_file = IoTSitewiseGettingStarted.get_template_as_string()

        self.stack = self.deploy_cloudformation_stack(
            "python-iot-sitewise-basics", template_file
        )
        outputs = self.stack.outputs
        iam_role = None

        for output in outputs:
            if output.get("OutputKey") == "SitewiseRoleArn":
                iam_role = output.get("OutputValue")

        if iam_role is None:
            error_string = f"Failed to retrieve iam_role from CloudFormation
 stack."
            logger.error(error_string)
            raise ValueError(error_string)

        print(f"The ARN of the IAM role is {iam_role}")
        print_dashes()
        print_dashes()
        print(f"1. Create an AWS SiteWise Asset Model")
```

```
        print(
            """
An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such
 as equipment,
processes, and systems, that exist in an industrial environment. This model
 provides a structured and
hierarchical representation of these assets, allowing users to define the
 relationships and values
of each asset.

This scenario creates two asset model values: temperature and humidity.
        """
        )
        press_enter_to_continue()
        asset_model_name = "MyAssetModel1"
        temperature_property_name = "temperature"
        humidity_property_name = "humidity"
        try:
            properties = [
                {
                    "name": temperature_property_name,
                    "dataType": "DOUBLE",
                    "type": {
                        "measurement": {},
                    },
                },
                {
                    "name": humidity_property_name,
                    "dataType": "DOUBLE",
                    "type": {
                        "measurement": {},
                    },
                },
            ]
            self.asset_model_id = self.iot_sitewise_wrapper.create_asset_model(
                asset_model_name, properties
            )
            print(
                f"Asset Model successfully created. Asset Model ID:
 {self.asset_model_id}. "
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
```

```
                self.asset_model_id =
 self.get_model_id_for_model_name(asset_model_name)
                print(
                    f"Asset Model {asset_model_name} already exists. Asset Model
 ID: {self.asset_model_id}. "
                )
            else:
                raise

        press_enter_to_continue()
        print_dashes()
        print(f"2. Create an AWS IoT SiteWise Asset")
        print(
            """
The IoT SiteWise model that we just created defines the structure and metadata
 for your physical assets.
Now we create an asset from the asset model.

        """
        )
        press_enter_to_continue()

        self.asset_id = self.iot_sitewise_wrapper.create_asset(
            "MyAsset1", self.asset_model_id
        )

        print(f"Asset created with ID: {self.asset_id}")
        press_enter_to_continue()
        print_dashes()
        print_dashes()
        print(f"3. Retrieve the property ID values")
        print(
            """
To send data to an asset, we need to get the property ID values. In this
 scenario, we access the
temperature and humidity property ID values.
        """
        )
        press_enter_to_continue()
        property_ids = self.iot_sitewise_wrapper.list_asset_model_properties(
            self.asset_model_id
        )
        humidity_property_id = None
        temperature_property_id = None
```

```
        for property_id in property_ids:
            if property_id.get("name") == humidity_property_name:
                humidity_property_id = property_id.get("id")
            elif property_id.get("name") == temperature_property_name:
                temperature_property_id = property_id.get("id")
        if humidity_property_id is None or temperature_property_id is None:
            error_string = f"Failed to retrieve property IDs from Asset Model."
            logger.error(error_string)
            raise ValueError(error_string)

        print(f"The Humidity property Id is {humidity_property_id}")
        print(f"The Temperature property Id is {temperature_property_id}")
        press_enter_to_continue()
        print_dashes()
        print_dashes()

        print(f"4. Send data to an AWS IoT SiteWise Asset")
        print(
            """
By sending data to an IoT SiteWise Asset, you can aggregate data from
multiple sources, normalize the data into a standard format, and store it in a
centralized location. This makes it easier to analyze and gain insights from the
 data.

In this example, we generate sample temperature and humidity data and send it to
 the AWS IoT SiteWise asset.

        """
        )
        press_enter_to_continue()

        values = [
            {
                "propertyId": humidity_property_id,
                "valueType": "doubleValue",
                "value": 65.0,
            },
            {
                "propertyId": temperature_property_id,
                "valueType": "doubleValue",
                "value": 23.5,
            },
        ]
```

```
        self.iot_sitewise_wrapper.batch_put_asset_property_value(self.asset_id,
 values)
        print(f"Data sent successfully.")

        press_enter_to_continue()
        print_dashes()
        print_dashes()

        print(f"5. Retrieve the value of the IoT SiteWise Asset property")
        print(
            """
IoT SiteWise is an AWS service that allows you to collect, process, and analyze
 industrial data
from connected equipment and sensors. One of the key benefits of reading an IoT
 SiteWise property
is the ability to gain valuable insights from your industrial data.

        """
        )
        press_enter_to_continue()

        property_value = self.iot_sitewise_wrapper.get_asset_property_value(
            self.asset_id, temperature_property_id
        )
        print(f"The property name is '{temperature_property_name}'.")

        print(
            f"The value of this property is: {property_value['value']
['doubleValue']}"
        )
        press_enter_to_continue()

        property_value = self.iot_sitewise_wrapper.get_asset_property_value(
            self.asset_id, humidity_property_id
        )
        print(f"The property name is '{humidity_property_name}'.")
        print(
            f"The value of this property is: {property_value['value']
['doubleValue']}"
        )
        press_enter_to_continue()
        print_dashes()
        print_dashes()
```

```
        print(f"6. Create an IoT SiteWise Portal")
        print(
            """
An IoT SiteWise Portal allows you to aggregate data from multiple industrial
 sources,
such as sensors, equipment, and control systems, into a centralized platform.
        """
        )

        press_enter_to_continue()
        contact_email = q.ask("Enter a contact email for the portal:",
 q.non_empty)
        print("Creating the portal. The portal may take a while to become
 active.")
        self.portal_id = self.iot_sitewise_wrapper.create_portal(
            "MyPortal1", iam_role, contact_email
        )
        print(f"Portal created successfully. Portal ID {self.portal_id}")
        press_enter_to_continue()
        print_dashes()
        print_dashes()

        print(f"7. Describe the Portal")
        print(
            """
In this step, we get a description of the portal and display the portal URL.
        """
        )
        press_enter_to_continue()
        portal_description =
 self.iot_sitewise_wrapper.describe_portal(self.portal_id)
        print(f"Portal URL: {portal_description['portalStartUrl']}")
        press_enter_to_continue()
        print_dashes()
        print_dashes()

        print(f"8. Create an IoT SiteWise Gateway")
        press_enter_to_continue()
        self.gateway_id = self.iot_sitewise_wrapper.create_gateway(
            "MyGateway1", "MyThing1"
        )
        print(f"Gateway creation completed successfully. id is
 {self.gateway_id}")
        print_dashes()
```

```
        print_dashes()
        print(f"9. Describe the IoT SiteWise Gateway")
        press_enter_to_continue()

        gateway_description = self.iot_sitewise_wrapper.describe_gateway(
            self.gateway_id
        )
        print(f"Gateway Name: {gateway_description['gatewayName']}")
        print(f"Gateway ARN: {gateway_description['gatewayArn']}")
        print(f"Gateway Platform:\n{gateway_description['gatewayPlatform']}")
        print(f"Gateway Creation Date: {gateway_description['gatewayArn']}")
        print_dashes()
        print_dashes()

        print(f"10. Delete the AWS IoT SiteWise Assets")
        if q.ask("Would you like to delete the IoT SiteWise Assets? (y/n)",
 q.is_yesno):
            self.cleanup()
        else:
            print(f"The resources will not be deleted.")
        print_dashes()
        print_dashes()
        print(f"This concludes the AWS IoT SiteWise Scenario")

    def cleanup(self) -> None:
        """
        Deletes the CloudFormation stack and the resources created for the demo.
        """

        if self.gateway_id is not None:
            self.iot_sitewise_wrapper.delete_gateway(self.gateway_id)
            print(f"Deleted gateway with id {self.gateway_id}.")
            self.gateway_id = None
        if self.portal_id is not None:
            self.iot_sitewise_wrapper.delete_portal(self.portal_id)
            print(f"Deleted portal with id {self.portal_id}.")
            self.portal_id = None
        if self.asset_id is not None:
            self.iot_sitewise_wrapper.delete_asset(self.asset_id)
            print(f"Deleted asset with id {self.asset_id}.")
            self.iot_sitewise_wrapper.wait_asset_deleted(self.asset_id)
            self.asset_id = None
        if self.asset_model_id is not None:
            self.iot_sitewise_wrapper.delete_asset_model(self.asset_model_id)
```

```
            print(f"Deleted asset model with id {self.asset_model_id}.")
            self.asset_model_id = None
        if self.stack is not None:
            stack = self.stack
            self.stack = None
            self.destroy_cloudformation_stack(stack)

    def deploy_cloudformation_stack(
        self, stack_name: str, cfn_template: str
    ) -> ServiceResource:
        """
        Deploys prerequisite resources used by the scenario. The resources are
        defined in the associated `SitewiseRoles-template.yaml` AWS
 CloudFormation script and are deployed
        as a CloudFormation stack, so they can be easily managed and destroyed.

        :param stack_name: The name of the CloudFormation stack.
        :param cfn_template: The CloudFormation template as a string.
        :return: The CloudFormation stack resource.
        """
        print(f"Deploying CloudFormation stack: {stack_name}.")
        stack = self.cloud_formation_resource.create_stack(
            StackName=stack_name,
            TemplateBody=cfn_template,
            Capabilities=["CAPABILITY_NAMED_IAM"],
        )
        print(f"CloudFormation stack creation started: {stack_name}")
        print("Waiting for CloudFormation stack creation to complete...")
        waiter = self.cloud_formation_resource.meta.client.get_waiter(
            "stack_create_complete"
        )
        waiter.wait(StackName=stack.name)
        stack.load()
        print("CloudFormation stack creation complete.")

        return stack

    def destroy_cloudformation_stack(self, stack: ServiceResource) -> None:
        """
        Destroys the resources managed by the CloudFormation stack, and the
 CloudFormation
        stack itself.
```

```
        :param stack: The CloudFormation stack that manages the example
 resources.
        """
        print(
            f"CloudFormation stack '{stack.name}' is being deleted. This may take
 a few minutes."
        )
        stack.delete()
        waiter = self.cloud_formation_resource.meta.client.get_waiter(
            "stack_delete_complete"
        )
        waiter.wait(StackName=stack.name)
        print(f"CloudFormation stack '{stack.name}' has been deleted.")

    @staticmethod
    def get_template_as_string() -> str:
        """
        Returns a string containing this scenario's CloudFormation template.
        """
        template_file_path = os.path.join(script_dir, "SitewiseRoles-
template.yaml")
        file = open(template_file_path, "r")
        return file.read()

    def get_model_id_for_model_name(self, model_name: str) -> str:
        """
        Returns the model ID for the given model name.

        :param model_name: The name of the model.
        :return: The model ID.
        """
        model_id = None
        asset_models = self.iot_sitewise_wrapper.list_asset_models()
        for asset_model in asset_models:
            if asset_model["name"] == model_name:
                model_id = asset_model["id"]
                break
        return model_id
```

IoTSitewiseWrapper class that wraps AWS IoT SiteWise actions.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                              access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def create_asset_model(
        self, asset_model_name: str, properties: List[Dict[str, Any]]
    ) -> str:
        """
        Creates an AWS IoT SiteWise Asset Model.

        :param asset_model_name: The name of the asset model to create.
        :param properties: The property definitions of the asset model.
        :return: The ID of the created asset model.
        """
        try:
            response = self.iotsitewise_client.create_asset_model(
                assetModelName=asset_model_name,
                assetModelDescription="This is a sample asset model
 description.",
```

```
                assetModelProperties=properties,
            )
            asset_model_id = response["assetModelId"]
            waiter = self.iotsitewise_client.get_waiter("asset_model_active")
            waiter.wait(assetModelId=asset_model_id)
            return asset_model_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Asset model %s already exists.", asset_model_name)
            else:
                logger.error(
                    "Error creating asset model %s. Here's why %s",
                    asset_model_name,
                    err.response["Error"]["Message"],
                )
            raise


    def create_asset(self, asset_name: str, asset_model_id: str) -> str:
        """
        Creates an AWS IoT SiteWise Asset.

        :param asset_name: The name of the asset to create.
        :param asset_model_id: The ID of the asset model to associate with the 
asset.
        :return: The ID of the created asset.
        """
        try:
            response = self.iotsitewise_client.create_asset(
                assetName=asset_name, assetModelId=asset_model_id
            )
            asset_id = response["assetId"]
            waiter = self.iotsitewise_client.get_waiter("asset_active")
            waiter.wait(assetId=asset_id)
            return asset_id
        except ClientError as err:
            if err.response["Error"] == "ResourceNotFoundException":
                logger.error("Asset model %s does not exist.", asset_model_id)
            else:
                logger.error(
                    "Error creating asset %s. Here's why %s",
                    asset_name,
                    err.response["Error"]["Message"],
                )
```

```
            raise


    def list_asset_models(self) -> List[Dict[str, Any]]:
        """
        Lists all AWS IoT SiteWise Asset Models.

        :return: A list of dictionaries containing information about each asset
 model.

        """
        try:
            asset_models = []
            paginator =
 self.iotsitewise_client.get_paginator("list_asset_models")
            pages = paginator.paginate()
            for page in pages:
                asset_models.extend(page["assetModelSummaries"])
            return asset_models
        except ClientError as err:
            logger.error(
                "Error listing asset models. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise


    def list_asset_model_properties(self, asset_model_id: str) -> List[Dict[str,
 Any]]:
        """
        Lists all AWS IoT SiteWise Asset Model Properties.

        :param asset_model_id: The ID of the asset model to list values for.
        :return: A list of dictionaries containing information about each asset
 model property.
        """
        try:
            asset_model_properties = []
            paginator = self.iotsitewise_client.get_paginator(
                "list_asset_model_properties"
            )
            pages = paginator.paginate(assetModelId=asset_model_id)
            for page in pages:
```

```
 asset_model_properties.extend(page["assetModelPropertySummaries"])
            return asset_model_properties
        except ClientError as err:
            logger.error(
                "Error listing asset model values. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise


    def batch_put_asset_property_value(
        self, asset_id: str, values: List[Dict[str, str]]
    ) -> None:
        """
        Sends data to an AWS IoT SiteWise Asset.

        :param asset_id: The asset ID.
        :param values: A list of dictionaries containing the values in the form
                        {propertyId : property_id,
                        valueType : [stringValue|integerValue|doubleValue|
booleanValue],
                        value : the_value}.
        """
        try:
            entries = self.properties_to_values(asset_id, values)

 self.iotsitewise_client.batch_put_asset_property_value(entries=entries)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Asset %s does not exist.", asset_id)
            else:
                logger.error(
                    "Error sending data to asset. Here's why %s",
                    err.response["Error"]["Message"],
                )
            raise


    def properties_to_values(
        self, asset_id: str, values: list[dict[str, Any]]
    ) -> list[dict[str, Any]]:
        """
```

```
        Utility function to convert a values list to the entries parameter for
 batch_put_asset_property_value.
        :param asset_id : The asset ID.
        :param values : A list of dictionaries containing the values in the form
                        {propertyId : property_id,
                        valueType : [stringValue|integerValue|doubleValue|
booleanValue],
                        value : the_value}.
        :return: An entries list to pass as the 'entries' parameter to
 batch_put_asset_property_value.
        """
        entries = []
        for value in values:
            epoch_ns = time.time_ns()
            self.entry_id += 1
            if value["valueType"] == "stringValue":
                property_value = {"stringValue": value["value"]}
            elif value["valueType"] == "integerValue":
                property_value = {"integerValue": value["value"]}
            elif value["valueType"] == "booleanValue":
                property_value = {"booleanValue": value["value"]}
            elif value["valueType"] == "doubleValue":
                property_value = {"doubleValue": value["value"]}
            else:
                raise ValueError("Invalid valueType: %s", value["valueType"])
            entry = {
                "entryId": f"{self.entry_id}",
                "assetId": asset_id,
                "propertyId": value["propertyId"],
                "propertyValues": [
                    {
                        "value": property_value,
                        "timestamp": {
                            "timeInSeconds": int(epoch_ns / 1000000000),
                            "offsetInNanos": epoch_ns % 1000000000,
                        },
                    }
                ],
            }
            entries.append(entry)
        return entries


    def get_asset_property_value(
```

```
        self, asset_id: str, property_id: str
    ) -> Dict[str, Any]:
        """
        Gets the value of an AWS IoT SiteWise Asset Property.

        :param asset_id: The ID of the asset.
        :param property_id: The ID of the property.
        :return: A dictionary containing the value of the property.
        """
        try:
            response = self.iotsitewise_client.get_asset_property_value(
                assetId=asset_id, propertyId=property_id
            )
            return response["propertyValue"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Asset %s or property %s does not exist.", asset_id,
 property_id
                )
            else:
                logger.error(
                    "Error getting asset property value. Here's why %s",
                    err.response["Error"]["Message"],
                )
            raise


    def create_portal(
        self, portal_name: str, iam_role_arn: str, portal_contact_email: str
    ) -> str:
        """
        Creates an AWS IoT SiteWise Portal.

        :param portal_name: The name of the portal to create.
        :param iam_role_arn: The ARN of an IAM role.
        :param portal_contact_email: The contact email of the portal.
        :return: The ID of the created portal.
        """
        try:
            response = self.iotsitewise_client.create_portal(
                portalName=portal_name,
                roleArn=iam_role_arn,
                portalContactEmail=portal_contact_email,
```

```
            )
            portal_id = response["portalId"]
            waiter = self.iotsitewise_client.get_waiter("portal_active")
            waiter.wait(portalId=portal_id, WaiterConfig={"MaxAttempts": 40})
            return portal_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Portal %s already exists.", portal_name)
            else:
                logger.error(
                    "Error creating portal %s. Here's why %s",
                    portal_name,
                    err.response["Error"]["Message"],
                )
            raise


    def describe_portal(self, portal_id: str) -> Dict[str, Any]:
        """
        Describes an AWS IoT SiteWise Portal.

        :param portal_id: The ID of the portal to describe.
        :return: A dictionary containing information about the portal.
        """
        try:
            response =
 self.iotsitewise_client.describe_portal(portalId=portal_id)
            return response
        except ClientError as err:
            logger.error(
                "Error describing portal %s. Here's why %s",
                portal_id,
                err.response["Error"]["Message"],
            )
            raise


    def create_gateway(self, gateway_name: str, my_thing: str) -> str:
        """
        Creates an AWS IoT SiteWise Gateway.

        :param gateway_name: The name of the gateway to create.
        :param my_thing: The core device thing name.
        :return: The ID of the created gateway.
```

```
        """
        try:
            response = self.iotsitewise_client.create_gateway(
                gatewayName=gateway_name,
                gatewayPlatform={
                    "greengrassV2": {"coreDeviceThingName": my_thing},
                },
                tags={"Environment": "Production"},
            )
            gateway_id = response["gatewayId"]
            return gateway_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Gateway %s already exists.", gateway_name)
            else:
                logger.error(
                    "Error creating gateway %s. Here's why %s",
                    gateway_name,
                    err.response["Error"]["Message"],
                )
            raise


    def describe_gateway(self, gateway_id: str) -> Dict[str, Any]:
        """
        Describes an AWS IoT SiteWise Gateway.

        :param gateway_id: The ID of the gateway to describe.
        :return: A dictionary containing information about the gateway.
        """
        try:
            response = 
 self.iotsitewise_client.describe_gateway(gatewayId=gateway_id)
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Gateway %s does not exist.", gateway_id)
            else:
                logger.error(
                    "Error describing gateway %s. Here's why %s",
                    gateway_id,
                    err.response["Error"]["Message"],
                )
            raise
```

```
    def delete_gateway(self, gateway_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Gateway.

        :param gateway_id: The ID of the gateway to delete.
        """
        try:
            self.iotsitewise_client.delete_gateway(gatewayId=gateway_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Gateway %s does not exist.", gateway_id)
            else:
                logger.error(
                    "Error deleting gateway %s. Here's why %s",
                    gateway_id,
                    err.response["Error"]["Message"],
                )
            raise


    def delete_portal(self, portal_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Portal.

        :param portal_id: The ID of the portal to delete.
        """
        try:
            self.iotsitewise_client.delete_portal(portalId=portal_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Portal %s does not exist.", portal_id)
            else:
                logger.error(
                    "Error deleting portal %s. Here's why %s",
                    portal_id,
                    err.response["Error"]["Message"],
                )
            raise


    def delete_asset(self, asset_id: str) -> None:
        """
```

```
        Deletes an AWS IoT SiteWise Asset.

        :param asset_id: The ID of the asset to delete.
        """
        try:
            self.iotsitewise_client.delete_asset(assetId=asset_id)
        except ClientError as err:
            logger.error(
                "Error deleting asset %s. Here's why %s",
                asset_id,
                err.response["Error"]["Message"],
            )
            raise


    def delete_asset_model(self, asset_model_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Asset Model.

        :param asset_model_id: The ID of the asset model to delete.
        """
        try:

self.iotsitewise_client.delete_asset_model(assetModelId=asset_model_id)
        except ClientError as err:
            logger.error(
                "Error deleting asset model %s. Here's why %s",
                asset_model_id,
                err.response["Error"]["Message"],
            )
            raise


    def wait_asset_deleted(self, asset_id: str) -> None:
        """
        Waits for an AWS IoT SiteWise Asset to be deleted.

        :param asset_id: The ID of the asset to wait for.
        """
        try:
            waiter = self.iotsitewise_client.get_waiter("asset_not_exists")
            waiter.wait(assetId=asset_id)
        except ClientError as err:
            logger.error(
```

```
                "Error waiting for asset %s to be deleted. Here's why %s",
                asset_id,
                err.response["Error"]["Message"],
            )
            raise
```

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](). This topic also includes information about getting started and details about previous SDK versions.

Actions for AWS IoT SiteWise using AWS SDKs

The following code examples demonstrate how to perform individual AWS IoT SiteWise actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the AWS IoT SiteWise API Reference.

Examples

- Use BatchPutAssetPropertyValue with an AWS SDK or CLI
- Use CreateAsset with an AWS SDK or CLI
- Use CreateAssetModel with an AWS SDK or CLI
- Use CreateGateway with an AWS SDK or CLI
- Use CreatePortal with an AWS SDK or CLI
- Use DeleteAsset with an AWS SDK or CLI
- Use DeleteAssetModel with an AWS SDK or CLI
- Use DeleteGateway with an AWS SDK or CLI
- Use DeletePortal with an AWS SDK or CLI
- Use DescribeAssetModel with an AWS SDK or CLI
- Use DescribeGateway with an AWS SDK or CLI
- Use DescribePortal with an AWS SDK or CLI
- Use GetAssetPropertyValue with an AWS SDK or CLI

- Use ListAssetModels with an AWS SDK or CLI

Use BatchPutAssetPropertyValue with an AWS SDK or CLI

The following code examples show how to use `BatchPutAssetPropertyValue`.

CLI

AWS CLI

To send data to asset properties

The following `batch-put-asset-property-value` example sends power and temperature data to the asset properties identified by property aliases.

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

Contents of `batch-put-asset-property-value.json`:

```
{
    "entries": [
        {
            "entryId": "1575691200-company-windfarm-3-turbine-7-power",
            "propertyAlias": "company-windfarm-3-turbine-7-power",
            "propertyValues": [
                {
                    "value": {
                        "doubleValue": 4.92
                    },
                    "timestamp": {
                        "timeInSeconds": 1575691200
                    },
                    "quality": "GOOD"
                }
            ]
        },
        {
            "entryId": "1575691200-company-windfarm-3-turbine-7-temperature",
            "propertyAlias": "company-windfarm-3-turbine-7-temperature",
            "propertyValues": [
                {
```

```
                    "value": {
                        "integerValue": 38
                    },
                    "timestamp": {
                        "timeInSeconds": 1575691200
                    }
                }
            ]
        }
    ]
}
```

Output:

```
{
    "errorEntries": []
}
```

For more information, see Ingesting data using the AWS IoT SiteWise API in the *AWS IoT SiteWise User Guide*.

- For API details, see BatchPutAssetPropertyValue in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Sends data to the SiteWise service.
     *
     * @param assetId        the ID of the asset to which the data will be sent.
     * @param tempPropertyId the ID of the temperature property.
     * @param humidityPropId the ID of the humidity property.
     * @return a {@link CompletableFuture} that represents a {@link
 BatchPutAssetPropertyValueResponse} result. The
```

```
     *         calling code can attach callbacks, then handle the result or
 exception by calling
     *         {@link CompletableFuture#join()} or {@link
 CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps it
     *         available to the calling code as a {@link CompletionException}. By
 calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<BatchPutAssetPropertyValueResponse>
 sendDataToSiteWiseAsync(String assetId, String tempPropertyId, String
 humidityPropId) {
        Map<String, Double> sampleData = generateSampleData();
        long timestamp = Instant.now().toEpochMilli();

        TimeInNanos time = TimeInNanos.builder()
            .timeInSeconds(timestamp / 1000)
            .offsetInNanos((int) ((timestamp % 1000) * 1000000))
            .build();

        BatchPutAssetPropertyValueRequest request =
 BatchPutAssetPropertyValueRequest.builder()
            .entries(Arrays.asList(
                PutAssetPropertyValueEntry.builder()
                    .entryId("entry-3")
                    .assetId(assetId)
                    .propertyId(tempPropertyId)
                    .propertyValues(Arrays.asList(
                        AssetPropertyValue.builder()
                            .value(Variant.builder()
                                .doubleValue(sampleData.get("Temperature"))
                                .build())
                            .timestamp(time)
                            .build()
                    ))
                    .build(),
                PutAssetPropertyValueEntry.builder()
                    .entryId("entry-4")
                    .assetId(assetId)
                    .propertyId(humidityPropId)
                    .propertyValues(Arrays.asList(
```

```
                         AssetPropertyValue.builder()
                             .value(Variant.builder()
                                 .doubleValue(sampleData.get("Humidity"))
                                 .build())
                             .timestamp(time)
                             .build()
                     ))
                     .build()
             ))
             .build();

         return getAsyncClient().batchPutAssetPropertyValue(request)
             .whenComplete((response, exception) -> {
                 if (exception != null) {
                     logger.error("An exception occurred: {}",
 exception.getCause().getMessage());
                 }
             });
     }
```

- For API details, see BatchPutAssetPropertyValue in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  BatchPutAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Batch put asset property values.
 * @param {{ entries : array }}
```

```
 */
export const main = async ({ entries }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new BatchPutAssetPropertyValueCommand({
        entries: entries,
      }),
    );
    console.log("Asset properties batch put successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(`${caught.message}. A resource could not be found.`);
    } else {
      throw caught;
    }
  }
};
```

- For API details, see BatchPutAssetPropertyValue in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
```

```
                            access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def batch_put_asset_property_value(
        self, asset_id: str, values: List[Dict[str, str]]
    ) -> None:
        """
        Sends data to an AWS IoT SiteWise Asset.

        :param asset_id: The asset ID.
        :param values: A list of dictionaries containing the values in the form
                        {propertyId : property_id,
                        valueType : [stringValue|integerValue|doubleValue|
booleanValue],
                        value : the_value}.
        """
        try:
            entries = self.properties_to_values(asset_id, values)

 self.iotsitewise_client.batch_put_asset_property_value(entries=entries)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Asset %s does not exist.", asset_id)
            else:
                logger.error(
                    "Error sending data to asset. Here's why %s",
                    err.response["Error"]["Message"],
                )
```

```
            raise
```

A helper function to generate the entries parameter from a values list.

```
    def properties_to_values(
        self, asset_id: str, values: list[dict[str, Any]]
    ) -> list[dict[str, Any]]:
        """
        Utility function to convert a values list to the entries parameter for
 batch_put_asset_property_value.
        :param asset_id : The asset ID.
        :param values : A list of dictionaries containing the values in the form
                        {propertyId : property_id,
                        valueType : [stringValue|integerValue|doubleValue|
booleanValue],
                        value : the_value}.
        :return: An entries list to pass as the 'entries' parameter to
 batch_put_asset_property_value.
        """
        entries = []
        for value in values:
            epoch_ns = time.time_ns()
            self.entry_id += 1
            if value["valueType"] == "stringValue":
                property_value = {"stringValue": value["value"]}
            elif value["valueType"] == "integerValue":
                property_value = {"integerValue": value["value"]}
            elif value["valueType"] == "booleanValue":
                property_value = {"booleanValue": value["value"]}
            elif value["valueType"] == "doubleValue":
                property_value = {"doubleValue": value["value"]}
            else:
                raise ValueError("Invalid valueType: %s", value["valueType"])
            entry = {
                "entryId": f"{self.entry_id}",
                "assetId": asset_id,
                "propertyId": value["propertyId"],
                "propertyValues": [
                    {
                        "value": property_value,
                        "timestamp": {
```

```
                        "timeInSeconds": int(epoch_ns / 1000000000),
                        "offsetInNanos": epoch_ns % 1000000000,
                    },
                }
            ],
        }
        entries.append(entry)
    return entries
```

Here is an example of a values list to pass to the helper function.

```
    values = [
        {
            "propertyId": humidity_property_id,
            "valueType": "doubleValue",
            "value": 65.0,
        },
        {
            "propertyId": temperature_property_id,
            "valueType": "doubleValue",
            "value": 23.5,
        },
    ]
```

- For API details, see BatchPutAssetPropertyValue in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateAsset` with an AWS SDK or CLI

The following code examples show how to use `CreateAsset`.

CLI

AWS CLI

To create an asset

The following `create-asset` example creates a wind turbine asset from a wind turbine asset model.

```
aws iotsitewise create-asset \
    --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE \
    --asset-name "Wind Turbine 1"
```

Output:

```
{
    "assetId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
    "assetArn": "arn:aws:iotsitewise:us-west-2:123456789012:asset/
a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
    "assetStatus": {
        "state": "CREATING"
    }
}
```

For more information, see Creating assets in the *AWS IoT SiteWise User Guide*.

- For API details, see CreateAsset in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
```

```
     * Creates an asset with the specified name and asset model Id.
     *
     * @param assetName    the name of the asset to create.
     * @param assetModelId the Id of the asset model to associate with the asset.
     * @return a {@link CompletableFuture} that represents a {@link
 CreateAssetResponse} result. The calling code can
     *         attach callbacks, then handle the result or exception by calling
 {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps it
     *         available to the calling code as a {@link CompletionException}. By
 calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<CreateAssetResponse> createAssetAsync(String
 assetName, String assetModelId) {
        CreateAssetRequest createAssetRequest = CreateAssetRequest.builder()
            .assetModelId(assetModelId)
            .assetDescription("Created using the AWS SDK for Java")
            .assetName(assetName)
            .build();

        return getAsyncClient().createAsset(createAssetRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create asset: {}",
 exception.getCause().getMessage());
                }
            });
    }
```

- For API details, see CreateAsset in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
import {
  CreateAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetCommand({
        assetName: assetName, // The name to give the Asset.
        assetModelId: assetModelId, // The ID of the asset model from which to
 create the asset.
      }),
    );
    console.log("Asset created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
      );
    } else {
      throw caught;
    }
  }
```

```
};
```

- For API details, see [CreateAsset](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                          access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)
```

```
    def create_asset(self, asset_name: str, asset_model_id: str) -> str:
        """
        Creates an AWS IoT SiteWise Asset.

        :param asset_name: The name of the asset to create.
        :param asset_model_id: The ID of the asset model to associate with the
 asset.
        :return: The ID of the created asset.
        """
        try:
            response = self.iotsitewise_client.create_asset(
                assetName=asset_name, assetModelId=asset_model_id
            )
            asset_id = response["assetId"]
            waiter = self.iotsitewise_client.get_waiter("asset_active")
            waiter.wait(assetId=asset_id)
            return asset_id
        except ClientError as err:
            if err.response["Error"] == "ResourceNotFoundException":
                logger.error("Asset model %s does not exist.", asset_model_id)
            else:
                logger.error(
                    "Error creating asset %s. Here's why %s",
                    asset_name,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see CreateAsset in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateAssetModel` with an AWS SDK or CLI

The following code examples show how to use `CreateAssetModel`.

CLI

AWS CLI

To create an asset model

The following `create-asset-model` example creates an asset model that defines a wind turbine with the following properties:

Serial number - The serial number of a wind turbineGenerated power - The generated power data stream from a wind turbineTemperature C - The temperature data stream from a wind turbine in CelsiusTemperature F - The mapped temperature data points from Celsius to Fahrenheit

```
aws iotsitewise create-asset-model \
    --cli-input-json file://create-wind-turbine-model.json
```

Contents of `create-wind-turbine-model.json`:

```
{
    "assetModelName": "Wind Turbine Model",
    "assetModelDescription": "Represents a wind turbine",
    "assetModelProperties": [
        {
            "name": "Serial Number",
            "dataType": "STRING",
            "type": {
                "attribute": {}
            }
        },
        {
            "name": "Generated Power",
            "dataType": "DOUBLE",
            "unit": "kW",
            "type": {
                "measurement": {}
            }
        },
        {
            "name": "Temperature C",
            "dataType": "DOUBLE",
            "unit": "Celsius",
            "type": {
```

```
                    "measurement": {}
                }
            },
            {
                "name": "Temperature F",
                "dataType": "DOUBLE",
                "unit": "Fahrenheit",
                "type": {
                    "transform": {
                        "expression": "temp_c * 9 / 5 + 32",
                        "variables": [
                            {
                                "name": "temp_c",
                                "value": {
                                    "propertyId": "Temperature C"
                                }
                            }
                        ]
                    }
                }
            },
            {
                "name": "Total Generated Power",
                "dataType": "DOUBLE",
                "unit": "kW",
                "type": {
                    "metric": {
                        "expression": "sum(power)",
                        "variables": [
                            {
                                "name": "power",
                                "value": {
                                    "propertyId": "Generated Power"
                                }
                            }
                        ],
                        "window": {
                            "tumbling": {
                                "interval": "1h"
                            }
                        }
                    }
                }
            }
```

```
    ]
}
```

Output:


```
{
    "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
    "assetModelArn": "arn:aws:iotsitewise:us-west-2:123456789012:asset-model/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
    "assetModelStatus": {
        "state": "CREATING"
    }
}
```


For more information, see Defining asset models in the *AWS IoT SiteWise User Guide*.

- For API details, see CreateAssetModel in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Creates an asset model.
     *
     * @param name the name of the asset model to create.
     * @return a {@link CompletableFuture} that represents a {@link
 CreateAssetModelResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps it
```

```
     *         available to the calling code as a {@link CompletionException}. By
calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<CreateAssetModelResponse>
createAssetModelAsync(String name) {
        PropertyType humidity = PropertyType.builder()
            .measurement(Measurement.builder().build())
            .build();

        PropertyType temperaturePropertyType = PropertyType.builder()
            .measurement(Measurement.builder().build())
            .build();

        AssetModelPropertyDefinition temperatureProperty =
AssetModelPropertyDefinition.builder()
            .name("Temperature")
            .dataType(PropertyDataType.DOUBLE)
            .type(temperaturePropertyType)
            .build();

        AssetModelPropertyDefinition humidityProperty =
AssetModelPropertyDefinition.builder()
            .name("Humidity")
            .dataType(PropertyDataType.DOUBLE)
            .type(humidity)
            .build();

        CreateAssetModelRequest createAssetModelRequest =
CreateAssetModelRequest.builder()
            .assetModelName(name)
            .assetModelDescription("This is my asset model")
            .assetModelProperties(temperatureProperty, humidityProperty)
            .build();

        return getAsyncClient().createAssetModel(createAssetModelRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create asset model: {} ",
exception.getCause().getMessage());
                }
            });
    }
```

- For API details, see CreateAssetModel in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  CreateAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset Model.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetModelName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetModelCommand({
        assetModelName: assetModelName, // The name to give the Asset Model.
      }),
    );
    console.log("Asset model created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the asset model.`,
      );
    } else {
      throw caught;
    }
```

```
  }
};
```

- For API details, see [CreateAssetModel](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                              access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)
```

```
    def create_asset_model(
        self, asset_model_name: str, properties: List[Dict[str, Any]]
    ) -> str:
        """
        Creates an AWS IoT SiteWise Asset Model.

        :param asset_model_name: The name of the asset model to create.
        :param properties: The property definitions of the asset model.
        :return: The ID of the created asset model.
        """
        try:
            response = self.iotsitewise_client.create_asset_model(
                assetModelName=asset_model_name,
                assetModelDescription="This is a sample asset model
 description.",
                assetModelProperties=properties,
            )
            asset_model_id = response["assetModelId"]
            waiter = self.iotsitewise_client.get_waiter("asset_model_active")
            waiter.wait(assetModelId=asset_model_id)
            return asset_model_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Asset model %s already exists.", asset_model_name)
            else:
                logger.error(
                    "Error creating asset model %s. Here's why %s",
                    asset_model_name,
                    err.response["Error"]["Message"],
                )
            raise
```

Here is an example of a properties list to pass to the function.

```
            properties = [
                {
                    "name": temperature_property_name,
                    "dataType": "DOUBLE",
                    "type": {
```

```
                            "measurement": {},
                        },
                    },
                    {
                        "name": humidity_property_name,
                        "dataType": "DOUBLE",
                        "type": {
                            "measurement": {},
                        },
                    },
                ]
```

- For API details, see CreateAssetModel in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateGateway` with an AWS SDK or CLI

The following code examples show how to use `CreateGateway`.

CLI

AWS CLI

To create a gateway

The following `create-gateway` example creates a gateway that runs on AWS IoT Greengrass.

```
aws iotsitewise create-gateway \
    --gateway-name ExampleCorpGateway \
    --gateway-platform greengrass={groupArn=arn:aws:greengrass:us-
west-2:123456789012:/greengrass/groups/a1b2c3d4-5678-90ab-cdef-1b1b1EXAMPLE}
```

Output:

```
{
    "gatewayId": "a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE",
```

```
    "gatewayArn": "arn:aws:iotsitewise:us-west-2:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE"
}
```

For more information, see [Configuring a gateway](#) in the *AWS IoT SiteWise User Guide*.

- For API details, see [CreateGateway](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    /**
     * Creates a new IoT Sitewise gateway.
     *
     * @param gatewayName The name of the gateway to create.
     * @param myThing     The name of the core device thing to associate with the
 gateway.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the gateways ID. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<String> createGatewayAsync(String gatewayName,
 String myThing) {
        GreengrassV2 gg = GreengrassV2.builder()
            .coreDeviceThingName(myThing)
```

```
            .build();

        GatewayPlatform platform = GatewayPlatform.builder()
            .greengrassV2(gg)
            .build();

        Map<String, String> tag = new HashMap<>();
        tag.put("Environment", "Production");

        CreateGatewayRequest createGatewayRequest =
CreateGatewayRequest.builder()
            .gatewayName(gatewayName)
            .gatewayPlatform(platform)
            .tags(tag)
            .build();

        return getAsyncClient().createGateway(createGatewayRequest)
            .handle((response, exception) -> {
                if (exception != null) {
                    logger.error("Error creating the gateway.");
                    throw (CompletionException) exception;
                }
                logger.info("The ARN of the gateway is {}" ,
response.gatewayArn());
                return response.gatewayId();
            });
    }
```

- For API details, see [CreateGateway](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
```

```
  CreateGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Gateway.
 * @param {{  }}
 */
export const main = async ({ gatewayName }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateGatewayCommand({
        gatewayName: gatewayName, // The name to give the created Gateway.
      }),
    );
    console.log("Gateway created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Gateway.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [CreateGateway](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                           access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def create_gateway(self, gateway_name: str, my_thing: str) -> str:
        """
        Creates an AWS IoT SiteWise Gateway.

        :param gateway_name: The name of the gateway to create.
        :param my_thing: The core device thing name.
        :return: The ID of the created gateway.
        """
        try:
            response = self.iotsitewise_client.create_gateway(
                gatewayName=gateway_name,
                gatewayPlatform={
                    "greengrassV2": {"coreDeviceThingName": my_thing},
                },
                tags={"Environment": "Production"},
```

```
            )
            gateway_id = response["gatewayId"]
            return gateway_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Gateway %s already exists.", gateway_name)
            else:
                logger.error(
                    "Error creating gateway %s. Here's why %s",
                    gateway_name,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see CreateGateway in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreatePortal` with an AWS SDK or CLI

The following code examples show how to use `CreatePortal`.

CLI

AWS CLI

To create a portal

The following `create-portal` example creates a web portal for a wind farm company. You can create portals only in the same Region where you enabled AWS Single Sign-On.

```
aws iotsitewise create-portal \
    --portal-name WindFarmPortal \
    --portal-description "A portal that contains wind farm projects for Example
 Corp." \
    --portal-contact-email support@example.com \
    --role-arn arn:aws:iam::123456789012:role/service-role/
MySiteWiseMonitorServiceRole
```

Output:

```
{
    "portalId": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalArn": "arn:aws:iotsitewise:us-west-2:123456789012:portal/
a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalStartUrl": "https://a1b2c3d4-5678-90ab-cdef-
aaaaaEXAMPLE.app.iotsitewise.aws",
    "portalStatus": {
        "state": "CREATING"
    },
    "ssoApplicationId": "ins-a1b2c3d4-EXAMPLE"
}
```

For more information, see Getting started with AWS IoT SiteWise Monitor in the *AWS IoT SiteWise User Guide* and Enabling AWS SSO in the *AWS IoT SiteWise User Guide*..

- For API details, see CreatePortal in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Creates a new IoT SiteWise portal.
     *
     * @param portalName   the name of the portal to create.
     * @param iamRole      the IAM role ARN to use for the portal.
     * @param contactEmail the email address of the portal contact.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the portal ID. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
```

```
     *          If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *          it available to the calling code as a {@link CompletionException}.
 By calling
     *          {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<String> createPortalAsync(String portalName, String
 iamRole, String contactEmail) {
        CreatePortalRequest createPortalRequest = CreatePortalRequest.builder()
            .portalName(portalName)
            .portalDescription("This is my custom IoT SiteWise portal.")
            .portalContactEmail(contactEmail)
            .roleArn(iamRole)
            .build();

        return getAsyncClient().createPortal(createPortalRequest)
            .handle((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to create portal: {} ",
 exception.getCause().getMessage());
                    throw (CompletionException) exception;
                }
                return response.portalId();
            });
    }
```

- For API details, see [CreatePortal](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  CreatePortalCommand,
```

```
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Portal.
 * @param {{ portalName: string, portalContactEmail: string, roleArn: string }}
 */
export const main = async ({ portalName, portalContactEmail, roleArn }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreatePortalCommand({
        portalName: portalName, // The name to give the created Portal.
        portalContactEmail: portalContactEmail, // A valid contact email.
        roleArn: roleArn, // The ARN of a service role that allows the portal's
 users to access the portal's resources.
      }),
    );
    console.log("Portal created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Portal.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [CreatePortal](https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/iotsitewise/command/CreatePortalCommand) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                                   access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def create_portal(
        self, portal_name: str, iam_role_arn: str, portal_contact_email: str
    ) -> str:
        """
```

```
        Creates an AWS IoT SiteWise Portal.

        :param portal_name: The name of the portal to create.
        :param iam_role_arn: The ARN of an IAM role.
        :param portal_contact_email: The contact email of the portal.
        :return: The ID of the created portal.
        """
        try:
            response = self.iotsitewise_client.create_portal(
                portalName=portal_name,
                roleArn=iam_role_arn,
                portalContactEmail=portal_contact_email,
            )
            portal_id = response["portalId"]
            waiter = self.iotsitewise_client.get_waiter("portal_active")
            waiter.wait(portalId=portal_id, WaiterConfig={"MaxAttempts": 40})
            return portal_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Portal %s already exists.", portal_name)
            else:
                logger.error(
                    "Error creating portal %s. Here's why %s",
                    portal_name,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see CreatePortal in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAsset` with an AWS SDK or CLI

The following code examples show how to use `DeleteAsset`.

CLI

AWS CLI

To delete an asset

The following `delete-asset` example deletes a wind turbine asset.

```
aws iotsitewise delete-asset \
    --asset-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
```

Output:

```
{
    "assetStatus": {
        "state": "DELETING"
    }
}
```

For more information, see Deleting assets in the *AWS IoT SiteWise User Guide*.

- For API details, see DeleteAsset in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Deletes an asset.
     *
     * @param assetId the ID of the asset to be deleted.
     * @return a {@link CompletableFuture} that represents a {@link
 DeleteAssetResponse} result. The calling code can
     *         attach callbacks, then handle the result or exception by calling
 {@link CompletableFuture#join()} or
```

```
     *          {@link CompletableFuture#get()}.
     *          <p>
     *          If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *          it available to the calling code as a {@link CompletionException}.
 By calling
     *          {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DeleteAssetResponse> deleteAssetAsync(String
 assetId) {
        DeleteAssetRequest deleteAssetRequest = DeleteAssetRequest.builder()
            .assetId(assetId)
            .build();

        return getAsyncClient().deleteAsset(deleteAssetRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("An error occurred deleting asset with id: {}",
 assetId);
                }
            });
    }
```

- For API details, see DeleteAsset in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  DeleteAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";
```

```
/**
 * Delete an asset.
 * @param {{ assetId : string }}
 */
export const main = async ({ assetId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetCommand({
        assetId: assetId, // The model id to delete.
      }),
    );
    console.log("Asset deleted successfully.");
    return { assetDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [DeleteAsset](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""
```

```
    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                          access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def delete_asset(self, asset_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Asset.

        :param asset_id: The ID of the asset to delete.
        """
        try:
            self.iotsitewise_client.delete_asset(assetId=asset_id)
        except ClientError as err:
            logger.error(
                "Error deleting asset %s. Here's why %s",
                asset_id,
                err.response["Error"]["Message"],
            )
            raise
```

- For API details, see DeleteAsset in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteAssetModel` with an AWS SDK or CLI

The following code examples show how to use `DeleteAssetModel`.

CLI

AWS CLI

To delete an asset model

The following `delete-asset-model` example deletes a wind turbine asset model.

```
aws iotsitewise delete-asset-model \
    --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE
```

Output:

```
{
    "assetModelStatus": {
        "state": "DELETING"
    }
}
```

For more information, see Deleting asset models in the *AWS IoT SiteWise User Guide*.

- For API details, see DeleteAssetModel in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Deletes an Asset Model with the specified ID.
     *
     * @param assetModelId the ID of the Asset Model to delete.
     * @return a {@link CompletableFuture} that represents a {@link
DeleteAssetModelResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
By calling
     *         {@link CompletionException#getCause()}, the calling code can
access the original exception.
     */
    public CompletableFuture<DeleteAssetModelResponse>
deleteAssetModelAsync(String assetModelId) {
        DeleteAssetModelRequest deleteAssetModelRequest =
DeleteAssetModelRequest.builder()
            .assetModelId(assetModelId)
            .build();

        return getAsyncClient().deleteAssetModel(deleteAssetModelRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to delete asset model with ID:{}.",
exception.getMessage());
                }
            });
    }
```

- For API details, see DeleteAssetModel in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
import {
  DeleteAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetModelCommand({
        assetModelId: assetModelId, // The model id to delete.
      }),
    );
    console.log("Asset model deleted successfully.");
    return { assetModelDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset model.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [DeleteAssetModel](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                              access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def delete_asset_model(self, asset_model_id: str) -> None:
        """
```

```
        Deletes an AWS IoT SiteWise Asset Model.

        :param asset_model_id: The ID of the asset model to delete.
        """
        try:

 self.iotsitewise_client.delete_asset_model(assetModelId=asset_model_id)
        except ClientError as err:
            logger.error(
                "Error deleting asset model %s. Here's why %s",
                asset_model_id,
                err.response["Error"]["Message"],
            )
            raise
```

- For API details, see [DeleteAssetModel](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using this service with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteGateway` with an AWS SDK or CLI

The following code examples show how to use `DeleteGateway`.

CLI

AWS CLI

To delete a gateway

The following `delete-gateway` example deletes a gateway.

```
aws iotsitewise delete-gateway \
    --gateway-id a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE
```

This command produces no output.

For more information, see [Ingesting data using a gateway](#) in the *AWS IoT SiteWise User Guide*.

- For API details, see [DeleteGateway](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    /**
     * Deletes the specified gateway.
     *
     * @param gatewayId the ID of the gateway to delete.
     * @return a {@link CompletableFuture} that represents a {@link
 DeleteGatewayResponse} result.. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DeleteGatewayResponse> deleteGatewayAsync(String
 gatewayId) {
        DeleteGatewayRequest deleteGatewayRequest =
 DeleteGatewayRequest.builder()
            .gatewayId(gatewayId)
            .build();

        return getAsyncClient().deleteGateway(deleteGatewayRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to delete gateway: {}",
 exception.getCause().getMessage());
                }
```

```
            });
    }
```

- For API details, see [DeleteGateway](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  DeleteGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway deleted successfully.");
    return { gatewayDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the
 Gateway Id.`,
      );
```

```
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [DeleteGateway](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                                   access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
```

```
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def delete_gateway(self, gateway_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Gateway.

        :param gateway_id: The ID of the gateway to delete.
        """
        try:
            self.iotsitewise_client.delete_gateway(gatewayId=gateway_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Gateway %s does not exist.", gateway_id)
            else:
                logger.error(
                    "Error deleting gateway %s. Here's why %s",
                    gateway_id,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see DeleteGateway in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeletePortal` with an AWS SDK or CLI

The following code examples show how to use `DeletePortal`.

CLI

AWS CLI

To delete a portal

The following `delete-portal` example deletes a web portal for a wind farm company.

```
aws iotsitewise delete-portal \
    --portal-id a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE
```

Output:

```
{
    "portalStatus": {
        "state": "DELETING"
    }
}
```

For more information, see Deleting a portal in the *AWS IoT SiteWise User Guide*.

- For API details, see DeletePortal in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Deletes a portal.
     *
     * @param portalId the ID of the portal to be deleted.
     * @return a {@link CompletableFuture} that represents a {@link
 DeletePortalResponse}. The calling code can attach
     *         callbacks, then handle the result or exception by calling {@link
 CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
```

```
     */
    public CompletableFuture<DeletePortalResponse> deletePortalAsync(String
 portalId) {
        DeletePortalRequest deletePortalRequest = DeletePortalRequest.builder()
            .portalId(portalId)
            .build();

        return getAsyncClient().deletePortal(deletePortalRequest)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("Failed to delete portal with ID: {}. Error:
 {}", portalId, exception.getCause().getMessage());
                }
            });
    }
```

- For API details, see DeletePortal in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  DeletePortalCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
 * @param {{ portalId : string }}
 */
export const main = async ({ portalId }) => {
  const client = new IoTSiteWiseClient({});
  try {
```

```
    await client.send(
      new DeletePortalCommand({
        portalId: portalId, // The id of the portal.
      }),
    );
    console.log("Portal deleted successfully.");
    return { portalDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the portal. Please check
 the portal id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see DeletePortal in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                                access to AWS IoT SiteWise services.
```

```
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def delete_portal(self, portal_id: str) -> None:
        """
        Deletes an AWS IoT SiteWise Portal.

        :param portal_id: The ID of the portal to delete.
        """
        try:
            self.iotsitewise_client.delete_portal(portalId=portal_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Portal %s does not exist.", portal_id)
            else:
                logger.error(
                    "Error deleting portal %s. Here's why %s",
                    portal_id,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see [DeletePortal](#) in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use DescribeAssetModel with an AWS SDK or CLI

The following code examples show how to use `DescribeAssetModel`.

CLI

AWS CLI

To describe an asset model

The following `describe-asset-model` example describes a wind farm asset model.

```
aws iotsitewise describe-asset-model \
    --asset-model-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE
```

Output:

```
{
    "assetModelId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
    "assetModelArn": "arn:aws:iotsitewise:us-west-2:123456789012:asset-model/
a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
    "assetModelName": "Wind Farm Model",
    "assetModelDescription": "Represents a wind farm that comprises many wind
 turbines",
    "assetModelProperties": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-99999EXAMPLE",
            "name": "Total Generated Power",
            "dataType": "DOUBLE",
            "unit": "kW",
            "type": {
                "metric": {
                    "expression": "sum(power)",
                    "variables": [
                        {
                            "name": "power",
                            "value": {
                                "propertyId": "a1b2c3d4-5678-90ab-
cdef-66666EXAMPLE",
```

```
                                "hierarchyId": "a1b2c3d4-5678-90ab-
cdef-77777EXAMPLE"
                            }
                        }
                    ],
                    "window": {
                        "tumbling": {
                            "interval": "1h"
                        }
                    }
                }
            }
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-88888EXAMPLE",
            "name": "Region",
            "dataType": "STRING",
            "type": {
                "attribute": {
                    "defaultValue": " "
                }
            }
        }
    ],
    "assetModelHierarchies": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-77777EXAMPLE",
            "name": "Wind Turbines",
            "childAssetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"
        }
    ],
    "assetModelCreationDate": 1575671284.0,
    "assetModelLastUpdateDate": 1575671988.0,
    "assetModelStatus": {
        "state": "ACTIVE"
    }
}
```

For more information, see Describing a specific asset model in the *AWS IoT SiteWise User Guide*.

- For API details, see DescribeAssetModel in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/iotsitewise#code-examples).

```
    /**
     * Retrieves the property IDs associated with a specific asset model.
     *
     * @param assetModelId the ID of the asset model that defines the properties.
     * @return a {@link CompletableFuture} that represents a {@link Map} result
 that associates the property name to the
     *         propert ID. The calling code can attach callbacks, then handle the
 result or exception by calling
     *         {@link CompletableFuture#join()} or {@link
 CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<Map<String, String>> getPropertyIds(String
 assetModelId) {
        ListAssetModelPropertiesRequest modelPropertiesRequest =
 ListAssetModelPropertiesRequest.builder().assetModelId(assetModelId).build();
        return getAsyncClient().listAssetModelProperties(modelPropertiesRequest)
            .handle((response, throwable) -> {
                if (response != null) {
                    return response.assetModelPropertySummaries().stream()
                        .collect(Collectors
                            .toMap(AssetModelPropertySummary::name,
 AssetModelPropertySummary::id));
                } else {
                    logger.error("Error occurred while fetching property IDs:
 {}.", throwable.getCause().getMessage());
```

```
                    throw (CompletionException) throwable;
                }
            });
    }
```

- For API details, see [DescribeAssetModel](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  DescribeAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { assetModelDescription } = await client.send(
      new DescribeAssetModelCommand({
        assetModelId: assetModelId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset model information retrieved successfully.");
    return { assetModelDescription: assetModelDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
```

```
        `${caught.message}. The asset model could not be found. Please check the
 asset model id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see DescribeAssetModel in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeGateway` with an AWS SDK or CLI

The following code examples show how to use `DescribeGateway`.

CLI

AWS CLI

To describe a gateway

The following `describe-gateway` example describes a gateway.

```
aws iotsitewise describe-gateway \
    --gateway-id a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE
```

Output:

```
{
    "gatewayId": "a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE",
    "gatewayName": "ExampleCorpGateway",
    "gatewayArn": "arn:aws:iotsitewise:us-west-2:123456789012:gateway/
a1b2c3d4-5678-90ab-cdef-1a1a1EXAMPLE",
    "gatewayPlatform": {
        "greengrass": {
            "groupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/a1b2c3d4-5678-90ab-cdef-1b1b1EXAMPLE"
```

```
        }
    },
    "gatewayCapabilitySummaries": [
        {
            "capabilityNamespace": "iotsitewise:opcuacollector:1",
            "capabilitySyncStatus": "IN_SYNC"
        }
    ],
    "creationDate": 1588369971.457,
    "lastUpdateDate": 1588369971.457
}
```

For more information, see Ingesting data using a gateway in the *AWS IoT SiteWise User Guide*.

- For API details, see DescribeGateway in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Describes the specified gateway.
     *
     * @param gatewayId the ID of the gateway to describe.
     * @return a {@link CompletableFuture} that represents a {@link
 DescribeGatewayResponse} result. The calling code
     *         can attach callbacks, then handle the result or exception by
 calling {@link CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
```

```
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<DescribeGatewayResponse> describeGatewayAsync(String
 gatewayId) {
        DescribeGatewayRequest request = DescribeGatewayRequest.builder()
            .gatewayId(gatewayId)
            .build();

        return getAsyncClient().describeGateway(request)
            .whenComplete((response, exception) -> {
                if (exception != null) {
                    logger.error("An error occurred during the describeGateway
 method: {}", exception.getCause().getMessage());
                }
            });
    }
```

- For API details, see [DescribeGateway](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  DescribeGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
```

```
  const client = new IoTSiteWiseClient({});
  try {
    const { gatewayDescription } = await client.send(
      new DescribeGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway information retrieved successfully.");
    return { gatewayDescription: gatewayDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the
 Gateway Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [DescribeGateway](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.
```

```
        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                           access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
 AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def describe_gateway(self, gateway_id: str) -> Dict[str, Any]:
        """
        Describes an AWS IoT SiteWise Gateway.

        :param gateway_id: The ID of the gateway to describe.
        :return: A dictionary containing information about the gateway.
        """
        try:
            response =
 self.iotsitewise_client.describe_gateway(gatewayId=gateway_id)
            return response
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Gateway %s does not exist.", gateway_id)
            else:
                logger.error(
                    "Error describing gateway %s. Here's why %s",
                    gateway_id,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see DescribeGateway in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribePortal` with an AWS SDK or CLI

The following code examples show how to use `DescribePortal`.

CLI

AWS CLI

To describe a portal

The following `describe-portal` example describes a web portal for a wind farm company.

```
aws iotsitewise describe-portal \
    --portal-id a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE
```

Output:

```
{
    "portalId": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalArn": "arn:aws:iotsitewise:us-west-2:123456789012:portal/
a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
    "portalName": "WindFarmPortal",
    "portalDescription": "A portal that contains wind farm projects for Example
 Corp.",
    "portalClientId": "E-a1b2c3d4e5f6_a1b2c3d4e5f6EXAMPLE",
    "portalStartUrl": "https://a1b2c3d4-5678-90ab-cdef-
aaaaaEXAMPLE.app.iotsitewise.aws",
    "portalContactEmail": "support@example.com",
    "portalStatus": {
        "state": "ACTIVE"
    },
    "portalCreationDate": "2020-02-04T23:01:52.90248068Z",
    "portalLastUpdateDate": "2020-02-04T23:01:52.90248078Z",
```

```
    "roleArn": "arn:aws:iam::123456789012:role/MySiteWiseMonitorServiceRole"
}
```

For more information, see Administering your portals in the *AWS IoT SiteWise User Guide*.

- For API details, see DescribePortal in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Retrieves a portal's description.
     *
     * @param portalId the ID of the portal to describe.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the portal's start URL
     *         (see: {@link DescribePortalResponse#portalStartUrl()}). The
 calling code can attach callbacks, then handle the
     *         result or exception by calling {@link CompletableFuture#join()} or
 {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<String> describePortalAsync(String portalId) {
        DescribePortalRequest request = DescribePortalRequest.builder()
            .portalId(portalId)
            .build();

        return getAsyncClient().describePortal(request)
            .handle((response, exception) -> {
```

```
                if (exception != null) {
                    logger.error("An exception occurred retrieving the portal
 description: {}", exception.getCause().getMessage());
                    throw (CompletionException) exception;
                }
                return response.portalStartUrl();
            });
    }
```

- For API details, see DescribePortal in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  DescribePortalCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe a portal.
 * @param {{ portalId: string }}
 */
export const main = async ({ portalId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new DescribePortalCommand({
        portalId: portalId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Portal information retrieved successfully.");
    return result;
```

```
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Portal could not be found. Please check the
 Portal Id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see DescribePortal in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                            access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
```

```
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def create_gateway(self, gateway_name: str, my_thing: str) -> str:
        """
        Creates an AWS IoT SiteWise Gateway.

        :param gateway_name: The name of the gateway to create.
        :param my_thing: The core device thing name.
        :return: The ID of the created gateway.
        """
        try:
            response = self.iotsitewise_client.create_gateway(
                gatewayName=gateway_name,
                gatewayPlatform={
                    "greengrassV2": {"coreDeviceThingName": my_thing},
                },
                tags={"Environment": "Production"},
            )
            gateway_id = response["gatewayId"]
            return gateway_id
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.error("Gateway %s already exists.", gateway_name)
            else:
                logger.error(
                    "Error creating gateway %s. Here's why %s",
                    gateway_name,
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see [DescribePortal] in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetAssetPropertyValue` with an AWS SDK or CLI

The following code examples show how to use `GetAssetPropertyValue`.

CLI

AWS CLI

To retrieve an asset property's current value

The following `get-asset-property-value` example retrieves a wind turbine asset's current total power.

```
aws iotsitewise get-asset-property-value \
    --asset-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
    --property-id a1b2c3d4-5678-90ab-cdef-66666EXAMPLE
```

Output:

```
{
    "propertyValue": {
        "value": {
            "doubleValue": 6890.8677520453875
        },
        "timestamp": {
            "timeInSeconds": 1580853000,
            "offsetInNanos": 0
        },
        "quality": "GOOD"
    }
}
```

For more information, see Querying current asset property values in the *AWS IoT SiteWise User Guide*.

- For API details, see GetAssetPropertyValue in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/iotsitewise#code-examples).

```
    /**
     * Fetches the value of an asset property.
     *
     * @param propId  the ID of the asset property to fetch.
     * @param assetId the ID of the asset to fetch the property value for.
     * @return a {@link CompletableFuture} that represents a {@link Double}
 result. The calling code can attach
     *         callbacks, then handle the result or exception by calling {@link
 CompletableFuture#join()} or
     *         {@link CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
    public CompletableFuture<Double> getAssetPropValueAsync(String propId, String
 assetId) {
        GetAssetPropertyValueRequest assetPropertyValueRequest =
 GetAssetPropertyValueRequest.builder()
                .propertyId(propId)
                .assetId(assetId)
                .build();

        return getAsyncClient().getAssetPropertyValue(assetPropertyValueRequest)
                .handle((response, exception) -> {
                    if (exception != null) {
                        logger.error("Error occurred while fetching property
 value: {}.", exception.getCause().getMessage());
                        throw (CompletionException) exception;
```

```
                }
                return response.propertyValue().value().doubleValue();
            });
    }
```

- For API details, see [GetAssetPropertyValue](#) in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import {
  GetAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset property value.
 * @param {{ entryId : string }}
 */
export const main = async ({ entryId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new GetAssetPropertyValueCommand({
        entryId: entryId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset property information retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
```

```
        `${caught.message}. The asset property entry could not be found. Please
 check the entry id.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see [GetAssetPropertyValue](#) in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.

        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
 provides low-level
                              access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
 batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
 client.
```

```
        :return: An instance of IoTSitewiseWrapper initialized with the default
AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def get_asset_property_value(
        self, asset_id: str, property_id: str
    ) -> Dict[str, Any]:
        """
        Gets the value of an AWS IoT SiteWise Asset Property.

        :param asset_id: The ID of the asset.
        :param property_id: The ID of the property.
        :return: A dictionary containing the value of the property.
        """
        try:
            response = self.iotsitewise_client.get_asset_property_value(
                assetId=asset_id, propertyId=property_id
            )
            return response["propertyValue"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Asset %s or property %s does not exist.", asset_id,
property_id
                )
            else:
                logger.error(
                    "Error getting asset property value. Here's why %s",
                    err.response["Error"]["Message"],
                )
            raise
```

- For API details, see GetAssetPropertyValue in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use ListAssetModels with an AWS SDK or CLI

The following code examples show how to use ListAssetModels.

CLI

AWS CLI

To list all asset models

The following list-asset-models example lists all asset models that are defined in your AWS account in the current Region.

```
aws iotsitewise list-asset-models
```

Output:

```
{
    "assetModelSummaries": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "arn": "arn:aws:iotsitewise:us-west-2:123456789012:asset-model/
a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "name": "Wind Farm Model",
            "description": "Represents a wind farm that comprises many wind
 turbines",
            "creationDate": 1575671284.0,
            "lastUpdateDate": 1575671988.0,
            "status": {
                "state": "ACTIVE"
            }
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "arn": "arn:aws:iotsitewise:us-west-2:123456789012:asset-model/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
            "name": "Wind Turbine Model",
            "description": "Represents a wind turbine manufactured by Example
 Corp",
```

```
            "creationDate": 1575671207.0,
            "lastUpdateDate": 1575686273.0,
            "status": {
                "state": "ACTIVE"
            }
        }
    ]
}
```

For more information, see Listing all asset models in the *AWS IoT SiteWise User Guide*.

- For API details, see ListAssetModels in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Retrieves the asset model ID for the given asset model name.
     *
     * @param assetModelName the name of the asset model for the ID.
     * @return a {@link CompletableFuture} that represents a {@link String}
 result of the asset model ID or null if the
     *         asset model cannot be found. The calling code can attach
 callbacks, then handle the result or exception
     *         by calling {@link CompletableFuture#join()} or {@link
 CompletableFuture#get()}.
     *         <p>
     *         If any completion stage in this method throws an exception, the
 method logs the exception cause and keeps
     *         it available to the calling code as a {@link CompletionException}.
 By calling
     *         {@link CompletionException#getCause()}, the calling code can
 access the original exception.
     */
```

```
    public CompletableFuture<String> getAssetModelIdAsync(String assetModelName)
 {
        ListAssetModelsRequest listAssetModelsRequest =
 ListAssetModelsRequest.builder().build();
        return getAsyncClient().listAssetModels(listAssetModelsRequest)
                .handle((listAssetModelsResponse, exception) -> {
                    if (exception != null) {
                        logger.error("Failed to retrieve Asset Model ID: {}",
 exception.getCause().getMessage());
                        throw (CompletionException) exception;
                    }
                    for (AssetModelSummary assetModelSummary :
 listAssetModelsResponse.assetModelSummaries()) {
                        if (assetModelSummary.name().equals(assetModelName)) {
                            return assetModelSummary.id();
                        }
                    }
                    return null;
                });
    }
```

- For API details, see ListAssetModels in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  ListAssetModelsCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
```

```
 * @param {{ assetModelTypes : array }}
 */
export const main = async ({ assetModelTypes = [] }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new ListAssetModelsCommand({
        assetModelTypes: assetModelTypes, // The model types to list
      }),
    );
    console.log("Asset model types retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem listing the asset model types.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- For API details, see ListAssetModels in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class IoTSitewiseWrapper:
    """Encapsulates AWS IoT SiteWise actions using the client interface."""

    def __init__(self, iotsitewise_client: client) -> None:
        """
        Initializes the IoTSitewiseWrapper with an AWS IoT SiteWise client.
```

```
        :param iotsitewise_client: A Boto3 AWS IoT SiteWise client. This client
provides low-level
                           access to AWS IoT SiteWise services.
        """
        self.iotsitewise_client = iotsitewise_client
        self.entry_id = 0 # Incremented to generate unique entry IDs for
batch_put_asset_property_value.

    @classmethod
    def from_client(cls) -> "IoTSitewiseWrapper":
        """
        Creates an IoTSitewiseWrapper instance with a default AWS IoT SiteWise
client.

        :return: An instance of IoTSitewiseWrapper initialized with the default
AWS IoT SiteWise client.
        """
        iotsitewise_client = boto3.client("iotsitewise")
        return cls(iotsitewise_client)


    def list_asset_models(self) -> List[Dict[str, Any]]:
        """
        Lists all AWS IoT SiteWise Asset Models.

        :return: A list of dictionaries containing information about each asset
model.

        """
        try:
            asset_models = []
            paginator =
self.iotsitewise_client.get_paginator("list_asset_models")
            pages = paginator.paginate()
            for page in pages:
                asset_models.extend(page["assetModelSummaries"])
            return asset_models
        except ClientError as err:
            logger.error(
                "Error listing asset models. Here's why %s",
                err.response["Error"]["Message"],
            )
            raise
```

- For API details, see ListAssetModels in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

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

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. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT SiteWise or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- AWS IoT SiteWise Assistant Business Service improvement

Internetwork traffic privacy for AWS IoT SiteWise

Connections between AWS IoT SiteWise and on-premises applications, such as SiteWise Edge gateways, are secured over Transport Layer Security (TLS) connections. For more information, see Data encryption in transit for AWS IoT SiteWise.

AWS IoT SiteWise doesn't support connections between Availability Zones within an AWS Region or connections between AWS accounts.

You can configure IAM Identity Center in only one Region at a time. SiteWise Monitor connects to the Region that you configured for IAM Identity Center. This means that you use one Region for IAM Identity Center access, but you can create portals in any Region.

AWS IoT SiteWise Assistant Business Service improvement

AWS IoT SiteWise Assistant does not use customer data for service improvement or for improving underlying LLMs.

Data encryption in AWS IoT SiteWise

Data encryption refers to protecting data while in-transit (as it travels to and from AWS IoT SiteWise, and between SiteWise Edge 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.

> **Note**
>
> AWS IoT SiteWise edge processing exposes APIs that are hosted within SiteWise Edge 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 in AWS IoT SiteWise

AWS IoT SiteWise stores your data in the AWS Cloud and on AWS IoT SiteWise Edge 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 keyto 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",
      "Resource": "*"
    }
  ]
}
```

The encryption context for your created grant uses your `aws:iotsitewise:subscriberId` and account ID.

Data at rest on SiteWise Edge 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 SiteWise Edge gateway loses connection to the internet

SiteWise Edge 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*.

Data encryption in transit for AWS IoT SiteWise

AWS IoT SiteWise uses encryption in transit to secure the data transmitted between your devices, gateways, and the AWS Cloud. Communication with AWS IoT SiteWise is encrypted using HTTPS and TLS 1.2, ensuring that your data remains confidential and protected from unauthorized access or interception.

There are three modes of communication where data is in transit:

- Over the internet – Communication between local devices (including SiteWise Edge gateways) and AWS IoT SiteWise is encrypted.

- Over the local network – Communication between OpsHub for SiteWise application and SiteWise Edge gateways is always encrypted. Communication between the SiteWise monitor application running within your browser and SiteWise Edge gateways is always encrypted. Communication between SiteWise Edge gateways and OPC UA sources can be encrypted.
- Between components on SiteWise Edge gateways – Communication between AWS IoT Greengrass components on SiteWise Edge gateways isn't encrypted.

Topics

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. SiteWise Edge 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

SiteWise Edge 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 SiteWise Edge gateways and sources is signed but not encrypted. If you choose a *sign and encrypt* message security mode, the data in transit between SiteWise Edge gateways and sources is signed and encrypted. For more information about configuring sources, see Add data sources to your AWS IoT SiteWise Edge gateway.

The communication between the edge console application and SiteWise Edge gateways is always encrypted by TLS. The SiteWise Edge connector on the SiteWise Edge 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 SiteWise Edge gateway to the edge console for AWS IoT SiteWise application before you connect the application to the SiteWise Edge gateway. This ensures that the edge console for AWS IoT SiteWise application is able to verify that it has connected to your trusted SiteWise Edge 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 SiteWise Edge 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 “Manage gateways”.

Data in transit between local components on SiteWise Edge

SiteWise Edge 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 in AWS IoT SiteWise

AWS IoT SiteWise cloud 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 in the *AWS Key Management Service Developer Guide*.

5. Choose **Save** to update your settings.

SiteWise Edge gateway key management

SiteWise Edge 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 in the *AWS IoT Greengrass Version 1 Developer Guide*.

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 for AWS IoT SiteWise security

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 Troubleshoot AWS IoT SiteWise identity and access.

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 service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT SiteWise, see How AWS IoT SiteWise works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS IoT SiteWise. To view example AWS IoT SiteWise identity-based policies that you can use in IAM, see AWS IoT SiteWise identity-based policy examples.

Authenticate with identities in AWS IoT SiteWise

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users 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. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). 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 Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets] in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions –** An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access –** You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access –** Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS) –** When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an

[IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

How AWS IoT SiteWise works with IAM

Before you use AWS Identity and Access Management (IAM) to manage access to AWS IoT SiteWise, you should understand what IAM features are available to use with AWS IoT SiteWise.

IAM feature	Suppo by AWS IoT SiteWi
Identity-based policies with resource-level permissions	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
Resource-based policies	No
Access control lists (ACLs)	No
Tags-based authorization (ABAC)	Yes
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service-linked roles	Yes
Service roles	Yes

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 IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Use 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 IAM Identity Center or IAM credentials.

> **Important**
>
> 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 Use service roles for AWS IoT SiteWise Monitor.

Forward access sessions (FAS) for AWS IoT SiteWise

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service-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 AWS 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 Use service-linked roles for AWS IoT SiteWise.

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 AWS 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 Use service roles for AWS IoT SiteWise Monitor.

You must have required permissions before you can create AWS IoT Events alarm models in AWS IoT SiteWise. For more information, see Set up permissions for event alarms in AWS IoT SiteWise.

Choose 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 Use service roles for AWS IoT SiteWise Monitor.

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 `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging AWS IoT SiteWise resources, see Tag your AWS IoT SiteWise resources.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [View AWS IoT SiteWise assets based on tags].

AWS IoT SiteWise identity-based policies

IAM policies let you control who can do what in AWS IoT SiteWise. You can decide what actions are allowed or not and set specific conditions for these actions. For example, you can make rules about who can see or change information in AWS IoT SiteWise. 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*.

Policy actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

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, that action returns an error if permissions aren't granted. With `BatchPutAssetPropertyValue`, you can send multiple data entries to different assets and asset properties in a single API request. 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

Before you ingest data to a data stream, do the following:

- Authorize the `time-series` resource if you use a property alias to identify the data stream.
- Authorize the `asset` resource if you use an asset ID to identify the asset that contains the associated asset property.

Policy resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

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 Identify AWS resources with Amazon Resource Names (ARNs).

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 data streams that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:iotsitewise:region:123456789012:time-series/*"
```

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 Resource types 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.

Policy 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

Condition key	Description	Types
`iotsitewise:isAssociatedWithAssetProperty`	Whether data streams are associated with an asset property. Use this condition key to define permissions based on the existence of an associated asset property for data streams. Example value: `true`	String
`iotsitewise:assetHierarchyPath`	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`	String
`iotsitewise:propertyId`	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.	String

Condition key	Description	Types
	Example value: `a1b2c3d4-5678-90ab-cdef-33333EXAMPLE`	
`iotsitewise:childAssetId`	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`	String
`iotsitewise:iam`	The ARN of an IAM identity when listing access policies. Use this condition key to define access policy permissions for an IAM identity. Example value: `arn:aws:iam::123456789012:user/JohnDoe`	String, Null
`iotsitewise:propertyAlias`	The alias that identifies an asset property or data stream. Use this condition key to define permissions based on the alias.	String

Condition key	Description	Types
`iotsitewise:user`	The ID of an IAM Identity Center user when listing access policies. Use this condition key to define access policy permissions for an IAM Identity Center user. Example value: `a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE`	String, Null
`iotsitewise:group`	The ID of an IAM Identity Center group when listing access policies. Use this condition key to define access policy permissions for an IAM Identity Center group. Example value: `a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE`	String, Null
`iotsitewise:portal`	The ID of a portal in an access policy. Use this condition key to define access policy permissions based on a portal. Example value: `a1b2c3d4-5678-90ab-cdef-77777EXAMPLE`	String, Null

Condition key	Description	Types
`iotsitewise:project`	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. Example value: `a1b2c3d4-5678-90ab-cdef-888 88EXAMPLE`	String, Null

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](#).

AWS IoT SiteWise identity-based policy examples

By default, entities (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 Command Line Interface (AWS CLI), or AWS API. To adjust permissions, an AWS Identity and Access Management (IAM) administrator must do the following:

1. Create IAM policies that grant users and roles permission to perform specific API operations on resources they need.
2. Attach those policies to the 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
- Use the AWS IoT SiteWise console
- Allow users to view their own permissions
- Allow users to ingest data to assets in one hierarchy
- View AWS IoT SiteWise assets based on tags

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA]() in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM]() in the *IAM User Guide*.

Use the AWS IoT SiteWise console

To access the AWS IoT SiteWise console, you need a basic set of permissions. These permissions let you see and manage details about the AWS IoT SiteWise resources in your AWS account.

If you make a policy that's too restrictive, the console might not work as expected for users or roles (entities) with that policy. To ensure that those entities can still use the AWS IoT SiteWise console, attach the [AWSIoTSiteWiseConsoleFullAccess]() managed policy to them or define equivalent permissions for those entities. For more information, see [Adding permissions to a user]() in the *IAM User Guide*.

If entities are only using the AWS Command Line Interface (CLI) or the AWS IoT SiteWise API, and not the console, they don't need these minimum permissions. In that case, just give them access to the specific actions they need for their API tasks.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow users to ingest data to assets in one hierarchy

In this example, you want to grant a 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/*"
              ]
            }
          }
        }
      ]
    }
```

View AWS IoT SiteWise assets based on tags

Use conditions in your identity-based policy to control access to AWS IoT SiteWise resources based on tags. This example shows how to create a policy that allows asset viewing. However, permission is granted only if the asset tag `Owner` has the value of that user's user name. This policy also grants permission to complete this action on the console.

Attach this policy to the 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, Richard is denied access. The condition tag key names are not case-sensitive. So, `Owner` matches both `Owner` and `owner`. For more information, see [IAM JSON Policy Elements: Condition] in the *IAM User Guide*.

Manage access using policies in AWS IoT SiteWise

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies] in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed 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 Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) 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*.

AWS managed policies for AWS IoT SiteWise

Simplify adding permissions to users, groups, and roles using AWS managed policies rather than to writing policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team precise permissions. For a faster setup, consider using our AWS managed policies for common use cases. Find AWS managed policies in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services take care of updating and maintaining AWS managed policies, meaning you cannot modify these policies' permissions. Occasionally, AWS IoT SiteWise may add permissions to accommodate new features, impacting all identities with the policy attached. Such updates are common with the introduction of new services or features. However, permissions are never removed, ensuring your setups remain intact.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list with descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AWSIoTSiteWiseReadOnlyAccess

Use the `AWSIoTSiteWiseReadOnlyAccess` AWS managed policy to allow read-only access to AWS IoT SiteWise.

You can attach the `AWSIoTSiteWiseReadOnlyAccess` policy to your IAM identities.

Service-level permissions

This policy provides read-only access to AWS IoT SiteWise. No other service permissions are included in this policy.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:Describe*",
```

```
                "iotsitewise:List*",
                "iotsitewise:BatchGet*",
                "iotsitewise:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AWSServiceRoleForIoTSiteWise

The `AWSServiceRoleForIoTSiteWise` role uses the `AWSServiceRoleForIoTSiteWise` policy with the following permissions. This policy:

- Allows AWS IoT SiteWise to deploy SiteWise Edge gateways (which run on `AWS IoT Greengrass`).
- Allows AWS IoT SiteWise to perform logging.
- Allows AWS IoT SiteWise to run a metadata search query, against the AWS IoT TwinMaker database.

If you are using AWS IoT SiteWise with a singe user account,the `AWSServiceRoleForIoTSiteWise` role creates the `AWSServiceRoleForIoTSiteWise` policy in your IAM account, and attaches it to the `AWSServiceRoleForIoTSiteWise` [Service-linked roles for AWS IoT SiteWise](#).

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
    "greengrass:GetCoreDefinitionVersion",
    "greengrass:GetGroup",
```

```
    "greengrass:GetGroupVersion"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AllowSiteWiseAccessLogGroup",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:DescribeLogGroups"
   ],
   "Resource": "arn:aws:logs:*:*:log-group:/aws/iotsitewise*"
  },
  {
   "Sid": "AllowSiteWiseAccessLog",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:DescribeLogStreams",
    "logs:PutLogEvents"
   ],
   "Resource": "arn:aws:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
  },
  {
   "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
   "Effect": "Allow",
   "Action": [
    "iottwinmaker:GetWorkspace",
    "iottwinmaker:ExecuteQuery"
   ],
   "Resource": "arn:aws:iottwinmaker:*:*:workspace/*",
   "Condition": {
    "ForAnyValue:StringEquals": {
     "iottwinmaker:linkedServices": [
      "IOTSITEWISE"
     ]
    }
   }
  }
 ]
}
```

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
    "greengrass:GetCoreDefinitionVersion",
    "greengrass:GetGroup",
    "greengrass:GetGroupVersion"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AllowSiteWiseAccessLogGroup",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:DescribeLogGroups"
   ],
   "Resource": "arn:aws-us-gov:logs:*:*:log-group:/aws/iotsitewise*"
  },
  {
   "Sid": "AllowSiteWiseAccessLog",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:DescribeLogStreams",
    "logs:PutLogEvents"
   ],
   "Resource": "arn:aws-us-gov:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
  },
  {
   "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
   "Effect": "Allow",
   "Action": [
    "iottwinmaker:GetWorkspace",
    "iottwinmaker:ExecuteQuery"
```

```
   ],
   "Resource": "arn:aws-us-gov:iottwinmaker:*:*:workspace/*",
   "Condition": {
    "ForAnyValue:StringEquals": {
     "iottwinmaker:linkedServices": [
      "IOTSITEWISE"
     ]
    }
   }
  }
 ]
}
```

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
    "greengrass:GetCoreDefinitionVersion",
    "greengrass:GetGroup",
    "greengrass:GetGroupVersion"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AllowSiteWiseAccessLogGroup",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:DescribeLogGroups"
   ],
   "Resource": "arn:aws-cn:logs:*:*:log-group:/aws/iotsitewise*"
  },
  {
```

```
    "Sid": "AllowSiteWiseAccessLog",
    "Effect": "Allow",
    "Action": [
     "logs:CreateLogStream",
     "logs:DescribeLogStreams",
     "logs:PutLogEvents"
    ],
    "Resource": "arn:aws-cn:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
   },
   {
    "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
    "Effect": "Allow",
    "Action": [
     "iottwinmaker:GetWorkspace",
     "iottwinmaker:ExecuteQuery"
    ],
    "Resource": "arn:aws-cn:iottwinmaker:*:*:workspace/*",
    "Condition": {
     "ForAnyValue:StringEquals": {
      "iottwinmaker:linkedServices": [
       "IOTSITEWISE"
      ]
     }
    }
   }
  ]
}
```

AWS IoT SiteWise updates to AWS managed policies

You can view details about updates to AWS managed policies for AWS IoT SiteWise, beginning from when this service began tracking the changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS IoT SiteWise Document history page.

Change	Description	Date
AWSServiceRoleForIoTSiteWise – Update to an existing policy	AWS IoT SiteWise now can run a metadata search	November 6, 2023

Change	Description	Date
	query, against the AWS IoT TwinMaker database.	
AWSIoTSiteWiseReadOnlyAccess – Update to an existing policy	AWS IoT SiteWise added a new policy prefix, `BatchGet*`, that enables you to do batch read operations.	September 16, 2022
AWSIoTSiteWiseReadOnlyAccess – New policy	AWS IoT SiteWise added a new policy to grant read-only access to AWS IoT SiteWise.	November 24, 2021
AWS IoT SiteWise started tracking changes	AWS IoT SiteWise started tracking changes for its AWS managed policies.	November 24, 2021

Use 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.

Service-linked roles simplify the configuration of AWS IoT SiteWise by automatically including all 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.

Topics

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 SiteWise Edge gateways (which run on AWS IoT Greengrass) and perform logging.

The `AWSServiceRoleForIoTSiteWise` service-linked role uses the `AWSServiceRoleForIoTSiteWise` policy with the following permissions. This policy:

- Allows AWS IoT SiteWise to deploy SiteWise Edge gateways (which run on `AWS IoT Greengrass`).
- Allows AWS IoT SiteWise to perform logging.
- Allows AWS IoT SiteWise to run a metadata search query, against the AWS IoT TwinMaker database.

For more information on the allowed actions in `AWSServiceRoleForIoTSiteWise`, see [AWS managed policies for AWS IoT SiteWise].

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
```

```
      "greengrass:GetCoreDefinitionVersion",
      "greengrass:GetGroup",
      "greengrass:GetGroupVersion"
    ],
    "Resource": "*"
  },
  {
    "Sid": "AllowSiteWiseAccessLogGroup",
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:DescribeLogGroups"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/iotsitewise*"
  },
  {
    "Sid": "AllowSiteWiseAccessLog",
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogStream",
      "logs:DescribeLogStreams",
      "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
  },
  {
    "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
    "Effect": "Allow",
    "Action": [
      "iottwinmaker:GetWorkspace",
      "iottwinmaker:ExecuteQuery"
    ],
    "Resource": "arn:aws:iottwinmaker:*:*:workspace/*",
    "Condition": {
      "ForAnyValue:StringEquals": {
        "iottwinmaker:linkedServices": [
          "IOTSITEWISE"
        ]
      }
    }
  }
 ]
}
```

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
    "greengrass:GetCoreDefinitionVersion",
    "greengrass:GetGroup",
    "greengrass:GetGroupVersion"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AllowSiteWiseAccessLogGroup",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:DescribeLogGroups"
   ],
   "Resource": "arn:aws-us-gov:logs:*:*:log-group:/aws/iotsitewise*"
  },
  {
   "Sid": "AllowSiteWiseAccessLog",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogStream",
    "logs:DescribeLogStreams",
    "logs:PutLogEvents"
   ],
   "Resource": "arn:aws-us-gov:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
  },
  {
   "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
   "Effect": "Allow",
```

```
    "Action": [
     "iottwinmaker:GetWorkspace",
     "iottwinmaker:ExecuteQuery"
    ],
    "Resource": "arn:aws-us-gov:iottwinmaker:*:*:workspace/*",
    "Condition": {
     "ForAnyValue:StringEquals": {
      "iottwinmaker:linkedServices": [
       "IOTSITEWISE"
      ]
     }
    }
   }
  ]
}
```

JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AllowSiteWiseReadGreenGrass",
   "Effect": "Allow",
   "Action": [
    "greengrass:GetAssociatedRole",
    "greengrass:GetCoreDefinition",
    "greengrass:GetCoreDefinitionVersion",
    "greengrass:GetGroup",
    "greengrass:GetGroupVersion"
   ],
   "Resource": "*"
  },
  {
   "Sid": "AllowSiteWiseAccessLogGroup",
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:DescribeLogGroups"
   ],
```

```
      "Resource": "arn:aws-cn:logs:*:*:log-group:/aws/iotsitewise*"
    },
    {
      "Sid": "AllowSiteWiseAccessLog",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws-cn:logs:*:*:log-group:/aws/iotsitewise*:log-stream:*"
    },
    {
      "Sid": "AllowSiteWiseAccessSiteWiseManagedWorkspaceInTwinMaker",
      "Effect": "Allow",
      "Action": [
        "iottwinmaker:GetWorkspace",
        "iottwinmaker:ExecuteQuery"
      ],
      "Resource": "arn:aws-cn:iottwinmaker:*:*:workspace/*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "iottwinmaker:linkedServices": [
            "IOTSITEWISE"
          ]
        }
      }
    }
  ]
}
```

You can use the logs to monitor and troubleshoot your SiteWise Edge gateways. For more information, see Monitor SiteWise Edge gateway logs.

To allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role, first configure permissions. For more information, see Service-linked role permissions in the *IAM User Guide*.

Create a service-linked role for AWS IoT SiteWise

AWS IoT SiteWise requires a service-linked role to perform certain actions and to access resources on your behalf. A service-linked role is a unique type of AWS Identity and Access Management (IAM) role that is linked directly to AWS IoT SiteWise. By creating this role, you grant AWS IoT SiteWise the necessary permissions to access other AWS services and resources required for its operation, such as Amazon S3 for data storage or AWS IoT for device communication.

You don't need to manually create a service-linked role. When you perform the following operations in the AWS IoT SiteWise console, AWS IoT SiteWise creates the service-linked role for you.

- Create a Greengrass V1 gateway.
- Configure the logging option.
- Choosing the opt-in button in the execute query banner.

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`.

For more information, see [Create 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.

Update 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 [Update a service-linked role] in the *IAM User Guide*.

Delete a service-linked role for AWS IoT SiteWise

If a feature or service requiring a service-linked role is no longer in use, it's advisable to delete the associated role. This is to avoid having an inactive entity that isn't being 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 Change your logging level
2. Delete any active SiteWise Edge 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 Delete roles or instance profiles 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.

Use 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 Create 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 IAM Identity Center 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. This policy is maintained by AWS and defines the set of permissions that SiteWise Monitor uses to access your AWS IoT SiteWise and IAM Identity Center 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 IAM Identity Center 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 (Classic)

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 IAM Identity Center configuration.

The role trusts the following service to assume the role:

- `monitor.iotsitewise.amazonaws.com`

The role uses the following permissions policy, which 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",
            "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"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "Null": {
                    "aws:RequestTag/iotsitewisemonitor": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:UpdateAlarmModel",
                "iotevents:DeleteAlarmModel"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:ResourceTag/iotsitewisemonitor": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "iotevents.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

JSON

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

```
        "Action": [
          "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"
        ],
        "Resource": "*"
      }
    ]
}
```

For more information about the required permissions for alarms, see Set up permissions for event alarms in AWS IoT SiteWise.

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 Administer your SiteWise Monitor portals and CreateAccessPolicy.

Service role permissions for SiteWise Monitor (AI-aware)

When you create a portal, AWS IoT SiteWise lets you create a role whose name starts with **IoTSiteWisePortalRole**. This role allows federated SiteWise Monitor users to access your portal configuration, assets, asset data, and IAM Identity Center configuration.

> **Warning**
>
> **Project owner** and **Project viewer** roles are not supported for SiteWise Monitor (AI-aware).

The role trusts the following service to assume the role:

- `monitor.iotsitewise.amazonaws.com`

The role uses the following permissions policy, which starts with **IoTSiteWiseAIPortalAccessPolicy**, to allow SiteWise Monitor users to complete actions on resources in your account.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:CreateProject",
                "iotsitewise:DescribePortal",
                "iotsitewise:ListProjects",
                "iotsitewise:DescribeProject",
                "iotsitewise:UpdateProject",
                "iotsitewise:DeleteProject",
                "iotsitewise:CreateDashboard",
                "iotsitewise:DescribeDashboard",
                "iotsitewise:UpdateDashboard",
                "iotsitewise:DeleteDashboard",
                "iotsitewise:ListDashboards",
                "iotsitewise:ListAssets",
                "iotsitewise:DescribeAsset",
                "iotsitewise:ListAssociatedAssets",
                "iotsitewise:ListAssetProperties",
```

```
                "iotsitewise:DescribeAssetProperty",
                "iotsitewise:GetAssetPropertyValue",
                "iotsitewise:GetAssetPropertyValueHistory",
                "iotsitewise:GetAssetPropertyAggregates",
                "iotsitewise:GetInterpolatedAssetPropertyValues",
                "iotsitewise:BatchGetAssetPropertyAggregates",
                "iotsitewise:BatchGetAssetPropertyValue",
                "iotsitewise:BatchGetAssetPropertyValueHistory",
                "iotsitewise:ListAssetRelationships",
                "iotsitewise:DescribeAssetModel",
                "iotsitewise:ListAssetModels",
                "iotsitewise:DescribeAssetCompositeModel",
                "iotsitewise:DescribeAssetModelCompositeModel",
                "iotsitewise:ListAssetModelProperties",
                "iotsitewise:ExecuteQuery",
                "iotsitewise:ListTimeSeries",
                "iotsitewise:DescribeTimeSeries",
                "iotsitewise:InvokeAssistant",
                "iotsitewise:DescribeDataset",
                "iotsitewise:ListDatasets",
                "iotevents:DescribeAlarmModel",
                "iotevents:ListTagsForResource",
                "iottwinmaker:ListWorkspaces",
                "iottwinmaker:ExecuteQuery",
                "iottwinmaker:GetWorkspace",
                "identitystore:DescribeUser"
            ],
            "Resource": "*"
        }
    ]
}
```

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.

Manage the SiteWise Monitor service role (console)

The AWS IoT SiteWise console facilitates the management of the SiteWise Monitor service role for portals. Upon creating a portal, the console checks for existing roles suitable for attachment. If none are available, the console can create and configure a service role for you. For more information, see Create a portal in SiteWise Monitor.

Topics

Find 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**.

Create 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 Create a portal in SiteWise Monitor.

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.
4. Under **Portal details**, choose **Edit**.
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**.

Create 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**.

Change 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.
4. Under **Portal details**, choose **Edit**.
5. Under **Permissions**, choose **Use an existing role**.
6. Choose an existing role to attach to this portal.
7. Choose **Save**.

Manage the SiteWise Monitor service role (CLI)

You can use the AWS CLI for the following portal service role management tasks:

Topics

- Find a portal's service role (CLI)
- Create the SiteWise Monitor service role (CLI)

Find 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.

```
aws iotsitewise list-portals
```

The operation returns a response that contains your portal summaries in the following format.

```
{
  "portalSummaries": [
    {
      "id": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
      "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-aaaaaEXAMPLE.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.

Create 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

```
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

```
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\"}]}"
```

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*.

3. a. For the SiteWise Monitor (Classic) – Attach the `AWSIoTSiteWiseMonitorPortalAccess` policy to the role, or attach a policy that defines equivalent permissions.

```
aws iam attach-role-policy --role-name MySiteWiseMonitorPortalRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSIoTSiteWiseMonitorPortalAccess
```

   b. For the SiteWise Monitor (AI-aware) – Attach the `IoTSiteWiseAIPortalAccessPolicy` policy to the role, or attach a policy that defines equivalent permissions. For example, create a policy with portal access permissions. The following example creates a policy named `MySiteWiseMonitorPortalAccess`.

```
aws iam create-policy \
    --policy-name MySiteWiseMonitorPortalAccess \
```

```
    --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:CreateProject",
                "iotsitewise:DescribePortal",
                "iotsitewise:ListProjects",
                "iotsitewise:DescribeProject",
                "iotsitewise:UpdateProject",
                "iotsitewise:DeleteProject",
                "iotsitewise:CreateDashboard",
                "iotsitewise:DescribeDashboard",
                "iotsitewise:UpdateDashboard",
                "iotsitewise:DeleteDashboard",
                "iotsitewise:ListDashboards",
                "iotsitewise:ListAssets",
                "iotsitewise:DescribeAsset",
                "iotsitewise:ListAssociatedAssets",
                "iotsitewise:ListAssetProperties",
                "iotsitewise:DescribeAssetProperty",
                "iotsitewise:GetAssetPropertyValue",
                "iotsitewise:GetAssetPropertyValueHistory",
                "iotsitewise:GetAssetPropertyAggregates",
                "iotsitewise:GetInterpolatedAssetPropertyValues",
                "iotsitewise:BatchGetAssetPropertyAggregates",
                "iotsitewise:BatchGetAssetPropertyValue",
                "iotsitewise:BatchGetAssetPropertyValueHistory",
                "iotsitewise:ListAssetRelationships",
                "iotsitewise:DescribeAssetModel",
                "iotsitewise:ListAssetModels",
                "iotsitewise:DescribeAssetCompositeModel",
                "iotsitewise:DescribeAssetModelCompositeModel",
                "iotsitewise:ListAssetModelProperties",
                "iotsitewise:ExecuteQuery",
                "iotsitewise:ListTimeSeries",
                "iotsitewise:DescribeTimeSeries",
                "iotsitewise:InvokeAssistant",
                "iotsitewise:DescribeDataset",
                "iotsitewise:ListDatasets",
                "iotevents:DescribeAlarmModel",
                "iotevents:ListTagsForResource",
                "iottwinmaker:ListWorkspaces",
```

```
                "iottwinmaker:ExecuteQuery",
                "iottwinmaker:GetWorkspace",
                "identitystore:DescribeUser"
            ],
            "Resource": "*"
        }
    ]
}'
```

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.

```
{
    "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_sn6tbqHVzLWVEXAMPLE",
    "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
```

SiteWise Monitor updates to AWSIoTSiteWiseMonitorServiceRole

You can view details about updates to **AWSIoTSiteWiseMonitorServiceRole** for SiteWise Monitor, beginning from when this service began tracking the changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS IoT SiteWise Document history page.

Change	Description	Date
AWSIoTSiteWiseMonitorPortalAccess – Updated policy	AWS IoT SiteWise updated the AWSIoTSiteWiseMonitorPortalAccess managed policy for the alarms feature.	May 27, 2021
AWS IoT SiteWise started tracking changes	AWS IoT SiteWise started tracking changes for its service role.	December 15, 2020

Set up permissions for event alarms in AWS IoT SiteWise

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`, `iotsitewise:ListAssetModels` – 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:ListAssetModels",
                "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.

(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 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 with the following policy:

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"
```

```
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:RequestTag/iotsitewisemonitor": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotevents:UpdateAlarmModel",
                "iotevents:DeleteAlarmModel"
            ],
            "Resource": "*",
            "Condition": {
                "Null": {
                    "aws:ResourceTag/iotsitewisemonitor": "false"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "iotevents.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Cross-service confused deputy prevention in AWS IoT SiteWise

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it shouldn't otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS IoT SiteWise gives another service to the resource. If the `aws:SourceArn` value doesn't contain the account ID, such as an Amazon S3 bucket Amazon Resource Name (ARN), you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

- Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the AWS IoT SiteWise customer resource that is associated with the `sts:AssumeRole` request.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you're specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename:*:123456789012:*`.

Example – Confused Deputy Prevention

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWS IoT SiteWise to prevent the confused deputy problem.

JSON

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

```
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "iotsitewise.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:iotsitewise:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Troubleshoot 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 AWS Identity and Access Management (IAM).

Topics

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, your policies must be updated 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 that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

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.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS IoT 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.
- Ten security golden rules for Industrial IoT solutions – This blog post introduces ten golden rules that help secure your industrial control systems (ICS), industrial Internet of Things (IIoT), and cloud environments.

- Security Best Practices for Manufacturing OT – This whitepaper describes security best practices to design, deploy, and architect these on-premises hybrid manufacturing workloads for the AWS Cloud.

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, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

AWS IoT SiteWise is fully managed and uses highly available and durable AWS services, such as Amazon S3 and Amazon EC2. To ensure availability in the event of an availability zone disruption, AWS IoT SiteWise operates across multiple availability zones.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, AWS IoT SiteWise offers several features to help support your data resiliency and backup needs:

- You can publish property value updates to AWS IoT Core through MQTT messages, then configure rules to act upon that data. With this feature, you can back up data in other AWS services such as Amazon S3 and Amazon DynamoDB. For more information, see Interact with other AWS services and Export data to Amazon S3 with asset property notifications.
- You can use the AWS IoT SiteWise `Get*` APIs to retrieve and backup historical asset property data. For more information, see Query historical asset property values in AWS IoT SiteWise.
- You can use the AWS IoT SiteWise `Describe*` APIs to retrieve the definitions for your resources, such as assets and models. You can backup these definitions and later use them to recreate your resources. For more information, see the *AWS IoT SiteWise API Reference*.

Infrastructure security in AWS IoT SiteWise

As a managed service, AWS IoT SiteWise is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud

Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access AWS IoT SiteWise through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). 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.

SiteWise Edge 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 usually have limited processing power, memory, and storage, they can't always support encryption and other security measures. 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. 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 What is AWS IoT Device Defender in the *AWS IoT Device Defender Developer Guide*.

If you use SiteWise Edge gateways to ingest data to the service, it's your responsibility to configure and maintain your SiteWise Edge gateway's environment. This responsibility includes upgrading to the latest versions of the SiteWise Edge 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 Manage SiteWise Edge gateways.

VPC endpoints for AWS IoT SiteWise

An *interface VPC endpoint* establishes a private connection between your virtual private cloud (VPC) and AWS IoT SiteWise. AWS PrivateLink powers interface endpoints, enabling private access to AWS IoT SiteWise API operations. AWS IoT SiteWise supports both IPv4 and IPv6 (dual-stack) through its interface endpoints. You can bypass the need for an internet gateway, NAT device, VPN connection, or AWS Direct Connect. 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.

Before you set up an interface VPC endpoint for AWS IoT SiteWise, review the Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

API operations for VPC endpoints in AWS IoT SiteWise

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 the following:

  - BatchGetAssetPropertyValue
  - BatchGetAssetPropertyValueHistory
  - BatchPutAssetPropertyValue
  - GetAssetPropertyAggregates
  - GetAssetPropertyValue

  - [GetAssetPropertyValueHistory]
  - [GetInterpolatedAssetPropertyValues]
- For the **control plane** API operations that you use to manage asset models, assets, SiteWise Edge gateways, tags, and account configurations, use the following endpoint. Replace *region* with your AWS Region.

  ```
  api.iotsitewise.region.amazonaws.com
  ```

  The supported control plane API operations include the following:

  - AssociateAssets
  - CreateAsset
  - CreateAssetModel
  - DeleteAsset
  - DeleteAssetModel
  - DeleteDashboard
  - DescribeAsset
  - DescribeAssetModel
  - DescribeAssetProperty
  - DescribeDashboard
  - DescribeLoggingOptions
  - DisassociateAssets
  - ListAssetModels
  - ListAssetRelationships
  - ListAssets
  - ListAssociatedAssets
  - PutLoggingOptions
  - UpdateAsset
  - UpdateAssetModel
  - UpdateAssetProperty
  - CreateGateway
  - DeleteGateway

- DescribeDefaultEncryptionConfiguration
- DescribeGateway
- DescribeGatewayCapabilityConfiguration
- DescribeStorageConfiguration
- ListGateways
- ListTagsForResource
- UpdateGateway
- UpdateGatewayCapabilityConfiguration
- PutDefaultEncryptionConfiguration
- PutStorageConfiguration
- TagResource
- UntagResource

Note

The interface VPC endpoint for the **control plane** API operations currently doesn't support making calls to the following SiteWise Monitor API operations:

- BatchAssociateProjectAssets
- BatchDisassociateProjectAssets
- CreateAccessPolicy
- CreateDashboard
- CreatePortal
- CreateProject
- DeleteAccessPolicy
- DeletePortal
- DeleteProject
- DescribeAccessPolicy
- DescribePortal
- DescribeProject
- ListAccessPolicies
- ListDashboards

- ListPortals
- ListProjects
- ListProjectAssets
- UpdateAccessPolicy
- UpdateDashboard
- UpdatePortal
- UpdateProject

Create an interface VPC endpoint for AWS IoT SiteWise

To create a VPC endpoint for the AWS IoT SiteWise service, use either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink 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
```

Access 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:

1. 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.

   ```
   vpc-endpoint-id.data.iotsitewise.region.vpce.amazonaws.com
   ```

   - For the **control plane** API operations, use the following endpoint url. Replace *vpc-endpoint-id* and *region* with your VPC endpoint ID and Region.

   ```
   vpc-endpoint-id.api.iotsitewise.region.vpce.amazonaws.com
   ```

2. Disable host prefix injection. The AWS CLI and AWS SDKs prepend the service endpoint with various host prefixes when you call each API operation. This feature causes the AWS CLI and AWS SDKs to produce URLs that are not valid for AWS IoT SiteWise when you specify a VPC endpoint.

   Important

   You can't disable host prefix injection in the AWS CLI or the AWS Tools for PowerShell. This means that if you disable private DNS, then you can't use these tools to access AWS IoT SiteWise through the VPC endpoint. Enable private DNS to use the AWS CLI or the AWS Tools for PowerShell to access AWS IoT SiteWise through the endpoint.

   For more information about how to disable host prefix injection in the AWS SDKs, see the following documentation sections for each SDK:

   - AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Go v2
- AWS SDK for Java
- AWS SDK for Java 2.x
- AWS SDK for JavaScript
- AWS SDK for .NET
- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

For more information, see Access an AWS service using an interface VPC endpoint in the *AWS PrivateLink Guide*.

Create 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 Control access to VPC endpoints using endpoint policies 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 user *iotsitewiseadmin* in AWS account *123456789012* on the specified asset.

```
{
    "Statement": [
        {
            "Action": [
                "iotsitewise:CreateAsset",
```

```
                "iotsitewise:ListGateways",
                "iotsitewise:ListTagsForResource"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iotsitewise:us-west-2:123456789012:asset/
a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
            "Principal": {
                "AWS": [
                    "123456789012:user/iotsitewiseadmin"
                ]
            }
        }
    ]
}
```

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 SiteWise Edge gateway to connect to your OPC UA servers, add server authentication secrets to your SiteWise Edge gateway. For more information, see Configure data source authentication for SiteWise Edge.

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 SiteWise Edge gateway. This helps secure your industrial data as it moves from your OPC UA servers to the SiteWise Edge gateway. For more information, see Data in transit over the local network and Set up an OPC UA source in SiteWise Edge.

Keep your components up to date

If you use SiteWise Edge gateways to ingest data to the service, it's your responsibility to configure and maintain your SiteWise Edge 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 SiteWise Edge gateway. It's strongly recommended that you turn on disk or file-system encryption for the system running your SiteWise Edge gateway.

For information on how to upgrade components in the AWS IoT SiteWise console, see Change the version of SiteWise Edge gateway component packs.

Encrypt your SiteWise Edge gateway's file system

Encrypt and secure your SiteWise Edge gateway, so your industrial data is secure as it moves through the SiteWise Edge gateway. If your SiteWise Edge gateway has a hardware security module, you can configure AWS IoT Greengrass to secure your SiteWise Edge gateway. For more information, see Hardware security integration in the *AWS IoT Greengrass Version 1 Developer Guide*. Otherwise, consult the documentation for your operating system to learn how to encrypt and secure your file system.

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.

Securing data on Siemens Industrial Edge Management

The device data you choose to share with AWS IoT SiteWise Edge is determined in your Siemens IEM Databus configuration topics. By electing topics to share with SiteWise Edge, you are sharing topic-level data with AWS IoT SiteWise. The Siemens Industrial Edge Marketplace is an independent marketplace, separate from AWS. To protect your shared data, the SiteWise Edge application will not run unless you utilize Siemens Secured Storage. For more information, see Secure Storage, in Siemens documentation.

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 Use service roles for AWS IoT SiteWise Monitor.
- 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.

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 SiteWise Edge 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 SiteWise Edge 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 SiteWise Edge 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 Industrial IoT solutions

Log and monitor 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. Collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a certain threshold. 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* monitors, stores, and accesses your log files from SiteWise Edge 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. Then CloudTrail 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

Monitor with Amazon CloudWatch Logs

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 [Create a service-linked role for AWS IoT SiteWise](#).

You must have a resource policy that allows AWS IoT SiteWise to put log events into CloudWatch streams. To create and update a resource policy for CloudWatch Logs, run the following command. Replace *logging-policy-name* with the name of the policy to create.

```
aws logs put-resource-policy --policy-name logging-policy-name --policy-
document "{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Sid\":
 \"IoTSiteWiseToCloudWatchLogs\", \"Effect\": \"Allow\", \"Principal\": { \"Service\":
 [ \"iotsitewise.amazonaws.com\" ] }, \"Action\":\"logs:PutLogEvents\", \"Resource\":
 \"*\" } ] }"
```

CloudWatch Logs also supports aws:SourceArn and aws:SourceAccount condition context keys. These condition context keys are optional.

To create or update a resource policy that allows AWS IoT SiteWise to only put logs associated with the specified AWS IoT SiteWise resource into CloudWatch streams, run the command and do the following:

- Replace *logging-policy-name* with the name of the policy to create.
- Replace *source-ARN* with the ARN of your AWS IoT SiteWise resource, such as an asset model or asset. To find the ARN for each AWS IoT SiteWise resource type, see Resource types defined by AWS IoT SiteWise in the *Service Authorization Reference*.
- Replace *account-ID* with the AWS account ID associated with the specified AWS IoT SiteWise resource.

```
aws logs put-resource-policy --policy-name logging-policy-name --policy-
document "{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Sid\":
 \"IoTSiteWiseToCloudWatchLogs\", \"Effect\": \"Allow\", \"Principal\": { \"Service
\": [ \"iotsitewise.amazonaws.com\" ] }, \"Action\":\"logs:PutLogEvents\", \"Resource
\": \"*\", \"Condition\":{\"StringLike\":{\"aws:SourceArn\":[\"source-ARN\"],
\"aws:SourceAccount\":[\"account-ID\"]}}}]}"
```

By default, AWS IoT SiteWise doesn't log information to CloudWatch Logs. To activate logging, choose a logging level other than **Disabled** (`OFF`). AWS IoT SiteWise supports the following logging levels:

- `OFF` – Logging is turned off.

- `ERROR` – Errors are logged.
- `INFO` – Errors and informational messages are logged.

You can configure SiteWise Edge gateways to log information to CloudWatch Logs through AWS IoT Greengrass. For more information, see Monitor SiteWise Edge gateway logs.

You can also configure AWS IoT Core to log information to CloudWatch Logs if you are troubleshooting an AWS IoT SiteWise rule action. For more information, see Troubleshoot an AWS IoT SiteWise rule action.

Contents

Manage logging in AWS IoT SiteWise

Use the AWS IoT SiteWise console or AWS CLI for the following logging configuration tasks.

Find 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 activated, the current logging level appears under **Level of verbosity**.

AWS 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"
  }
}
```

Change your logging level

Use the following procedure to change your logging level in the AWS IoT SiteWise console or using AWS CLI.

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 activate.
5. Choose **Save**.

AWS 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 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"
  }
}
```

Monitor SiteWise Edge gateway logs

You can configure your AWS IoT SiteWise Edge gateway to log information to Amazon CloudWatch Logs or the local file system.

Topics

- Use Amazon CloudWatch Logs
- Use service logs in AWS IoT SiteWise

Use Amazon CloudWatch Logs

You can configure your SiteWise Edge gateway to send logs to CloudWatch Logs. For more information, see Enable logging for CloudWatch Logs in the *AWS IoT Greengrass Version 2 Developer Guide*.

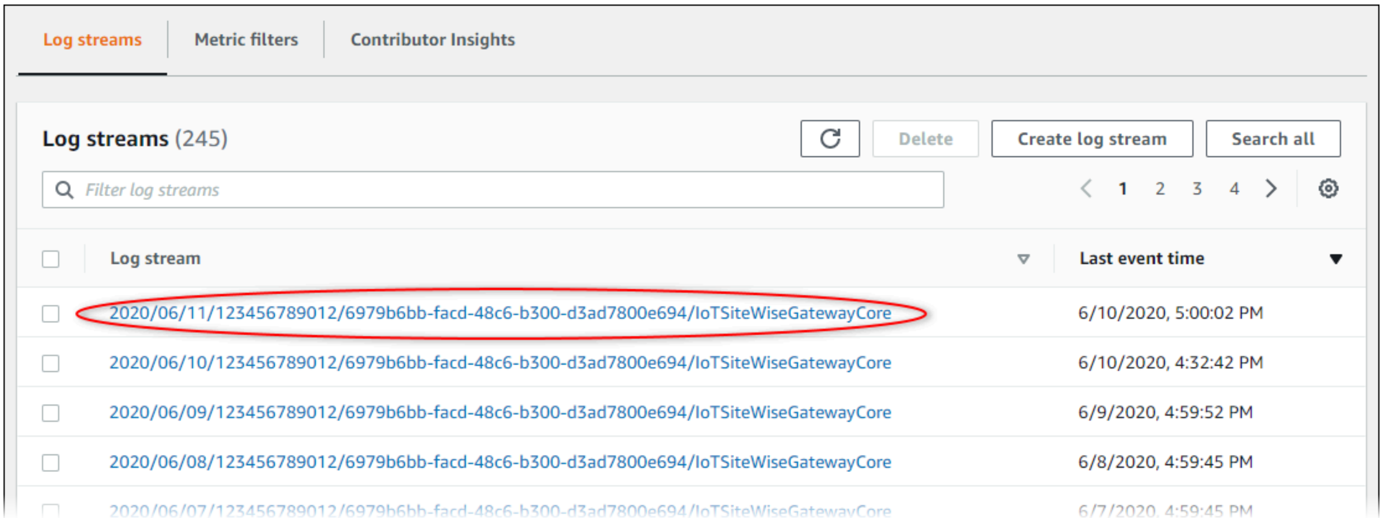
To configure and access CloudWatch Logs (Console)

1. Navigate to the CloudWatch console.
2. In the navigation pane, choose **Log groups**.
3. You can find the AWS IoT SiteWise component logs in the following log groups:
   - `/aws/greengrass/UserComponent/`*`region`*`/aws.iot.SiteWiseEdgeCollectorOpcua` – The logs for the SiteWise Edge gateway's component that collects data from the SiteWise Edge gateway's OPC UA sources.

- `/aws/greengrass/UserComponent/region/aws.iot.SiteWiseEdgePublisher` – The logs for the SiteWise Edge gateway's component that publishes OPC UA data streams to AWS IoT SiteWise.

Choose the log group for the function to debug.

4. 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.



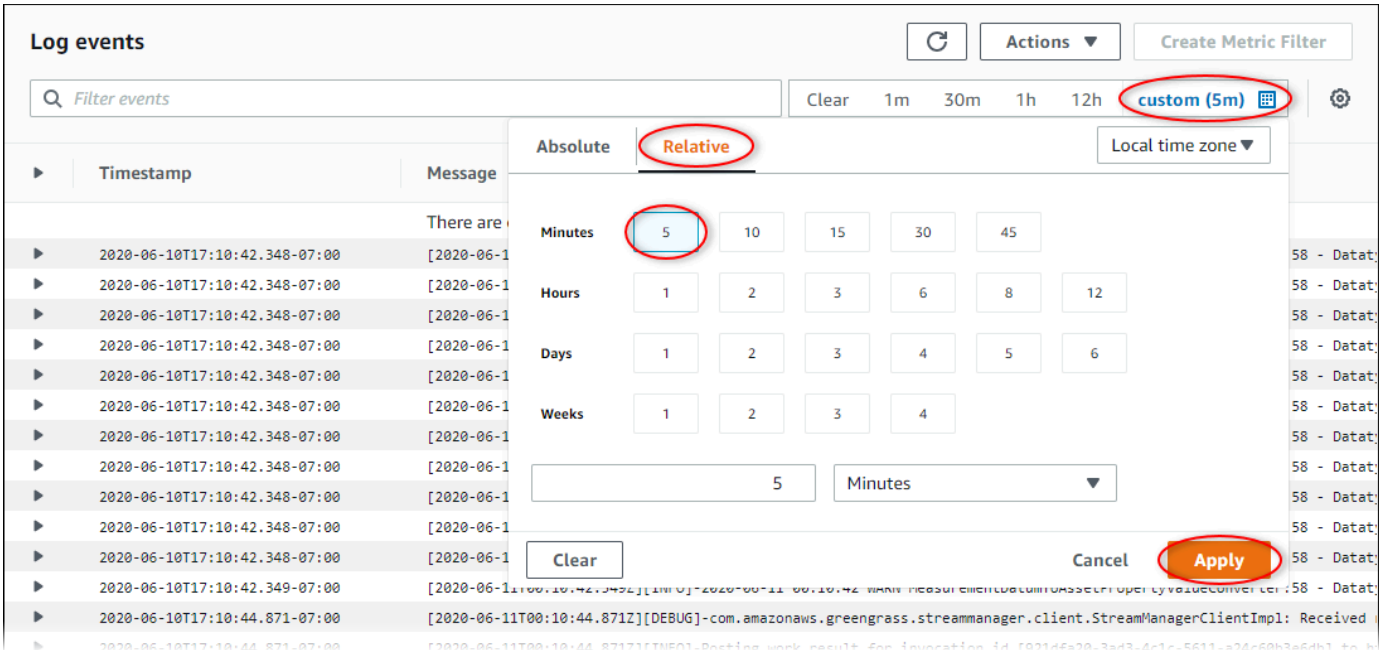
5. To show logs from the last 5 minutes, do the following:

   a. Choose **custom** in the upper-right corner.

   b. Choose **Relative**.

   c. Choose **5** minutes.

   d. Choose **Apply**.



6. (Optional) To see fewer logs, you can choose **1m** from the upper-right corner.
7. Scroll to the bottom of the log entries to show the most recent logs.

Use service logs in AWS IoT SiteWise

SiteWise Edge gateway devices include service log files to help debug issues. The following sections will help you find and utilize the service log files for the AWS IoT SiteWise OPC UA Collector and AWS IoT SiteWise Publisher components.

AWS IoT SiteWise OPC UA Collector service log file

The AWS IoT SiteWise OPC UA Collector component uses the following log file.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

To view this component's logs

- Run the following command on the core device to view this component's log file in real time. Replace */greengrass/v2* or *C:\greengrass\v2* with the path to the AWS IoT Greengrass root folder.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

AWS IoT SiteWise Publisher service log file

The AWS IoT SiteWise Publisher component uses the following log file.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

To view this component's logs

- Run the following command on the core device to view this component's log file in real time. Replace */greengrass/v2* or *C:\greengrass\v2* with the path to the AWS IoT Greengrass root folder.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
Wait
```

Monitor 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.

Topics

AWS IoT Greengrass Version 2 gateway metrics

AWS IoT SiteWise publishes gateway metrics for Classic streams, V2 gateways and MQTT-enabled, V3 gateways. Unless otherwise indicated, each metric is applicable to both self-hosted gateway versions. All SiteWise Edge gateway metrics are published on a one minute interval.

SiteWise Edge gateway metrics

Metric	Description
`Gateway.AvailableMemory`	The available memory of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.AvailableDiskSpace`	The available disk space of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.CloudConnectivity`	The cloud connectivity status of a SiteWise Edge gateway. Unit: None Dimension: GatewayId
`Gateway.CpuUsage`	The CPU usage of a SiteWise Edge gateway. Unit: Percentage Dimension: None
`Gateway.TotalDiskSpace`	The total disk space of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.TotalMemory`	The total memory of a SiteWise Edge gateway. Unit: Bytes

Metric	Description
	Dimension: None
`Gateway.UsedDiskSpace`	The used disk space of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.UsedMemory`	The used memory of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.UsedPercentageDiskSpace`	The used percentage of disk space of a SiteWise Edge gateway. Unit: Bytes Dimension: None
`Gateway.UsedPercentageMemory`	The used percentage memory of a SiteWise Edge gateway. Unit: Bytes Dimension: None

AWS IoT SiteWise publisher metrics

Metric	Description
`IoTSiteWisePublisher.ComponentBuildVersion`	This metric indicates the build version of the IoT SiteWise publisher component running on the gateway. A value of 1 signifies that the gateway is running a version of the publisher corresponding to the `ComponentBuildVersion` dimension.

Metric	Description
	Unit: 1 Dimensions: GatewayId, ComponentBuildVersion
`IoTSiteWisePublisher.DroppedCount`	The number of data points that are dropped by a SiteWise Edge gateway (`GatewayId`) and not published to the cloud, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.Heartbeat`	Generated every minute by the Publisher in the SiteWise Edge gateway. Unit: 1 (1 representing the Publisher is running and missing the data point representing the Publisher is not running.) Dimensions: GatewayId
`IoTSiteWisePublisher.IsConnectedToMqttBroker`	Generated every minute by the Publisher in the SiteWise Edge gateway. Unit: 1 (1 representing the Publisher is connected to a MQTT broker.) Dimensions: GatewayId
`IoTSiteWisePublisher.MessageCheckpointPersistenceErrorCount`	The metric indicates that the gateway has detected an issue with the checkpoint file used to track data processed by the publisher. A value of 1 signifies that a failure has occurred. Unit: None Dimensions: AccountId, GatewayId

Metric	Description
`IoTSiteWisePublisher.MqttMessageReceivedSuccessCount`	The number of messages successfully received by the Publisher from the MQTT broker, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.MqttReceivedSuccessBytes`	The number of bytes of message data successfully received by the Publisher from the MQTT broker, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker`	The number of topics subscribed to the MQTT broker by the Publisher, generated every minute. A multilevel wild card topic is counted as 1. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived`	The number of unique topics received by the Publisher from the MQTT broker, generated every minute. Unit: Count Dimensions: GatewayId

Metric	Description
`IoTSiteWisePublisher.PublishFailureCount`	The number of data points that a SiteWise Edge gateway (`GatewayId`) failed to publish, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.PublishRejectedCount`	The number of data points that a SiteWise Edge gateway (`GatewayId`) rejected from the cloud side, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.PublishSuccessCount`	The number of data points that a SiteWise Edge gateway (`GatewayId`) successfully published to the cloud, generated every minute. Unit: Count Dimensions: GatewayId
`IoTSiteWisePublisher.PublishToS3FailureCount`	The number of data points that a gateway (`GatewayId`) failed to publish to an Amazon S3 bucket. Unit: Count Dimensions: GatewayId

Metric	Description
`IoTSiteWisePublisher.PublishToS3SuccessCount`	The number of data points that a gateway (`GatewayId`) successfully published to an Amazon S3 bucket. Unit: Count Dimensions: GatewayId

OPC UA collector metrics

Metric	Description
`OpcUaCollector.ActiveDataStreamCount`	The number of data streams that a SiteWise Edge gateway (`gatewayId`) subscribed to for an OPC UA source (`sourceName`). Unit: Count Dimensions: GatewayId, SourceName, PropertyGroup
`OpcUaCollector.ComponentBuildVersion` (not available on Classic streams, V2 gateways)	This metric indicates the build version of the IoT SiteWise OPC UA collector component running on the gateway. A value of 1 signifies that the gateway is running a version of the collector corresponding to the `ComponentBuildVersion` dimension. Unit: 1 Dimensions: GatewayId, ComponentBuildVersion
`OpcUaCollector.ConversionErrors`	The number of data points that a SiteWise Edge gateway (`gatewayId`) received for an OPC UA source (`sourceName`) which resulted in conversion errors while sending the data to

Metric	Description
	AWS IoT SiteWise. These data points will not be ingested by OPC UA Collector. Unit: Count Dimensions: GatewayId, SourceName
`OpcUaCollector.Heartbeat`	Generated every minute for each OPC UA source (`sourceName`) connected to a SiteWise Edge gateway (`gatewayId`). Unit: Count (1 representing the source is connected and 0 representing the source is disconnected.) Dimensions: GatewayId, SourceName
`OpcUaCollector.IncomingValuesCount`	The number of data points that a SiteWise Edge gateway (`gatewayId`) received for an OPC UA source (`sourceName`), generated every minute. Unit: Count Dimensions: GatewayId, SourceName, PropertyGroup
`OpcUaCollector.IncomingValueErrors`	The number of data points that a SiteWise Edge gateway (`gatewayId`) receives from an OPC UA source (`sourceName`) that are not valid values. These data points are not ingested by the OPC UA Collector, generated every minute. Unit: Count Dimensions: GatewayId, SourceName, PropertyGroup

Metric	Description
`OpcUaCollector.IsConnectedToMqttBroker` (not available on Classic streams, V2 gateways)	Generated every minute by the IoT SiteWise OPC UA collector component in the SiteWise Edge gateway. Unit: 1 (1 representing the IoT SiteWise OPC UA collector component is connected to an MQTT broker) Dimensions: GatewayId
`OpcUaCollector.MqttMessagesDroppedCount` (not available on Classic streams, V2 gateways)	The number of MQTT messages dropped by the IoT SiteWise OPC UA collector component. Unit: Count Dimensions: GatewayId, SourceName
`OpcUaCollector.MqttMessagesPublishedBytes` (not available on Classic streams, V2 gateways)	The number of bytes of MQTT message data successfully published by the IoT SiteWise OPC UA collector component to the MQTT broker. Unit: Count Dimensions: GatewayId, SourceName
`OpcUaCollector.MqttMessagesPublishedCount` (not available on Classic streams, V2 gateways)	The number of MQTT messages successfully published by the IoT SiteWise OPC UA collector component to the MQTT broker. Unit: Count Dimensions: GatewayId, SourceName

Metric	Description
`OpcUaCollector.NullValueCount` (not available on Classic streams, V2 gateways)	The number of null values received by the IoT SiteWise OPC UA collector component from the OPC UA server. Unit: Count Dimensions: GatewayId, SourceName, PropertyGroup
`OpcUaCollector.NumberOfUniqueMqttTopicsPublished` (not available on Classic streams, V2 gateways)	The number of unique MQTT topics published by the IoT SiteWise OPC UA collector to the MQTT broker. Unit: Count Dimensions: GatewayId, SourceName

AWS IoT SiteWise processor metrics

Metric	Description
`Gateway.DataProcessor.IngestionThrottled` (not available on MQTT-enabled, V3 gateways)	The number of data points that were throttled , generated every minute. Unit: Count Dimensions: ThrottledAt
`Gateway.DataProcessor.MeasurementRejected` (not available on MQTT-enabled, V3 gateways)	The number of measurements that were rejected, generated every minute. Unit: Count Dimensions: Reason
`Gateway.DataProcessor.MessagesRemaining` (not available on MQTT-enabled, V3 gateways)	The number of messages remaining in a stream, generated every minute.

Metric	Description
	Unit: Count Dimensions: StreamName
`Gateway.DataProcessor.ProcessingError` (not available on MQTT-enabled, V3 gateways)	The number of processing errors, generated every minute. Unit: Count Dimensions: Reason

Log 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 activate 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 activated 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 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

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.

AWS IoT SiteWise data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, reading or writing to an Amazon S3 object). These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn’t log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the AWS IoT SiteWise resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. The table in this section shows the resource types available for AWS IoT SiteWise.

- To log data events using the CloudTrail console, create a trail or event data store to log data events, or update an existing trail or event data store to log data events.

  1. Choose **Data events** to log data events.

2. From the **Data event type** list, choose the resource type for which you want to log data events.

3. Choose the log selector template you want to use. You can log all data events for the resource type, log all `readOnly` events, log all `writeOnly` events, or create a custom log selector template to filter on the `readOnly`, `eventName`, and `resources.ARN` fields.

- To log data events using the AWS CLI, configure the `--advanced-event-selectors` parameter to set the `eventCategory` field equal to `Data` and the `resources.type` field equal to the resource type value (see table). You can add conditions to filter on the values of the `readOnly`, `eventName`, and `resources.ARN` fields.
  - To configure a trail to log data events, run the AWS CloudTrail put-event-selectors command. For more information, see Logging data events for trails with the AWS CLI.
  - To configure an event data store to log data events, run the AWS CloudTrail create-event-data-store command to create a new event data store to log data events, or run the AWS CloudTrail update-event-data-store command to update an existing event data store. For more information, see Logging data events for event data stores with the AWS CLI.

The following table lists the AWS IoT SiteWise resource types. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail*
AWS IoT SiteWise asset	`AWS::IoTSiteWise::Asset`	• BatchPutAssetPropertyValue • GetAssetPropertyValue • GetAssetPropertyValueHistory • GetAssetPropertyAggregates • GetInterpolatedAssetPropertyValues

Data event type (console)	resources.type value	Data APIs logged to CloudTrail*
		• BatchGetAssetPropertyValue • BatchGetAssetPropertyValueHistory • BatchGetAssetPropertyAggregates
AWS IoT SiteWise time series	`AWS::IoTSiteWise::TimeSeries`	• BatchPutAssetPropertyValue • GetAssetPropertyValue • GetAssetPropertyValueHistory • GetAssetPropertyAggregates • GetInterpolatedAssetPropertyValues • BatchGetAssetPropertyValue • BatchGetAssetPropertyValueHistory • BatchGetAssetPropertyAggregates
AWS IoT SiteWise Assistant	`AWS::SitewiseAssistant::Conversation`	• InvokeAssistant

> **Note**
>
> The resources.type logged in the Cloudtrail event depends on the identifier used in the API request. If an asset id is specified in the request then the Asset resources.type is logged, else the TimeSeries resources.type is logged.

*You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector.

AWS IoT SiteWise management events in CloudTrail

Logging management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

AWS IoT SiteWise logs all AWS IoT SiteWise control plane operations as management events. For a list of the AWS IoT SiteWise control plane operations that AWS IoT SiteWise logs to CloudTrail, see the AWS IoT SiteWise API Reference.

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.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "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"
      }
```

```
    },
    "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"
}
```

Tag your AWS IoT SiteWise resources

Tagging your AWS IoT SiteWise resources provides a powerful way to categorize, manage, and retrieve organizational assets efficiently. By assigning tags, which consist of key-value pairs, you can attach descriptive metadata to your resources. The metadata from tags can be used to streamline operations. For example, in a wind farm scenario, tags allow you to label turbines with specific attributes like location, capacity, and operational status, enabling quick identification and management within AWS IoT SiteWise.

Integrating tags with AWS Identity and Access Management (IAM) policies enhances security and operational control by defining conditional access rules. This means you can specify that only users with certain tags. For example, only those tagged with a certain role or department, can access or modify particular resources.

Use tags in AWS IoT SiteWise

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 is made up of a key and an optional value that you specify. For example, you can establish a series of tags for your asset models to track them according to the industrial processes they support. It's recommended to develop a tailored set of tag keys for each type of resource you manage. Using a consistent set of tag keys can makes it easier manage resources.

Tag 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 Getting started with Tag Editor in the *Tagging AWS Resources and Tag Editor User Guide*.

Tag with the AWS IoT SiteWise API

The AWS IoT SiteWise API also uses tags. 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

Resource	Create operation	Describe operation
Asset model or component model	CreateAssetModel	DescribeAssetModel
Asset	CreateAsset	DescribeAsset
SiteWise Edge gateway	CreateGateway	DescribeGateway
Portal	CreatePortal	DescribePortal
Project	CreateProject	DescribeProject
Dashboard	CreateDashboard	DescribeDashboard
Access policy	CreateAccessPolicy	DescribeAccessPolicy
Time series	BatchPutAssetPropertyValue	DescribeTimeSeries

For `BatchPutAssetPropertyValue`, you can configure your data sources to send industrial data to AWS IoT SiteWise before you create asset models and assets. AWS IoT SiteWise automatically creates data streams to receive streams of raw data from your equipment. For more information, see Managing data ingestion.

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.

Add or remove tags from a resource at any time. To update the value of an existing tag key, add a new tag with the same key and your desired new value to the resource. This action replaces the old value with the new one. While it's possible to assign an empty string as a tag value, you can't assign a null value.

Deleting a resource also removes any tags linked to it.

Use tags with IAM policies

Use resource tags in your IAM policies to control user access and permissions. For example, policies can allow users to only create resources that have a specific tag attached. 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 bypass 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.

`aws:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Allow or deny actions on resources with specific tags.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Require that a specific tag be used (or not used) when creating or modifying a taggable resource.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Require that a specific set of tag keys be used (or not used) when creating or modifying a taggable resource.

> **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

- View AWS IoT SiteWise assets based on tags

Troubleshooting AWS IoT SiteWise

Use the following information to troubleshoot issues with AWS IoT SiteWise.

Topics

Troubleshooting a SiteWise Edge gateway

Troubleshoot common AWS IoT SiteWise Edge gateway issues by exploring relevant topics.

You can also view CloudWatch metrics reported by your SiteWise Edge gateways to troubleshoot issues with connectivity or data streams. For more information, see Monitor AWS IoT SiteWise with Amazon CloudWatch metrics.

Topics

Configure and access SiteWise Edge gateway logs

Before you can view SiteWise Edge gateway logs, you must configure your SiteWise Edge 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 SiteWise Edge gateway's log files. For more information, see Use Amazon CloudWatch Logs.
- Use local file system logs if you want to use the command line or local software to view your SiteWise Edge gateway's log files. For more information, see Use service logs in AWS IoT SiteWise.

Troubleshooting SiteWise Edge gateway issues

Use the following information to troubleshoot SiteWise Edge gateway issues.

Issues

Unable to deploy packs to SiteWise Edge gateways

If the AWS IoT Greengrass nucleus component (`aws.greengrass.Nucleus`) is out of date, you might not be able to deploy packs to your SiteWise Edge gateway. You can use the AWS IoT Greengrass V2 console to upgrade the AWS IoT Greengrass nucleus component.

To upgrade the AWS IoT Greengrass nucleus component (console)

1. Navigate to the AWS IoT Greengrass console.
2. In the navigation pane, under **AWS IoT Greengrass**, choose **Deployments**.
3. In the **Deployments** list, select the deployment that you want to revise.
4. Choose **Revise**.
5. On the **Specify target** page, choose **Next**.

6. On the **Select components** page, under **Public components**, in the search box, enter **`aws.greengrass.Nucleus`**, and then select **aws.greengrass.Nucleus**.
7. Choose **Next**.
8. On the **Configure components** page, choose **Next**.
9. On the **Configure advanced settings** page, choose **Next**.
10. On the **Review** page, choose **Deploy**.

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 SiteWise Edge 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 SiteWise Edge 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 Manage data streams for AWS IoT SiteWise.

    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 SiteWise Edge gateway sends. For more information, see Use OPC UA node filters in SiteWise Edge.
- **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 SiteWise Edge 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 SiteWise

Edge gateway lost power or connection to the AWS Cloud, you might need to clear your SiteWise Edge 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.

No data shows in the dashboard

If there is no data shown in your dashboard, the **Publisher configuration** and the **Data Source** of the SiteWise Edge gateway may be out of sync. If they are out of sync, updating the name of the data source may expedite the sync from the cloud to the edge, fixing the Out of sync error.

To update the name of a data source

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the SiteWise Edge gateway connected to the dashboard.
4. Under **Data sources**, select **Edit**.
5. Select a new source **Name**, and select **Save** to confirm your change.
6. Verify your changes by confirming the the data source name has been updated in the **Data sources** table.

"Could not find or load main class" showing up in the aws.iot.SiteWiseEdgePublisher logs at /greengrass/v2/logs error

If you see this error, you may need to update the java version of your SiteWise Edge gateway.

- From a terminal, run the following command:

```
java -version
```

The version of java your SiteWise Edge gateway is running with will show up under `OpenJDK Runtime Environment`. You'll see a response like the following:

```
openjdk version "11.0.20" 2023-07-18 LTS
OpenJDK Runtime Environment Corretto011.0.20.8.1 (build 11.0.20+8-LTS
```

```
OpenJDK 64-Bit Server VM Corretto-11.0.20.8.1 (build 11.0.20+8-LTS, mixed node)
```

If you are running Java version 11.0.20.8.1 you must update the IoT SiteWise Publisher pack to version 2.4.1 or newer. Only java version 11.0.20.8.1 is affected, environments with other java versions can continue to use older versions of the IoT SiteWise Publisher component. For more information about updating a component pack, see Change the version of SiteWise Edge gateway component packs.

I see 'SESSION_TAKEN_OVER' or 'com.aws.greengrass.mqttclient.MqttClient: Failed to publish the message via Spooler and will retry.' in the logs

If you see a warning that includes `SESSION_TAKEN_OVER` or an error that includes `com.aws.greengrass.mqttclient.MqttClient: Failed to publish the message via Spooler and will retry.` in your logs at `/greengrass/v2/logs/greengrass.log`, you may be trying to use the same configuration file for multiple SiteWise Edge gateways on multiple devices. Each SiteWise Edge gateway needs a unique configuration file to connect to your AWS account.

I see 'com.aws.greengrass.deployment.IotJobsHelper: No deployment job found.' or 'Deployment result already reported.' in the logs

If you see `com.aws.greengrass.deployment.IotJobsHelper: No deployment job found.` or `Deployment result already reported.` in your logs at `/greengrass/v2/logs/greengrass.log`, you may be trying to reuse the same configuration file.

There are multiple solutions:

- If you want to reuse the configuration file, do the following:

  1. Navigate to the AWS IoT SiteWise console.
  2. In the navigation pane, choose **Edge gateways**.
  3. Choose the SiteWise Edge gateway you want to reuse.
  4. Choose the **Updates** tab.
  5. Select a different Publisher version and choose **Deploy**.

Follow the steps in Create a gateway for Siemens Industrial Edge to create a new configuration file.

I see a 'SYNC_FAILED' status when attempting to configure the timestamp setting in a property group on an OPC UA data source

When AWS IoT SiteWise updated the OPC UA collector component for AWS IoT Greengrass in version 2.5.0, we introduced a new timestamp configuration option. You can use the timestamp from either your device, or the timestamp from the server. Older versions of the OPC UA collector component do not support this option and fail to sync.

There are two ways to resolve a failing data source sync status. The recommended way is to upgrade the IoT SiteWise OPC UA collector component to version 2.5.0 or above. Alternatively, you can continue to use the older OPC UA collector component version, if you set the timestamp to `Source`. To learn how to upgrade the IoT SiteWise OPC UA collector component, see Update the version of an AWS IoT SiteWise component. We recommend using the latest versions of all components.

> **Note**
>
> There is no data interruption when a data source sync status fails. The source data continues to flow into AWS IoT SiteWise. The configuration simply isn't syncing with the IoT SiteWise OPC UA collector component on your AWS IoT Greengrass V2 deployment.

To change the timestamp configuration for a property group

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the gateway to edit.
4. In the **Data sources** section, select the data source with the failed sync status, and choose **Edit**.
5. Expand **Advanced configuration**, then expand **Group settings**.
6. In **Timestamp**, select **Source**. Selecting **Source** removes the `timestampToReturn` property from the configuration. This setting enables the collection of the data source timestamp from your device by default, allowing the data source to sync with the IoT SiteWise OPC UA collector component.
7. Choose **Save**.

Converted data types are not included

If you see an error when converting unsupported OPC UA data types to strings in AWS IoT SiteWise, there are a few possible reasons:

- The data type you're attempting to convert is a complex data type. Complex data types are not supported.
- When using **Destinations** as **AWS IoT SiteWise Buffered using Amazon S3**, the full string value is preserved in files pushed to an Amazon S3 bucket. When you later ingest data into AWS IoT SiteWise, full string values longer than 1024 bytes are rejected.

Trust store issues

If you encounter issues related to trust stores in SiteWise Edge, consider the following troubleshooting steps:

- Verify that the AWS IoT Greengrass root CA certificate is present and correctly formatted in the appropriate trust stores
- Ensure that the Java KeyStore password is correctly set and accessible to SiteWise Edge components
- Check that any custom certificates (such as for HTTPS proxies) are in the correct format (typically PEM) and properly imported into the trust stores
- Confirm that the trust stores have the correct file permissions and are accessible to the SiteWise Edge processes
- Review the SiteWise Edge logs for any SSL/TLS related errors, which may indicate trust store issues
- Test SSL/TLS connections independently using tools like `openssl` to verify trust store functionality

Proxy-enabled installation issues

If you encounter issues during the proxy configuration process, consider the following troubleshooting steps:

- Verify that the proxy URL is correctly formatted and includes the proper scheme (`http://` or `https://`)

- Ensure that any proxy credentials are URL-encoded if they contain special characters
- Confirm that the no-proxy list includes all necessary local addresses and AWS service endpoints
- For HTTPS proxies, verify that the provided CA certificate is in PEM format
- Review the installation logs for specific error messages that may indicate the source of the problem
- Test the proxy connection independently to ensure it's functioning correctly

Troubleshooting the AWS IoT SiteWise Edge application on Siemens Industrial Edge

To troubleshoot the AWS IoT SiteWise Edge application on your Siemens Industrial Edge device, you can access the logs for the application through the Siemens Industrial Edge Management or Siemens Industrial Edge Device (IED) portals. For more information, see Downloading Logs in the Siemens documentation.

My data doesn't display in AWS IoT SiteWise

- Ensure that there are no issues with your Databus users and that the checkmark icon for the **Databus_Configuration** is green rather than gray.
- You may not be running Siemens Industrial Edge Management on a version that contains Secure Storage. Upgrade your version of Siemens OS. For more information, see Siemens Secure Storage and the AWS IoT SiteWise Edge application.

I see 'Config file missing AWS_REGION' in the logs

If you see `Config file missing AWS_REGION` in the Siemens logs, the JSON of the configuration file has been corrupted. You'll need to create a new configuration file. Follow the steps in Create a gateway for Siemens Industrial Edge to create a new configuration file.

I see an 'Out of sync' error message on the Edge gateway configuration

If you see an `Out of sync` error message on your Siemens Industrial Edge gateway after deployment is complete, it means that IoT SiteWise publisher component is out of sync with your gateway. The IoT SiteWise publisher component works in the background on Siemens Industrial Edge gateways to provide MQTT topic functionality. We upgraded Siemens Industrial Edge gateways to use the capability namespace `iotsitewise:publisher:3` rather than `iotsitewise:publisher:2`. You can update to the latest version of the publisher to resolve this issue.

To upgrade to the latest version of the IoT SiteWise publisher

1. Navigate to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Edge gateways**.
3. Select the Siemens Industrial Edge gateway to edit.
4. In the **Edge capabilities** section, select the **View software versions**.
5. Select the latest version of the IoT SiteWise publisher under the **Publisher** dropdown menu.
6. Choose **Done**.

Troubleshooting open-source integrations at the Edge

This section provides solutions for common issues you might encounter when integrating open-source tools with SiteWise Edge.

> **Note**
>
> Node-RED®, InfluxDB®, and Grafana® are not vendors or suppliers for SiteWise Edge.

Connection issues

Node-RED can't connect to MQTT broker

Verify that the MQTT broker is running and accessible on the specified port. Check your network configuration and ensure that the broker address is correct.

To verify the MQTT broker status, run:

```
docker ps | grep emqx
```

InfluxDB connection errors

Ensure that your authentication token is valid and that you've specified the correct organization and bucket names. Check that InfluxDB is running and accessible.

To verify InfluxDB status, run:

```
curl -I http://localhost:8086
```

Grafana can't connect to InfluxDB

Verify that the InfluxDB data source configuration in Grafana is correct, including the URL, authentication token, organization, and bucket.

Data flow issues

No data appearing in AWS IoT SiteWise

Check that your property alias in the Node-RED flow matches the expected format. Verify that the MQTT topic structure is correct and that the SiteWise Edge gateway is properly configured to receive data from the MQTT broker.

No SiteWise Edge data stored in InfluxDB

Verify that the Node-RED retention flow is correctly configured and that the InfluxDB writer node has the proper bucket and measurement settings. Check the Node-RED debug output for any errors.

Data format errors

Ensure that your data transformation functions correctly convert data between formats. Use the Node-RED debug nodes to inspect the data at each stage of the flow.

Performance issues

High CPU or memory usage

Monitor resource usage and adjust the configuration of your components as needed. Consider reducing the data collection frequency or implementing data filtering to reduce the processing load.

To monitor resource usage, run:

```
docker stats
```

Slow Grafana dashboard loading

Optimize your InfluxDB queries and consider adding time range limitations to your dashboard panels. Reduce the number of data points displayed by using appropriate aggregation functions.

Logging and diagnostics

To troubleshoot issues, check the logs for each component:

Node-RED logs

View logs in the Node-RED console or run:

```
docker logs node-red
```

InfluxDB logs

Access logs by running:

```
docker logs influxdb
```

Grafana logs

View logs by running:

```
docker logs grafana
```

SiteWise Edge logs

Check the SiteWise Edge gateway logs for MQTT connection and data processing issues. For more information, see Troubleshooting a SiteWise Edge gateway.

Troubleshooting AWS IoT Greengrass issues

To find solutions to many issues configuring or deploying your SiteWise Edge gateway on AWS IoT Greengrass, see Troubleshooting AWS IoT Greengrass in the *AWS IoT Greengrass Developer Guide*.

Troubleshoot an AWS IoT SiteWise portal

Troubleshoot common issues with your AWS IoT SiteWise portals.

Users and administrators can't access AWS IoT SiteWise portal

If users or administrators cannot access your AWS IoT SiteWise portal, you may have restricted permissions in attached AWS Identity and Access Management (IAM) policies that prevent successful logins.

See the following examples of IAM policies that will result in login failure:

> **Note**
>
> Any attached IAM policies that include a `"Condition"` element will cause a login failure.

Example 1: The condition here is a limited IP, and this will cause a login failure.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribePortal"
            ],
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```

Example 2: The condition here is an included tag, and this will cause a login failure.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribePortal"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/project": "*"
                }
            }
        }
    ]
}
```

When adding users or administrators to the portal, avoid creating IAM policies that restrict user permissions, such as limited IP. Any attached policies with restricted permissions will not be able to connect to the AWS IoT SiteWise portal.

Troubleshoot 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 Amazon 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

- [Configure AWS IoT Core logs]

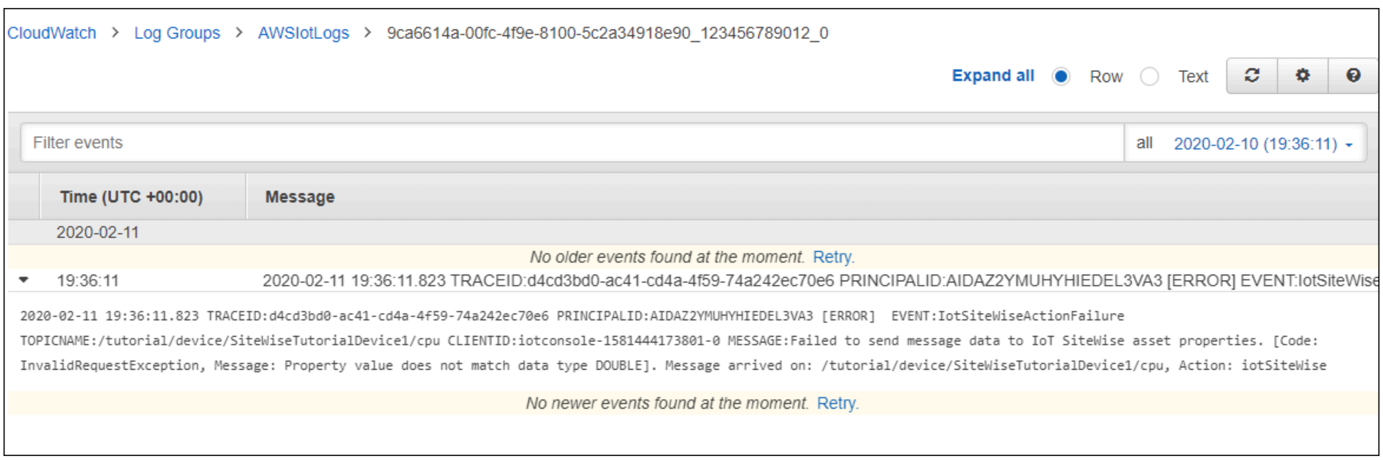
- Configure a republish error action
- Troubleshoot rule issues
- Troubleshoot a rule (AWS IoT SiteWise)
- Troubleshoot a rule (DynamoDB)

Configure 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.

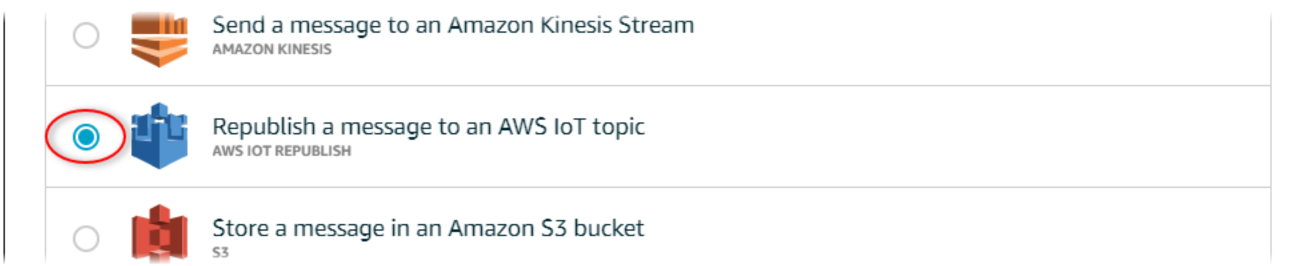
Configure a republish error action

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.

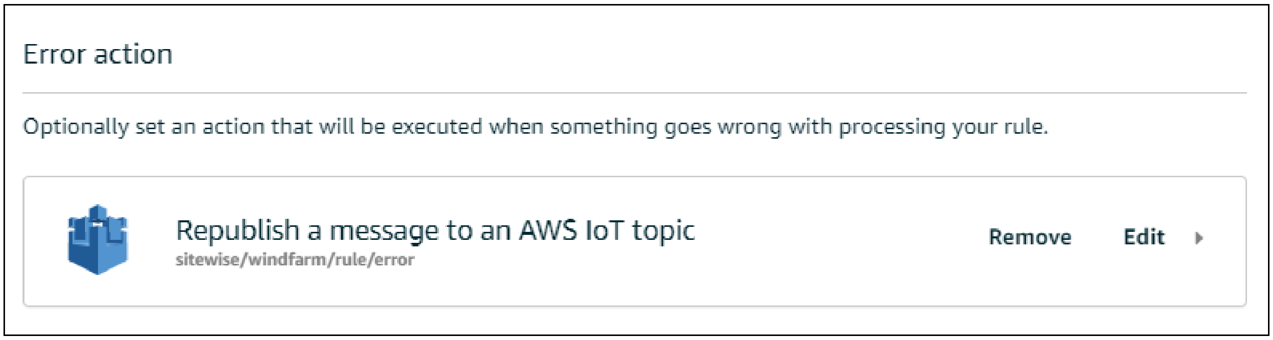
To add a republish error action to a rule

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.



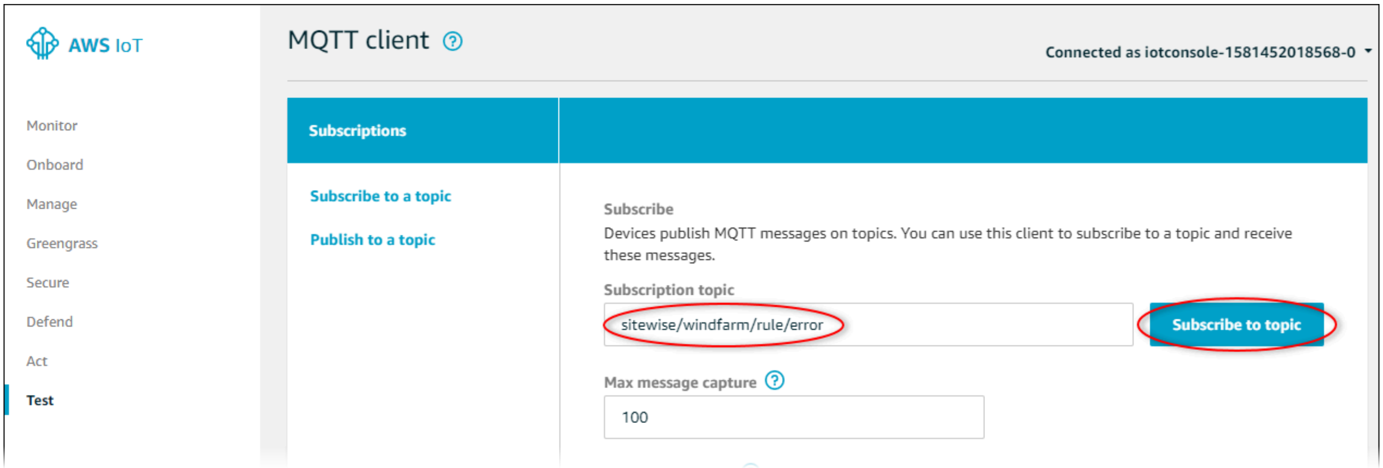
12. Choose the back arrow in the upper left of the console to return to the AWS IoT console home.

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](https://console.aws.amazon.com/iot/).
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.

Troubleshoot rule 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
- Error: Property value does not match data type <type>
- Error: User: <role-arn> is not authorized to perform: iotsitewise:BatchPutAssetPropertyValue on resource
- Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: <role-arn>
- Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates.

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, `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 Manage data streams for AWS IoT SiteWise.
- Check that your rule has a role and that the role allows `iotsitewise:BatchPutAssetPropertyValue` permission to the targeted asset property, such as through the target asset's hierarchy. For more information, see Grant AWS IoT the required access.

Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: <role-arn>

Your user isn't authorized to assume the role on your rule in AWS Identity and Access Management (IAM).

Check that your user is allowed `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 AS clauses in your rule query statement. For more information, see Substitution templates in the *AWS IoT Developer Guide*.

Troubleshoot a rule (AWS IoT SiteWise)

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 Troubleshoot an AWS IoT SiteWise rule action.

To add a republish error action to a rule

1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose **Message routing** and then choose **Rules**.
3. Choose the rule that you created earlier and choose **Edit**.
4. Under **Error action - *optional***, choose **Add error action**.
5. Choose **Republish a message to an AWS IoT topic**.
6. In **Topic**, enter the path to your error (for example, **`sitewise/rule/tutorial/error`**). AWS IoT Core will republish error messages to this topic.
7. Choose the role that you created earlier (for example, **SiteWiseTutorialDeviceRuleRole**).
8. Choose **Update**.

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 **MQTT test client** to open the MQTT test client.
3. In the **Topic filter** field, enter **`sitewise/rule/tutorial/error`** and choose **Subscribe**.

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 Troubleshoot an AWS IoT SiteWise rule action.

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.

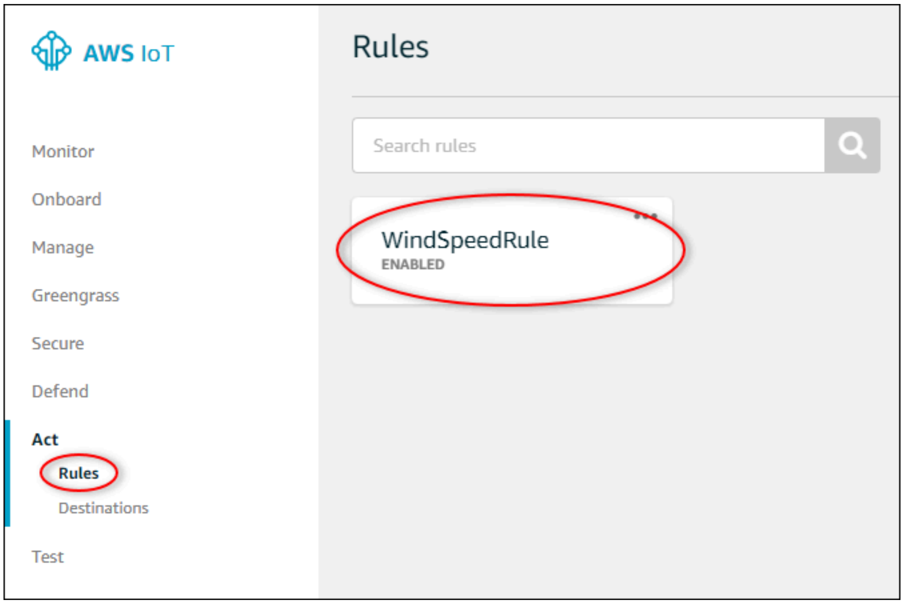
> **Note**
>
> You can also subscribe to your device's shadow update topic to view the payload that your AWS IoT SiteWise action parses. To do so, subscribe to the following topic.
>
> ```
> $aws/things/+/shadow/update/accepted
> ```

Troubleshoot a rule (DynamoDB)

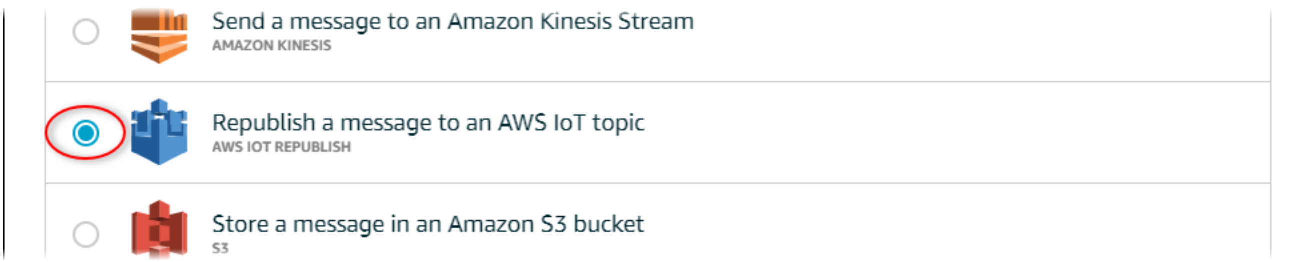
Follow the steps in this procedure to troubleshoot your rule if the demo asset data isn't appearing in the DynamoDB table as expected. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client. You can also configure logging to CloudWatch 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

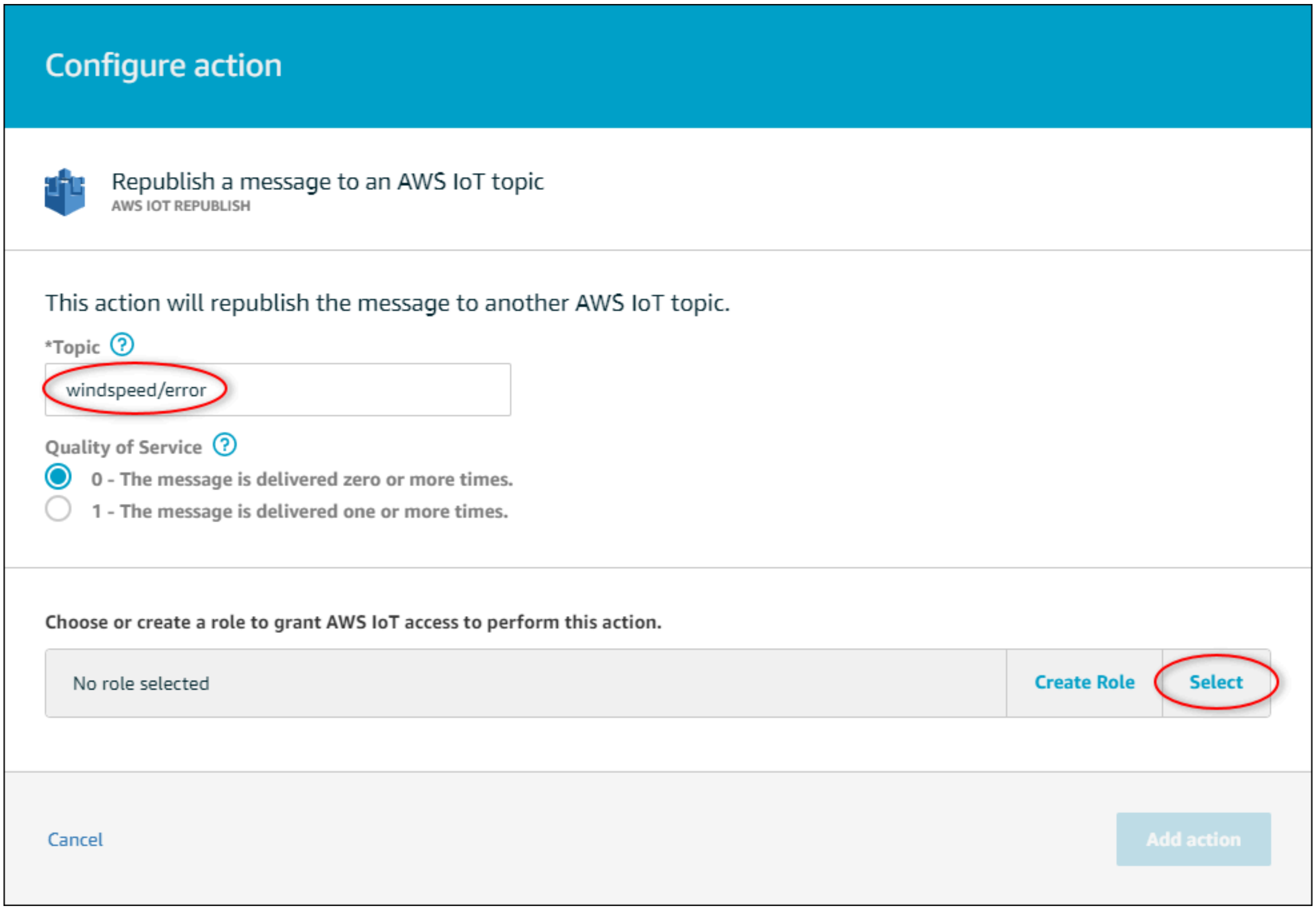
1. Navigate to the AWS IoT console.
2. In the left navigation pane, choose **Act** and then choose **Rules**.
3. Choose the rule that you created earlier.



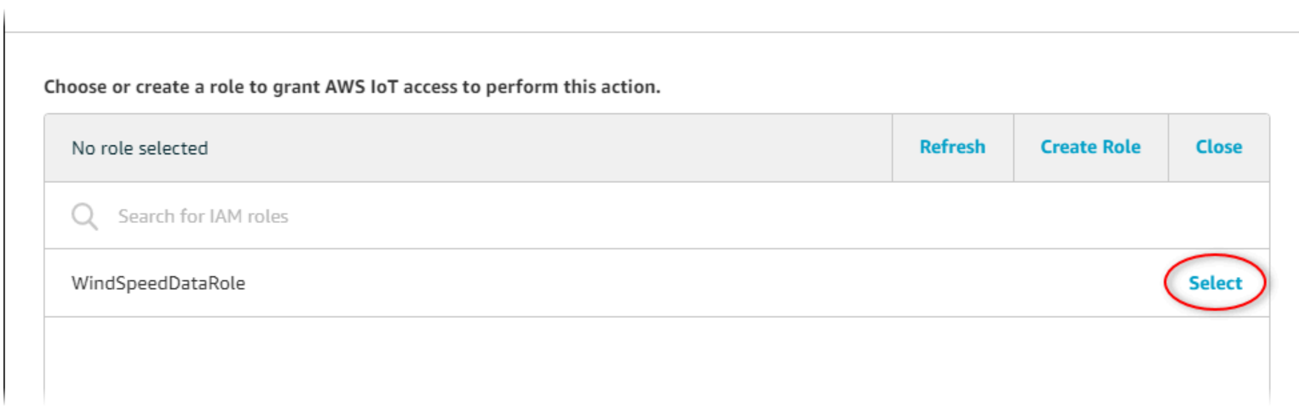
4. Under **Error action**, choose **Add action**.
5. Choose **Republish a message to an AWS IoT topic**.



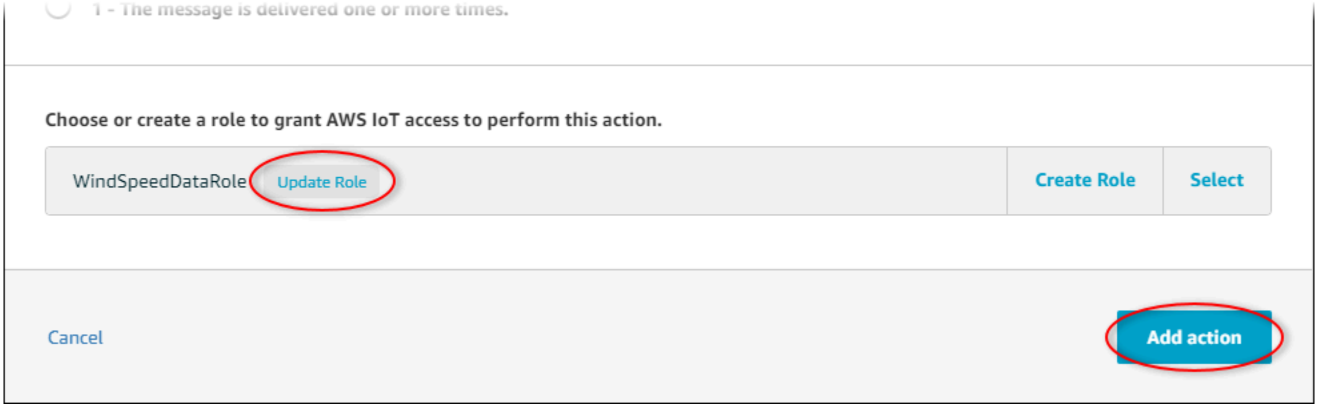
6. Choose **Configure action** at the bottom of the page.
7. In **Topic**, enter **`windspeed/error`**. AWS IoT Core will republish error messages to this topic.



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.



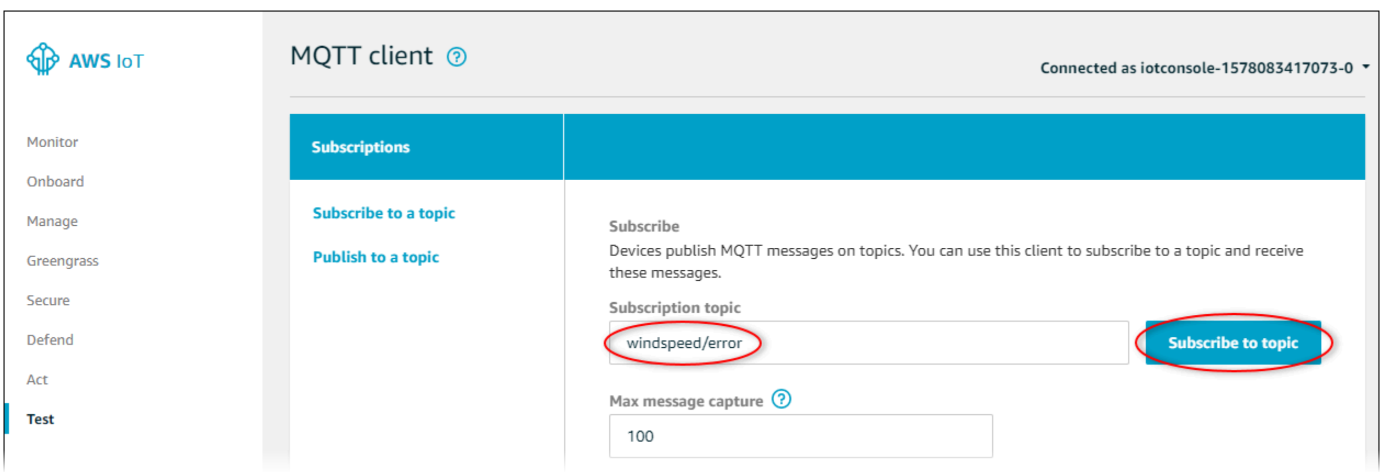
11. Choose **Add action** to finish adding the error action.
12. Choose the back arrow in the upper left of the console to return to the AWS IoT Core console home.

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**.



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.

Troubleshooting bulk import and export operations

To handle and diagnose errors produced during a transfer job, see the AWS IoT TwinMaker **GetMetadataTransferJob** API:

1. After creating and running a transfer job, call the **GetMetadataTransferJob** API:

```
aws iottwinmaker get-metadata-transfer-job \
--metadata-transfer-job-id your_metadata_transfer_job_id \
--region us-east-1
```

2. The state of the job changes to one of the below states:

   - COMPLETED
   - CANCELLED
   - ERROR

3. The **GetMetadataTransferJob** API returns a MetadataTransferJobProgress object.
4. The **MetadataTransferJobProgress** object contains the following parameters:

   - **failedCount** : Indicates the count of assets that failed during the transfer process.
   - **skippedCount** : Indicates the count of assets that were skipped during the transfer process.
   - **succeededCount** : Indicates the count of assets that succeeded during the transfer process.
   - **totalCount** : Indicates the total count of assets involved in the transfer process.

5. Additionally a **reportUrl** element is returned by the API call, which contains a pre-signed URL. If your transfer job has errors that needs investigation, you can download a full error report at this URL.

AWS IoT SiteWise endpoints and quotas

The following sections describe the endpoints and quotas for AWS IoT SiteWise.

Topics

AWS IoT SiteWise endpoints

The AWS General Reference Guide lists the AWS IoT SiteWise endpoints for an AWS account. For more information, see AWS IoT SiteWise endpoints and quotas in the *AWS General Reference Guide*.

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*. For more information about AWS IoT SiteWise quotas, see AWS IoT SiteWise service quotas in the *AWS General Reference*.

Quotas for AWS IoT SiteWise assets and asset models

Resource	Description	Quota	Adjustable
Number of asset models in each AWS Region for each AWS account	The maximum number of asset models that you can create in an AWS Region for an AWS account.	10000	Yes
Number of assets in each asset model	The maximum number of assets that you can create for each asset model.	10,000	Yes

Resource	Description	Quota	Adjustable
Number of child assets in each parent asset	The maximum number of child assets that you can associate with a parent asset.	2000	Yes
Depth of asset model hierarchy tree	The maximum asset hierarchy tree depth for an asset model.	30	Yes
Number of hierarchy definitions in each asset model	The maximum number of hierarchy definitions you can have in an asset model.	30	Yes
Number of propertie s in the root level in each asset model	The maximum number of `assetModelProperties` for each asset model. This count does not include `compositeModelProperties`. This quota also applies to any unique asset created from this asset model.	500	Yes

Resource	Description	Quota	Adjustable
Number of properties in an asset model	The maximum number of properties of an asset model of type `ASSET_MODEL` or `COMPONENT_MODEL`. This number is determined by combining the properties of the root asset model and any included component-model-based or inline composite models. This quota also applies to any unique asset created from this asset model.	5000	Yes
Number of properties in each composite model	The maximum number of properties allowed for composite models. Also, the maximum number of properties allowed for an asset model of type `COMPONENT_MODEL`.	100	Yes

Resource	Description	Quota	Adjustable
Depth of property tree in an asset model	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.	10	No
Number of asset models in each hierarchy tree	The maximum number of asset models you can include in a single hierarchy tree.	100	Yes

Resource	Description	Quota	Adjustable
Number of directly dependent properties for an asset model	This quota limits how many properties can directly depend on a single property, as defined in property formula expressions. The number of dependent properties must be greater than the number of directly dependent properties for an asset model. Request an increase for both quotas if there are more directly dependent properties than dependent properties for an asset model.	20	Yes
Number of dependent properties in an asset model	This quota limits how many properties can directly or indirectly depend on a single property, as defined in property formula expressions.	30	No
Number of composite models in an asset model	The maximum number of composite models you can have on a single asset model.	50	Yes

Resource	Description	Quota	Adjustable
Composite model depth	The maximum depth of the composite model tree in each asset model, including inline and component-model-based composite models.	2	Yes
Number of unique asset models that use the same component model	The maximum number of unique asset models that have at least one component-model-based composite model that directly references a specific asset model of type COMPONENT _MODEL.	20	Yes
Number of property variables in a property formula expression	For example, there are two property variables , `power` and `temp`, in the expressio n `avg(power) + max(temp)` . This also applies for transform computati on results.	10	No

Resource	Description	Quota	Adjustable
Number of functions in a property formula expression	For example, there are two functions , `avg` and `max`, in the expression `avg(power) + max(temp)` .	10	No

Quotas for AWS IoT SiteWise interfaces

Quotas for AWS IoT SiteWise interfaces

Resource	Description	Quota	Adjustable
Number of interface s in each AWS Region for each AWS account	The maximum number of interfaces that you can create in an AWS Region for an AWS account.	100	Yes
Number of properties in each interface	The maximum number of properties that you can define in an interface.	200	Yes
Number of unique asset models that use the same interface model	The maximum number of unique asset models that can use the same interface model.	500	Yes
Number of interfaces in each hierarchy tree	The maximum number of interface s you can include in a single hierarchy tree.	100	Yes

Resource	Description	Quota	Adjustable
Depth of interface hierarchy tree	The maximum interface hierarchy tree depth.	30	Yes
Number of hierarchy definitions in each interface	The maximum number of hierarchy definitions you can have in an interface.	20	Yes

Quotas for AWS IoT SiteWise asset property data

Resource	Description	Quota	Adjustable
Request rate for asset property data API operations	The maximum number of asset property data API requests each second that you can perform in each AWS Region in each AWS account. This quota applies to API operations such as `GetAssetPropertyValue` and `BatchPutAssetPropertyValue` .	1000	Yes
Number of data points each second for each data quality for each asset property	This quota applies to the maximum number of timestamp-quality-value (TQV) data points with the same timestamp in	10	No

Resource	Description	Quota	Adjustable
	seconds for each 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.		
Number of `BatchPutAssetPropertyValue` entries ingested each second in each non-attribute asset property for each AWS Region for an AWS account.	The maximum number of entries in each non-attribute asset property for `BatchPutAssetPropertyValue` from all sources, including SiteWise Edge gateways, AWS IoT Core rules, and API calls.	10	No

Resource	Description	Quota	Adjustable
Number of `BatchPutAssetPropertyValue` entries ingested each second for each attribute asset property for each AWS Region in an AWS account.	This quota applies to entries in each attribute asset property for `BatchPutAssetPropertyValue` from all sources, including SiteWise Edge gateways, AWS IoT Core rules, and API calls.	1	No
Rate of data points ingested	The maximum number of timestamp-quality-value (TQV) data points ingested per second in each AWS Region for an AWS account.	5000	Yes
Request rate for `BatchGetAssetPropertyAggregates`	The maximum number of `BatchGetAssetPropertyAggregates` requests each second that you can perform in each AWS Region in each AWS account.	200	Yes

Resource	Description	Quota	Adjustable
Request rate for `BatchGetAssetPropertyValue`	The maximum number of `BatchGetAssetPropertyValue` requests each second that you can perform in each AWS Region in each AWS account.	500	Yes
Request rate for `BatchGetAssetPropertyValueHistory`	The maximum number of `BatchGetAssetPropertyValueHistory` requests each second that you can perform .	200	Yes
Number of `BatchPutAssetPropertyValue` entries ingested each second for each asset property for each AWS Region in an AWS account.	This quota applies to entries in each asset property for `BatchPutAssetPropertyValue` from all sources, including SiteWise Edge gateways, AWS IoT Core rules, and API calls.	10	No

Resource	Description	Quota	Adjustable
Rate of `GetAssetPropertyAggregates` requests and `BatchGetAssetPropertyAggregates` entry queries for each asset property	The maximum number of total `GetAssetPropertyAggregates` requests and `BatchGetAssetPropertyAggregates` entries for each asset property per second in each AWS Region in each AWS account.	50	No
Rate of `GetAssetPropertyValue` requests and `BatchGetAssetPropertyValue` entry queries for each asset property	The maximum number of total `GetAssetPropertyValue` requests and `BatchGetAssetPropertyValue` entries for each asset property each second in each AWS Region in each AWS account.	500	No

Resource	Description	Quota	Adjustable
Rate of GetAssetPropertyValueHistory requests and BatchGetAssetPropertyValueHistory entry queries for each asset property	The maximum number of total GetAssetPropertyValueHistory requests and BatchGetAssetPropertyValueHistory entries for each asset property each second in each AWS Region in each AWS account.	30	No
Rate of GetInterpolatedAssetPropertyValues requests	The maximum number of GetInterpolatedAssetPropertyValues requests each second that you can perform in each AWS Region in each AWS account.	500	Yes
Number of results in each GetInterpolatedAssetPropertyValues request	The maximum number of results to return for a paginated GetInterpolatedAssetPropertyValues request.	10	Yes

Resource	Description	Quota	Adjustable
Rate of data points retrieved from `GetAssetPropertyValueHistory` and `BatchGetAssetPropertyValueHistory`	The maximum byte rate (MB/second) of datapoints retrieved each second for each AWS Region in an AWS account across `GetAssetPropertyValueHistory` and `BatchGetAssetPropertyValueHistory` . The response payload evaluated for this quota uses Timestamp-Quality-Value (TQV) fields for each datapoint and rounds the byte size for each API request to the next 4KB increment. Timestamp-quality-value (TQV) datapoints retrieved each second varies for each data type: • **Integer** – up to 5 Million TQV per second	100	Yes

Resource	Description	Quota	Adjustable
	• **Double** – up to 4 Million TQV per second • **Boolean** – up to 6 Million TQV per second • **String** – varies based on each string value size.		
Number of properties that depend on a single property within a linked asset model	The maximum number of properties that directly or indirectly depend on a single property within a linked asset model.	10	Yes

Quotas for SiteWise Edge gateways

Resource	Description	Quota	Adjustable
Number of SiteWise Edge gateways in each AWS Region for an AWS account	The maximum number of SiteWise Edge gateways that you can create in an AWS Region for an AWS account.	100	Yes
Number of OPC UA sources in a SiteWise Edge gateway	The maximum number of OPC UA sources you can configure in	100	No

Resource	Description	Quota	Adjustable
	an SiteWise Edge gateway.		
Total number of destinations in a SiteWise Edge gateway	The maximum number of destinati ons that you can configure in an SiteWise Edge gateway.	100	No

Quotas for AWS IoT SiteWise Monitor

Resource	Description	Quota	Adjustable
Number of portals in each AWS Region for an AWS account	The maximum number of SiteWise Monitor portals that you can create in an AWS Region for an AWS account.	100	Yes
Number of projects in a portal	The maximum number of projects that you can create within a SiteWise Monitor portal.	100	Yes
Number of dashboard s in a project	The maximum number of dashboard s that you can create within a project in SiteWise Monitor.	100	Yes
Number of root assets in a project	The maximum number of top-level	1	No

Resource	Description	Quota	Adjustable
	assets that you can add to a project in SiteWise Monitor.		
Number of visualizations in a dashboard	The maximum number of visual elements (such as charts, graphs, or tables) that you can add to a dashboard in SiteWise Monitor.	10	Yes
Number of metrics in each dashboard visualization	The maximum number of metrics or data points that you can display in a single visualization on a dashboard in SiteWise Monitor.	5	Yes
Number of thresholds for each dashboard visualization	The maximum number of threshold levels that you can set for each visualization on a dashboard in SiteWise Monitor.	12	No

Quotas for AWS IoT SiteWise bulk import and export of metadata

Resource	Description	Quota	Adjustable
Number of metadata transfer jobs in queue	The maximum number of `PENDING`	10	Yes

Resource	Description	Quota	Adjustable
	metadata transfer jobs in the queue.		
Size of the metadata transfer job import file	The maximum size of the imported file (in MB).	100	Yes
Number of AWS IoT SiteWise import resources in a job	The maximum number of AWS IoT SiteWise import resources in a single job. A resource includes assets, and asset models.	5000	Yes
Number of AWS IoT SiteWise export resources in a job	The maximum number of AWS IoT SiteWise export resources in a single job. A resource includes assets, and asset models.	5000	Yes

Quotas for AWS IoT SiteWise bulk import of data

Resource	Description	Quota	Adjustable
Number of running bulk import jobs	The maximum number of bulk import jobs that can concurrently run.	100	No
Size of the CSV file	The maximum CSV file size (in GB) in a bulk import job.	10	No

Resource	Description	Quota	Adjustable
Size of the uncompressed parquet file	The maximum file size (in MB) for an uncompressed parquet file in a bulk import job.	256 MB	No
Size of the CSV file for buffered ingestion	The maximum CSV file size (in MB) when using buffered ingestion on a bulk import job.	256 MB	No
Size of the uncompressed parquet row group	The maximum size of an uncompressed parquet row group.	64 MB	No
Number of unique measurements in a parquet file	The maximum number of unique measurements in a parquet file.	10000	No
Number of days between the timestamp in the past and today for buffered ingestion	The maximum number of days between a timestamp in the past and today's date when using buffered ingestion.	30	Yes
Request rate for `CreateBulkImportJobs` in each AWS Region in each AWS account		10	Yes

Resource	Description	Quota	Adjustable
Request rate for `ListBulkImportJobs` for each AWS Region in each AWS account		50	Yes
Request rate for `DescribeBulkImportJobs` for each AWS Region in each AWS account		50	Yes

Quotas for AWS IoT SiteWise Assistant API throttling

Quotas for AWS IoT SiteWise Assistant API throttling limits

Resource	Description	Quota	Adjustable
Request rate for `InvokeAssistant` operation	The maximum number of transactions each minute (TPM) that can be made to the AWS IoT SiteWise `InvokeAssistant` API in an AWS account. The TPM limits apply to all supported regions, and are adjustable in some regions.	10	No

Quotas for anomaly detection

Quotas for AWS IoT SiteWise native anomaly detection

Resource	Description	Quota	Adjustable
Maximum number of computational models	The maximum number of computational models.	1000	Yes
Maximum number of input properties for each computational model	The maximum number of input properties a computational model can have.	80	No
Maximum number of TQV's in training data (after resampling)	The maximum number of TQV's in training data after resampling.	1,500,000	No
Maximum number of TQV's in evaluation data (after resampling)	The maximum number of TQV's in evaluation data after resampling.	1,500,000	No
Minimum timespan of training data	The minimum time range for training data. See Understand the minimum date range for details.	14 days	No
Maximum number of simultaneous `ACTIVE` training jobs	The maximum number of simultaneous `ACTIVE` training jobs at any given time.	25	Yes

Resource	Description	Quota	Adjustable
Maximum size of label data file	The maximum size of the label data file.	2 MB	No
Maximum number of `ACTIVE` high frequency inferences	The maximum number of `ACTIVE` high frequency inferences at any given time.	250	Yes
Maximum number of `ACTIVE` low frequency inferences	The maximum number of `ACTIVE` low frequency inferences at any given time.	500	Yes
Maximum time for `ACTIVE` training duration	The maximum time that `ACTIVE` training can take place for.	24 hours	Yes

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

Change	Description	Date
Interfaces for asset models	Define a common structure that ensures consistency across asset models while allowing for variations in implementation.	August 5, 2025
Added native anomaly support to AWS IoT SiteWise	AWS IoT SiteWise native anomaly detection is a machine learning (ML) service for monitoring industrial equipment that detects abnormal equipment behavior and identifies potential failures. With native anomaly detection, you can implement predictiv e maintenance programs and identify suboptimal equipment processes.	July 28, 2025
Added Rich SQL functions , and new SQL datayptes , clauses and operators to existing SQL	Added support for new dataypes, clauses and operators.	July 22, 2025
Process and visualize data with SiteWise Edge and open-source tools	Integrate open source tools like Node-RED®, InfluxDB® , and Grafana® on AWS IoT SiteWise MQTT-enab	July 3, 2025

	led, V3 gateways. Use local processing and visualization to enhance your industrial data management capabilit ies.	
[Connect external applicati ons to the EMQX broker with usernames and passwords]()	Added support for connectin g external applications using usernames and passwords to the AWS IoT Greengrass EMQX broker that is deployed by default when creating a new MQTT-enabled, V3 gateway.	May 1, 2025

[Support for MQTT-enabled, V3 gateways on SiteWise Edge]()	Added new features and removed deprecated content • Added support for MQTT-enabled, V3 gateways. Enhanced destination configuration using path filters to subscribe to MQTT topics, including real-time data ingestion directly to AWS IoT SiteWise or buffered data ingestion using Amazon S3. • Released version 3.0.0 of the IoT SiteWise OPC UA collector and version 4.0.0 of the IoT SiteWise publisher component for AWS IoT Greengrass V2. • Renamed the previous version of self-hosted SiteWise Edge gateways to Classic streams, V2 gateways. • Removed references to AWS IoT Greengrass V1 in SiteWise Edge documenta tion because it's no longer supported for use with AWS IoT SiteWise.	February 26, 2025
[Support for AWS IoT SiteWise Assistant]()	Added support for the AWS IoT SiteWise Assistant - a generative AI-powered assistant.	November 18, 2024

Added configurable session timeouts for SiteWise Edge APIs	Added configurable session timeout settings to manage inactivity periods for AWS OpsHub and SiteWise Edge APIs.	October 31, 2024
Added configurable proxy settings for SiteWise Edge APIs	Added managing trust store information to enable HTTPS proxy support for SiteWise Edge gateways.	October 31, 2024
Enable CORS on for SiteWise Edge APIs	Added CORS support for SiteWise Edge APIs to enable secure cross-domain web application access.	September 30, 2024
Support for CloudRail and Litmus Edge partner data sources	Added support for both CloudRail and Litmus Edge as partner data sources.	September 5, 2024
General availability for running SiteWise Edge on Siemens Industrial Edge	AWS IoT SiteWise now supports general availabil ity of running SiteWise Edge on Siemens Industrial Edge devices.	July 24, 2024
Added support for timestamp configuration on OPC UA data sources	AWS IoT SiteWise now supports timestamp configura tion for OPC UA data sources.	July 24, 2024
Added support for data type conversion on OPC UA data sources	AWS IoT SiteWise now supports data type conversion for unsupported OPC UA data types.	July 24, 2024

Added support for running a preview of SiteWise Edge on Siemens Industrial Edge	AWS IoT SiteWise now supports running a preview of SiteWise Edge on Siemens Industrial Edge devices.	November 26, 2023
Added support for warm tier storage	AWS IoT SiteWise now supports warm storage, a fully-managed storage tier that makes it easy for customers to securely store and access industrial data.	November 15, 2023
Added support for user-defi ned unique identifiers	AWS IoT SiteWise now supports the use of user-defi ned unique identifiers for asset, asset models, propertie s and hierarchies.	November 15, 2023
Added support for multi variate anomaly detection of industrial assets	AWS IoT SiteWise now supports multi variate anomaly detection of industrial assets by integrati on of historical and real time equipment data with Amazon Lookout for Equipment.	November 15, 2023
Added support for cost-effi cient and scalable ingestion of time-series data in AWS IoT SiteWise	AWS IoT SiteWise now supports cost-efficient and scalable ingestion of time-seri es data needed for analytical use cases.	November 15, 2023
Added support for bulk import, export, and update	AWS IoT SiteWise now supports bulk import, export, and update industrial equipment metadata.	November 15, 2023

Added support for asset model components	AWS IoT SiteWise now supports Asset model components to help industria l customers create reusable components.	November 15, 2023
Added support for IoT dashboard application	AWS IoT SiteWise now supports an open source dashboard application where you can visualize and interact with operational data.	November 15, 2023
Updated the service-linked roles for AWS IoT SiteWise	AWS IoT SiteWise has new service-linked roles, and can run a metadata search query, against the AWS IoT TwinMaker database.	November 6, 2023
Updated tagging for AWS IoT SiteWise data stream resources	Added support for tagging data stream resources.	August 18, 2022
Updated SiteWise Edge gateways	You can now configure the publisher to control what data is sent from the edge to the cloud and the order that it's sent to the cloud.	January 12, 2022
Updated the AWS IoT SiteWise demo	You can now use the demo to create a SiteWise Monitor portal.	January 10, 2022
Updated storage managemen t	You can now define a retention period to control how long your data is kept in the hot tier.	November 29, 2021

Added support for data stream management	You can now ingest data to AWS IoT SiteWise before you create asset models and assets.	November 24, 2021
Updated asset model hierarchies	A child asset model now can be associated with multiple parent asset models.	October 28, 2021
Region launch	Launched AWS IoT SiteWise in AWS GovCloud (US-West).	September 29, 2021
Updated functions	Added the following features • In metrics, you can use nested expressions in aggregation functions and temporal functions. • 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.	August 10, 2021
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
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 `StreamManager` to improve handling when the source connection is lost and re-established. This version also accepts OPC UA values with a `ServerTimestamp` when no `SourceTimestamp` is available.	January 22, 2021
Date and time functions	AWS IoT SiteWise now supports date and time functions.	January 21, 2021
Function syntax	You can now use Uniform Function Call Syntax (UFCS) for AWS IoT SiteWise functions.	January 11, 2021
Integrating with Grafana	Added information about how to visualize AWS IoT SiteWise data in Grafana dashboards.	December 15, 2020

AWS IoT SiteWise feature release	You can now monitor your data with alarms, process industrial data at the edge, use Modbus TCP and Ethernet/IP sources to your SiteWise Edge gateway, filter incoming data with deadbands, and more. • Added the Monitoring data with alarms section that you can use to define, configure, and respond to alarms in AWS IoT SiteWise. • Added the Edge processing section that you can use to configure processing of your industrial data on your edge devices. • Added the Modbus TCP and Ethernet/IP sections to the SiteWise Edge gateway source documentation. • Added the source destination section that you can use to customize where you send your incoming industrial data. • Added the OPC UA filtering section that you can use to control the frequency and type of data that is sent to your SiteWise Edge gateway from your industrial local server.	December 15, 2020

AWS IoT SiteWise now supports customer managed CMKs.	AWS IoT SiteWise now supports encryption with customer managed CMKs.	November 24, 2020
IoT SiteWise connector version 8 released	Version 8 of the IoT SiteWise connector is available. This release improves stability when the connector experiences intermittent network connectivity.	November 19, 2020
Using strings and conditionals in formula expressions	Added information about how to strings and conditional functions in formula expressions for transforms and metrics.	November 16, 2020
Ingesting data using AWS IoT Greengrass stream manager	Added information about how to ingest high-volume IoT data from local data sources using an AWS IoT Greengrass edge device.	September 16, 2020
VPC endpoints (AWS PrivateLink)	Added information about how to establish a private connection between your virtual private cloud (VPC) and the AWS IoT SiteWise data APIs by creating an interface VPC endpoint.	September 4, 2020
IoT SiteWise connector version 7 released	Version 7 of the IoT SiteWise connector is available. This release fixes an issue with SiteWise Edge gateway metrics.	August 14, 2020

Creating IAM Identity Center users from the AWS IoT SiteWise console	Added information about how you can create IAM Identity Center users in the AWS IoT SiteWise console. You can now create IAM Identity Center users when you assign users to a new or existing portal. Updated the Visualizing and sharing wind farm data tutorial to use this feature. This change reduces the number of steps in the tutorial.	August 4, 2020
Improved SiteWise Edge gateway troubleshooting	Added additional information about how to troubleshoot a SiteWise Edge gateway and how to export the OPC UA client certificate for a source.	June 18, 2020
Console task documentation	Added console task documentation for Modeling industrial assets, Querying asset property data, and Interacting with other services. You can follow these instructions to complete tasks in the AWS IoT SiteWise console.	June 11, 2020
Analyzing exported data tutorial	Added a tutorial that you can follow to learn how to use Amazon Athena to analyze asset data that you exported to Amazon S3 with the export feature AWS CloudFormation template.	May 27, 2020

Improved using formula expressions	Added detailed informati on about the behavior of AWS IoT SiteWise formula properties and added an example of how to count filtered data points.	May 18, 2020
IoT SiteWise connector version 6 released	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 SiteWise Edge 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.	April 29, 2020

AWS IoT SiteWise feature release	AWS IoT SiteWise feature release. You can now manage SiteWise Edge gateways with the API, add your logo to portals, view SiteWise Edge gateway metrics, and more. • Added the Exporting data to Amazon S3 section with an AWS CloudFormation template that you can use to export new data values to an Amazon S3 bucket. • Added the Configuring data sources section that improves SiteWise Edge gateway source documenta tion and includes the new SiteWise Edge gateway APIs. • Added the SiteWise Edge gateway metrics section that describes the CloudWatch metrics that SiteWise Edge gateways publish. • Added the Configuring an SiteWise Edge gateway on Amazon EC2 section with an AWS CloudForm ation template that you can use to quickly configure SiteWise Edge gateway dependencies on an Amazon EC2 instance.	April 29, 2020

	• Added the portal service roles section that describes the new permissions feature of SiteWise Monitor portals. • Updated portal documentation for portal service roles and portal logos. • Added the Tagging your AWS IoT SiteWise resources section. • Updated the Creating dashboards (CLI) section for the new dashboard definition structure. • Added the Security section.	
Ingesting data from AWS IoT Events	Added information about how to ingest data from AWS IoT Events when an event occurs.	April 20, 2020
Visualizing and sharing wind farm data in SiteWise Monitor tutorial	Added a tutorial that you can follow to learn how to use AWS IoT SiteWise Monitor to visualize and share asset data.	March 12, 2020
AWS IoT SiteWise concepts	Added a glossary of AWS IoT SiteWise concepts that you can use to learn about the service and its common terms.	March 5, 2020

Removed AWS IoT Greengrass installation instructions	Removed the AWS IoT Greengrass Core software installation instructions from the AWS IoT SiteWise User Guide. The AWS IoT Greengrass Developer Guide offers a device setup script and instructions to set up AWS IoT Greengrass on other platforms such as Amazon EC2 and Docker.	February 14, 2020
Improved ingesting data using AWS IoT Core rules	Added detailed informati on 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.	February 14, 2020
IoT SiteWise connector version 5 released	Version 5 of the IoT SiteWise connector is available. This release fixes a compatibility issue with AWS IoT Greengras s Core software v1.9.4.	February 12, 2020
IoT SiteWise connector version 4 released	Version 4 of the IoT SiteWise connector is available. This release fixes an issue with OPC UA server reconnection.	February 7, 2020

Restructured modeling industrial assets	Restructured the Updating Assets and Models section into multiple topics within Modeling Industrial Assets. • Asset and model states • Manage data streams for AWS IoT SiteWise • Update attribute values • Associate and disassociate assets • Update assets and models • Delete assets and models in AWS IoT SiteWise	February 4, 2020
Ingesting data from AWS IoT things tutorial	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.	February 4, 2020
Restructured retrieving data from AWS IoT SiteWise	Restructured the Retrieving Data section into two top-level sections: Querying asset property values and aggregates and Interacting with other AWS services.	January 21, 2020
Publishing property value updates to Amazon DynamoDB tutorial	Added a tutorial that you can follow to learn how to use property value notificat ions to store asset data in DynamoDB.	January 8, 2020

Using formula expressions	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.	January 7, 2020
Using OPC UA node filters	Added information about how to use OPC UA node filters to improve SiteWise Edge gateway performance when adding SiteWise Edge gateway sources.	January 3, 2020
Upgrading a connector	Added information about how to upgrade a SiteWise Edge gateway when a new connector version is released.	December 30, 2019
IoT SiteWise connector version 3 released	Version 3 of the IoT SiteWise connector is available. This release removes the iot:* permissions requirement.	December 17, 2019
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.	December 10, 2019
Creating dashboards (AWS CLI)	Added information about how to create a dashboard in AWS IoT SiteWise Monitor using the AWS CLI.	December 6, 2019

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. • Rewrote the asset modeling section for changes to assets, asset models, and asset hierarchies. • Updated the data ingestion section to include AWS IoT Greengrass connector steps and non-gateway data ingestion sections. • Added the AWS IoT SiteWise Monitor section and a separate application guide that shows how to use the SiteWise Monitor web application. • Added Query data from AWS IoT SiteWise and Interact with other AWS services sections. • Rewrote the getting started section to match the updated demo experience.	December 2, 2019
AWS IoT SiteWise version 1 released	Released initial preview for version 1 of AWS IoT SiteWise.	February 25, 2019