# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is AWS IoT SiteWise?</td>
<td>vii</td>
</tr>
<tr>
<td>How AWS IoT SiteWise works</td>
<td>1</td>
</tr>
<tr>
<td>Why use AWS IoT SiteWise?</td>
<td>1</td>
</tr>
<tr>
<td>Benefits</td>
<td>2</td>
</tr>
<tr>
<td>Use cases</td>
<td>2</td>
</tr>
<tr>
<td>Are you new to AWS IoT SiteWise?</td>
<td>3</td>
</tr>
<tr>
<td>Concepts</td>
<td>3</td>
</tr>
<tr>
<td>Key components</td>
<td>5</td>
</tr>
<tr>
<td>We want to hear from you</td>
<td>6</td>
</tr>
<tr>
<td>Getting started</td>
<td>7</td>
</tr>
<tr>
<td>Requirements</td>
<td>7</td>
</tr>
<tr>
<td>Setting up an AWS account</td>
<td>7</td>
</tr>
<tr>
<td>Using the quick start demo</td>
<td>8</td>
</tr>
<tr>
<td>Creating the AWS IoT SiteWise demo</td>
<td>9</td>
</tr>
<tr>
<td>Deleting the AWS IoT SiteWise demo</td>
<td>10</td>
</tr>
<tr>
<td>Tutorials</td>
<td>11</td>
</tr>
<tr>
<td>Calculating OEE</td>
<td>11</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>11</td>
</tr>
<tr>
<td>How to calculate OEE</td>
<td>11</td>
</tr>
<tr>
<td>Configuring alarms for asset property values</td>
<td>13</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>13</td>
</tr>
<tr>
<td>Configuring an alarm detection system</td>
<td>14</td>
</tr>
<tr>
<td>Configuring alarms</td>
<td>33</td>
</tr>
<tr>
<td>Visualizing and sharing wind farm data in SiteWise Monitor</td>
<td>42</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>43</td>
</tr>
<tr>
<td>Creating a portal in SiteWise Monitor</td>
<td>43</td>
</tr>
<tr>
<td>Signing in to a portal</td>
<td>46</td>
</tr>
<tr>
<td>Creating a wind farm project</td>
<td>47</td>
</tr>
<tr>
<td>Creating dashboards to visualize wind farm data</td>
<td>49</td>
</tr>
<tr>
<td>Creating AWS SSO users</td>
<td>55</td>
</tr>
<tr>
<td>Adding AWS SSO users to a portal</td>
<td>57</td>
</tr>
<tr>
<td>Assigning users to a project</td>
<td>58</td>
</tr>
<tr>
<td>Cleaning up resources</td>
<td>60</td>
</tr>
<tr>
<td>Ingesting data to AWS IoT SiteWise from AWS IoT things</td>
<td>62</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>63</td>
</tr>
<tr>
<td>Creating an AWS IoT policy</td>
<td>63</td>
</tr>
<tr>
<td>Creating and configuring an AWS IoT thing</td>
<td>65</td>
</tr>
<tr>
<td>Creating a device asset model</td>
<td>68</td>
</tr>
<tr>
<td>Creating a device fleet asset model</td>
<td>69</td>
</tr>
<tr>
<td>Creating and configuring a device asset</td>
<td>70</td>
</tr>
<tr>
<td>Creating and configuring a device fleet asset</td>
<td>72</td>
</tr>
<tr>
<td>Creating a rule in AWS IoT Core to send data to device assets</td>
<td>73</td>
</tr>
<tr>
<td>Running the device client script</td>
<td>77</td>
</tr>
<tr>
<td>Cleaning up resources</td>
<td>81</td>
</tr>
<tr>
<td>Troubleshooting a rule</td>
<td>82</td>
</tr>
<tr>
<td>Publishing property value updates to Amazon DynamoDB</td>
<td>85</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>86</td>
</tr>
<tr>
<td>Configuring AWS IoT SiteWise to publish property value updates</td>
<td>86</td>
</tr>
<tr>
<td>Creating a rule in AWS IoT Core</td>
<td>88</td>
</tr>
<tr>
<td>Creating a DynamoDB table</td>
<td>90</td>
</tr>
<tr>
<td>Configuring the DynamoDB rule action</td>
<td>91</td>
</tr>
<tr>
<td>Exploring data in DynamoDB</td>
<td>92</td>
</tr>
<tr>
<td>Cleaning up resources</td>
<td>93</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Troubleshooting a rule</td>
<td>96</td>
</tr>
<tr>
<td>Ingesting data to AWS IoT SiteWise</td>
<td>100</td>
</tr>
<tr>
<td>Using a gateway</td>
<td>100</td>
</tr>
<tr>
<td>Choosing a gateway platform</td>
<td>100</td>
</tr>
<tr>
<td>Configuring a gateway</td>
<td>101</td>
</tr>
<tr>
<td>Configuring data sources</td>
<td>116</td>
</tr>
<tr>
<td>Upgrading a connector</td>
<td>129</td>
</tr>
<tr>
<td>Using AWS IoT Core rules</td>
<td>131</td>
</tr>
<tr>
<td>Granting AWS IoT the required access</td>
<td>131</td>
</tr>
<tr>
<td>Configuring the AWS IoT SiteWise rule action</td>
<td>132</td>
</tr>
<tr>
<td>Reducing costs with basic ingest</td>
<td>136</td>
</tr>
<tr>
<td>Troubleshooting the AWS IoT SiteWise rule action</td>
<td>137</td>
</tr>
<tr>
<td>Using AWS IoT Events actions</td>
<td>137</td>
</tr>
<tr>
<td>Using the AWS IoT SiteWise API</td>
<td>137</td>
</tr>
<tr>
<td>Modeling industrial assets</td>
<td>139</td>
</tr>
<tr>
<td>Asset and model states</td>
<td>140</td>
</tr>
<tr>
<td>Defining asset models</td>
<td>141</td>
</tr>
<tr>
<td>Asset properties</td>
<td>142</td>
</tr>
<tr>
<td>Asset hierarchies</td>
<td>150</td>
</tr>
<tr>
<td>Creating assets</td>
<td>151</td>
</tr>
<tr>
<td>Mapping industrial data streams to asset properties</td>
<td>152</td>
</tr>
<tr>
<td>Updating attribute values</td>
<td>152</td>
</tr>
<tr>
<td>Associating assets</td>
<td>154</td>
</tr>
<tr>
<td>Updating assets and models</td>
<td>154</td>
</tr>
<tr>
<td>Updating assets</td>
<td>155</td>
</tr>
<tr>
<td>Updating asset models</td>
<td>155</td>
</tr>
<tr>
<td>Deleting assets and models</td>
<td>156</td>
</tr>
<tr>
<td>Deleting assets</td>
<td>156</td>
</tr>
<tr>
<td>Deleting asset models</td>
<td>156</td>
</tr>
<tr>
<td>Discovering your asset resources</td>
<td>156</td>
</tr>
<tr>
<td>Listing all asset models</td>
<td>157</td>
</tr>
<tr>
<td>Describing a specific asset model</td>
<td>157</td>
</tr>
<tr>
<td>Listing assets</td>
<td>159</td>
</tr>
<tr>
<td>Describing a specific asset</td>
<td>160</td>
</tr>
<tr>
<td>Listing assets associated to a specific asset</td>
<td>160</td>
</tr>
<tr>
<td>Describing a specific asset property</td>
<td>161</td>
</tr>
<tr>
<td>Monitoring data with web portals</td>
<td>163</td>
</tr>
<tr>
<td>SiteWise Monitor roles</td>
<td>164</td>
</tr>
<tr>
<td>SiteWise Monitor concepts</td>
<td>165</td>
</tr>
<tr>
<td>Getting started</td>
<td>166</td>
</tr>
<tr>
<td>Creating a portal</td>
<td>166</td>
</tr>
<tr>
<td>Enabling AWS SSO</td>
<td>167</td>
</tr>
<tr>
<td>Configuring your portal</td>
<td>168</td>
</tr>
<tr>
<td>Inviting administrators</td>
<td>170</td>
</tr>
<tr>
<td>Adding portal users</td>
<td>172</td>
</tr>
<tr>
<td>Administering your portals</td>
<td>173</td>
</tr>
<tr>
<td>Changing a portal's name, description, branding, support email, and permissions</td>
<td>173</td>
</tr>
<tr>
<td>Adding or removing portal administrators</td>
<td>174</td>
</tr>
<tr>
<td>Sending email invitations to portal administrators</td>
<td>175</td>
</tr>
<tr>
<td>Adding or removing portal users</td>
<td>175</td>
</tr>
<tr>
<td>Deleting a portal</td>
<td>176</td>
</tr>
<tr>
<td>Creating dashboards (CLI)</td>
<td>177</td>
</tr>
<tr>
<td>Querying asset property data</td>
<td>180</td>
</tr>
<tr>
<td>Querying current values</td>
<td>180</td>
</tr>
<tr>
<td>Querying historical values</td>
<td>181</td>
</tr>
<tr>
<td>Querying aggregated values</td>
<td>182</td>
</tr>
<tr>
<td>Interacting with other services</td>
<td>184</td>
</tr>
</tbody>
</table>
Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates.

Quotas ........................................................................................................................................ 240
Document history ..................................................................................................................... 243
AWS glossary ............................................................................................................................ 248
The AWS IoT SiteWise service is currently in preview. Under preview service terms and conditions, the service is subject to change.
What is AWS IoT SiteWise?

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

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

How AWS IoT SiteWise works

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

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

- Use AWS IoT SiteWise gateway software that runs on any platform that supports AWS IoT Greengrass, such as common industrial gateways or virtual servers. This software can read data directly from on-site servers over the OPC-UA protocol. You can connect up to 100 OPC-UA servers to a single AWS IoT SiteWise gateway. For more information, see Ingesting data using a gateway (p. 100).
- Use AWS IoT Core rules. If you have devices connected to AWS IoT Core sending MQTT messages, you can use the AWS IoT Core rules engine to route those messages to AWS IoT SiteWise. For more information, see Ingesting data using AWS IoT Core rules (p. 131).
- Use AWS IoT Events actions. You can configure the IoT SiteWise action in AWS IoT Events to send data to AWS IoT SiteWise when events occur. For more information, see Ingesting data from AWS IoT Events (p. 137).
- Use the AWS IoT SiteWise API. Your applications at the edge or in the cloud can directly send data to AWS IoT SiteWise. For more information, see Ingesting data using the AWS IoT SiteWise API (p. 137).

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

Benefits

Collect data consistently from all your sources

With AWS IoT SiteWise, you can gather data reliably from multiple facilities, structure it, and make it accessible and understandable without developing additional software. You can index information and metrics about equipment or processes across multiple facilities, so it's readily available for applications.

Identify issues quickly with remote monitoring

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

Improve cross-facility processes with a central data source

Visibility across industrial facilities lets you streamline operations, and identify gaps in production and waste. With AWS IoT SiteWise, you can create models of industrial processes and equipment across multiple facilities, and then visualize live and historical data through customizable charts and dashboards. Through SiteWise Monitor, you can launch private web applications with your asset data in minutes with AWS. You and your industrial engineers can use these web applications to better understand your operations, improve processes, and reduce waste across your entire organization.

Use cases

Manufacturing

Manufacturing companies employ complex processes on their factory floors. Quality assurance engineers and assembly robots inspect each product at various points in the assembly line, but this often involves manual work and can be subject to human error. AWS IoT SiteWise helps you collect data from manufacturing lines and assembly robots, transfer it to the AWS Cloud, and structure performance metrics for your specific equipment and processes. You can view production line output in SiteWise Monitor to assist in production planning and identify equipment and process deficiencies, production gaps, or product defects. You can also use these metrics to understand the overall effectiveness of your operations and identify opportunities for innovation and improvement.

Food and beverage

Food and beverage industry facilities handle a wide variety of food processing, including grinding grain to flour, butchering and packing meat, and assembling, cooking, and freezing microwaveable meals. These processing plants often span multiple locations with process engineers and equipment operators in a centralized location monitoring processes and equipment. For example, they might monitor refrigeration units, assess ingredient handling and expiration, or 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 that your engineers and operators can better understand and improve processes across facilities.

Energy and utilities

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

Are you new to AWS IoT SiteWise?

If you’re a first-time user of AWS IoT SiteWise, we recommend that you read about the components and concepts of AWS IoT SiteWise and set up the AWS IoT SiteWise demo (p. 7).

- Key components of AWS IoT SiteWise (p. 5)
- AWS IoT SiteWise concepts (p. 3)

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

- Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 42)
- Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 62)
- Publishing property value updates to Amazon DynamoDB (p. 85)

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

- Ingesting data to AWS IoT SiteWise (p. 100)
- Modeling industrial assets (p. 139)
- Monitoring data with AWS IoT SiteWise Monitor (p. 163)
- Querying asset property values and aggregates (p. 180)
- Interacting with other AWS services (p. 184)

AWS IoT SiteWise concepts

The following are the core concepts of AWS IoT SiteWise:

**Gateway**

A gateway connects to OPC-UA servers to deliver your industrial data streams to AWS IoT SiteWise. You can create a gateway on any device or platform that can run AWS IoT Greengrass. The gateway software exists as a connector that you can add to your AWS IoT Greengrass group. For more information, see Ingesting data using a gateway (p. 100).

**Asset**

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

**Asset model**

Every asset is created from an asset model. Asset models are declarative structures that standardize the format of your assets. Asset models enforce consistent information across multiple assets of the same type, so that you can process data in assets that represent groups of devices. In each asset model, you can define attributes (p. 4), time series inputs (measurements (p. 4)), time series
transformations (transforms (p. 4)), time series aggregations (metrics (p. 4)), and asset hierarchies (p. 4). For more information, see Modeling industrial assets (p. 139).

Asset property

Asset properties are the structures within each asset that contain industrial data. Each property has a data type and can have a unit. A property can be an attribute (p. 4), a measurement (p. 4), a transform (p. 4), or a metric (p. 4). For more information, see Asset properties (p. 142).

Attribute

Attributes are asset properties that represent information that generally doesn't change, such as device manufacturer or device location. Attributes can have default values. Each asset that you create from an asset model contains the default values of the attributes of that model. For more information, see Attributes (p. 142).

Measurement

Measurements are asset properties that represent a device or equipment's raw sensor time series data streams. For more information, see Measurements (p. 143).

Transform

Transforms are asset properties that represent transformed time series data. Every transform has a mathematical expression (formula (p. 4)) that defines how to transform data points from one form to another. The transformed data points hold a one-to-one relationship with the input data points. For more information, see Transforms (p. 143).

Metric

Metrics are asset properties that represent aggregated time series data. Every metric has a mathematical expression (formula (p. 4)) that defines how to aggregate data points, and a time interval over which to compute that aggregation. Metrics output a single data point per given time interval. For more information, see Metrics (p. 144).

Asset hierarchy

You can define asset hierarchies to create logical representations of your industrial operations. To create a hierarchy, you define a hierarchy definition in an asset model, and then you associate assets created from that model and the model specified in the hierarchy definition. Metrics in parent assets can aggregate data from child assets' properties, so you can calculate statistics that provide insight to your operation or a subset of your operation. For more information, see Asset hierarchies (p. 150).

Formula

Every transform (p. 4) and metric (p. 4) property has a formula that defines how that property transforms or aggregates data. Formulas consist of property inputs, operators, and functions offered by AWS IoT SiteWise. For more information, see Using formula expressions (p. 146).

Property alias

You can define aliases on asset properties to easily identify an asset property when you ingest or retrieve asset data. When you use a gateway (p. 3) to ingest data from OPC-UA servers, your property aliases must match the OPC-UA paths of your raw data streams. For more information, see Mapping industrial data streams to asset properties (p. 152).

Property notification

When you enable property notifications for an asset property, AWS IoT SiteWise publishes an MQTT message to AWS IoT Core each time that property receives a new value. The message payload contains information about that property value update. You can use property value notifications to create solutions that connect your industrial data in AWS IoT SiteWise with other AWS services. For more information, see Interacting with other AWS services (p. 184).
**portal**

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

**Portal administrator**

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

**project**

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

**Project owner**

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

**Project 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 time frames to better understand operational data. Project viewers can only view dashboards in the projects to which they have access.

**dashboard**

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

**visualization**

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

---

**Key components of AWS IoT SiteWise**

The following are the core components of AWS IoT SiteWise:

- **Gateway software** – The AWS IoT SiteWise connector runs on any platform that supports AWS IoT Greengrass (version 1.10.0 or later). The connector polls data streams over the OPC-UA protocol and uploads the data to AWS IoT SiteWise over a secured internet connection. The connector also caches data locally in case internet connectivity is interrupted. To learn how to configure the AWS IoT SiteWise connector, see Ingesting data using a gateway (p. 100).
• **Gateway management** – Tools that the AWS IoT SiteWise console provides so that you can configure your edge gateways and update existing gateway configurations. To learn how to configure gateways and sources, see *Adding the gateway to AWS IoT SiteWise* (p. 114).

• **Asset modeling** – Tools provided in the AWS IoT SiteWise console to model assets, processes, and facilities. You can create multiple assets from asset models and uploaded data streams. You can define assets as children of other assets to represent processes and entire facilities, and you can define transforms and metrics that AWS IoT SiteWise computes from the uploaded data streams. To learn how to create models and assets, see *Modeling industrial assets* (p. 139).

• **AWS IoT SiteWise Monitor** – A data visualization and monitoring web application that you and others can access from any device. You can configure portals and dashboards with unique subsets of your industrial operation for your non-technical employees to gain insights from. To learn how to configure SiteWise Monitor, see *Monitoring data with AWS IoT SiteWise Monitor* (p. 163). To learn how to use the SiteWise Monitor web application, see the *SiteWise Monitor Application Guide*.

• **AWS IoT Core integration** – An AWS IoT Core rules action to ingest data to AWS IoT SiteWise and an option to publish asset property values to MQTT topics for integration with other AWS services. To learn how to ingest data from AWS IoT Core, see *Ingesting data using AWS IoT Core rules* (p. 131). To learn how to send asset property data to AWS IoT Core, see *Interacting with other AWS services* (p. 184).

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

---

**We want to hear from you**

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

• **Provide feedback** at the bottom of the page.

• **Feedback** at the top right of the page.
Getting started with AWS IoT SiteWise

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

AWS IoT SiteWise provides a demo that you can use to explore the service without configuring a real data source. For more information, see Using the AWS IoT SiteWise demo (p. 8).

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

- Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 42)
- Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 62)
- Publishing property value updates to Amazon DynamoDB (p. 85)

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

- Ingesting data to AWS IoT SiteWise (p. 100)
- Modeling industrial assets (p. 139)
- Monitoring data with AWS IoT SiteWise Monitor (p. 163)
- Querying asset property values and aggregates (p. 180)
- Interacting with other AWS services (p. 184)

Requirements

You must have an AWS account to get started with AWS IoT SiteWise. If you don't have one, see Setting up an AWS account (p. 7).

AWS IoT SiteWise is available in the following AWS Regions:

- US East (N. Virginia) – us-east-1
- US West (Oregon) – us-west-2
- Europe (Frankfurt) – eu-central-1
- Europe (Ireland) – eu-west-1

Use the Region selector in the AWS Management Console to switch to one of these Regions.

Setting up an AWS account

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

To sign up for an AWS account

2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.
To create an administrator user for yourself and add the user to an administrators group
(console)

1. Use your AWS account email address and password to sign in as the AWS account root user
to the IAM console at https://console.aws.amazon.com/iam/.
   
   **Note**
   We strongly recommend that you adhere to the best practice of using the Administrator IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

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

3. For User name, enter Administrator.

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

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

6. Choose Next: Permissions.

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

8. Choose Create group.

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

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

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

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

13. Choose Next: Tags.

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

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

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

Using the AWS IoT SiteWise demo

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

**Important**
You will be charged for the resources that this demo creates and consumes. We estimate that running the demo for a week costs around $1.50 USD. The demo shouldn't cost you anything if you have sufficient remaining usage on your AWS Free Tier.

**Topics**
Creating the AWS IoT SiteWise demo

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

**Note**

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

**To create the demo in the AWS IoT SiteWise console**

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

   The demo takes around 3 minutes to create. If the demo fails to create, your account might have insufficient permissions. Switch to an account that has administrative permissions, or use the following steps to delete the demo and try again:
   - Choose Delete demo.
   - The demo takes around 15 minutes to delete.
   - If the demo doesn't delete, open the AWS CloudFormation console, choose the stack named IoTSiteWiseDemoAssets, and choose Delete in the upper-right corner.
   - If the demo fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.
4. After the demo creates successfully, you can explore the demo assets and data in the AWS IoT SiteWise console.

**To create the demo in AWS CloudFormation**

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

   The stack takes around 3 minutes to create. If the stack fails to create, your account might have insufficient permissions. Switch to an account that has administrative permissions, or use the following steps to delete the demo and try again:
   - Choose Delete in the upper-right corner.
   - The stack takes around 15 minutes to delete.
b. If the demo fails to delete, choose Delete in the upper right corner again.

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.

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

Deleting the AWS IoT SiteWise demo

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

To delete the AWS IoT SiteWise demo

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

   When you delete the stack, all of the resources created for the demo are deleted.

4. In the confirmation dialog, choose Delete stack.

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

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

Topics

• Calculating OEE in AWS IoT SiteWise (p. 11)
• Configuring alarms for asset property values with AWS IoT Events (p. 13)
• Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 42)
• Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 62)
• Publishing property value updates to Amazon DynamoDB (p. 85)

Calculating OEE in AWS IoT SiteWise

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

Prerequisites

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

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

To configure data ingestion, see Ingesting data to AWS IoT SiteWise (p. 100). If you don't have an available industrial operation, you can write a script that generates and uploads sample data through the AWS IoT SiteWise API.

How to calculate OEE

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

Define the following measurements (p. 143) to represent the raw data streams from the packaging machine.
Measurements

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

Using the `Equipment_State` measurement data stream and the codes it contains, define the following transforms (p. 143) (or derived measurements). Transforms have a one-to-one relationship with raw measurements.

Transforms

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

Using the raw measurements and the transformed measurements, define the following metrics (p. 144) 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** = \( \text{sum}(\text{Good}_\text{Count}) \) – The number of successfully filled packages over the specified time interval.
- **Failures** = \( \text{sum}(\text{Bad}_\text{Count}) \) – The number of unsuccessfully filled packages over the specified time interval.
- **Idle_Time** = \( \text{statetime}(\text{Idle}) \) – The machine's total idle time (in seconds) per specified time interval.
- **Fault_Time** = \( \text{statetime}(\text{Fault}) \) – The machine's total fault time (in seconds) per specified time interval.
- **Stop_Time** = \( \text{statetime}(\text{Stop}) \) – The machine's total planned stop time (in seconds) per specified time interval.
- **Run_Time** = \( \text{statetime}(\text{Running}) \) – The machine's total time (in seconds) running without issue per specified time interval.
- **Down_Time** = Idle_Time + Fault_Time + Stop_Time – The machine's total downtime (in seconds) over the specified time interval, calculated as the sum of the machine states other than Run_Time.
- **Availability** = Run_Time / (Run_Time + Down_Time) – The machine's uptime or percentage of scheduled time that the machine is available to operate over the specified time interval.
• **Quality** = \( \frac{\text{Successes}}{\text{Successes} + \text{Failures}} \) – The machine's percentage of successfully filled packages over the specified time intervals.

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

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

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

### Configuring alarms for asset property values with AWS IoT Events

You can configure alarms for your data to send alert notifications to you and your team when your 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. Alarms let you quickly identify issues and notify operators to maximize performance of your equipment and processes.

You can use AWS IoT Events to build complex event monitoring applications, such as an alarm detection system, in the AWS Cloud. Then, you can stream asset data from AWS IoT SiteWise to AWS IoT Events to trigger alarm events for your data. For more information, see What is AWS IoT Events? in the AWS IoT Events Developer Guide.

AWS IoT Events lets you send a message to Amazon Simple Notification Service when an event occurs. You can create subscriptions in Amazon SNS to configure who receives notifications for your alarms. For more information, see What is Amazon SNS? in the Amazon Simple Notification Service Developer Guide.

In this tutorial, you configure resources in AWS IoT Events, Amazon SNS, and AWS Lambda to create a reusable alarm detection system in the AWS Cloud. Then, you learn how to use this system to configure alarms for your asset properties in AWS IoT SiteWise.

### Topics

- **Prerequisites** (p. 13)
- Configuring an alarm detection system (p. 14)
- Configuring alarms (p. 33)

### Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Setting up an AWS account (p. 7).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- Python 3 installed on your computer.
- AWS SDK for Python (Boto 3) installed on your computer and configured with your AWS authentication credentials. This tutorial requires version 1.12.49 or later of the SDK for Python (Boto 3). For more information, see Quickstart in the AWS SDK for Python (Boto 3) Getting Started.
Configuring an alarm detection system

You can create an alarm detection system that lets you easily configure alarms for your industrial data. You need to set up the alarm detection system only once in each Region that you use AWS IoT SiteWise. Then, you can configure alarms for your asset properties to receive an alert when data values exceed low or high thresholds. For more information, see Configuring alarms (p. 33).

**Important**
Before you can configure an alarm detection system, you must meet the prerequisites for this tutorial. For more information, see Prerequisites (p. 13).

**Topics**
- Creating an Amazon SNS topic for alert messages (p. 14)
- Creating inputs for your alarm detection model in AWS IoT Events (p. 15)
- Creating an alarm detection model in AWS IoT Events (p. 18)
- Creating a Lambda function to send asset property values to AWS IoT Events (p. 28)

Creating an Amazon SNS topic for alert messages

With Amazon SNS, you can publish messages to topics. Then, you can create subscriptions to receive notifications when a message is published to a topic. For this alarm detection system, you create a topic that receives alarm messages from AWS IoT Events. Then, you subscribe to the topic when you configure alarms.

**To create an Amazon SNS topic**

1. Navigate to the Amazon SNS console.
2. Review the AWS Regions (p. 7) where AWS IoT SiteWise and AWS IoT Events are supported and switch Regions, if needed.
3. In the left navigation pane, choose Topics.
4. Choose Create topic.
5. On the Create topic page, do the following:
   a. In Name, enter IoTSiteWiseAlarmTopic.
   b. (Optional) Enter a Display name for your topic, such as IoT SiteWise Alarm. If you configure email subscriptions, this value is the name of the sender in the emails that you receive. You must complete this step if you configure SMS subscriptions for this topic.
   c. Choose Create topic.
6. Copy the ARN of your new topic. You need this ARN later when you create an alarm detector model in AWS IoT Events.

Creating inputs for your alarm detection model in AWS IoT Events

In AWS IoT Events, inputs define the shape of data messages that detector models consume and act on. All inputs used in a detector model must share a common key, so the detector model can associate messages from different inputs. This alarm detection system uses the following three inputs that share a common key, `alarmId`, which is the concatenation of asset ID and property ID for an asset property:

- **AlarmDefinition** – Defines an alarm with the following properties:
  - `alarmId` – The ID of the alarm, defined as the concatenation of `assetId` and `propertyId`.
  - `assetModelId` – The ID of the asset’s model.
  - `assetModelName` – The name of the asset’s model.
  - `assetId` – The ID of the asset.
  - `assetName` – The name of the asset.
  - `propertyId` – The ID of the property.
  - `propertyName` – The name of the property.
  - `threshold` – A structure that defines the following threshold values for the alarm:
    - `high` – The upper threshold value for the alarm. The alarm triggers if asset property values are higher than this value.
    - `low` – The lower threshold value for the alarm. The alarm triggers if asset property values are lower than this value.
  - `thresholdCount` – The number of data points that exceed a threshold before the alarm triggers.
- **AlarmPropertyValue** – Contains an asset property value with the following properties:
  - `alarmId` – The ID of the alarm.
  - `propertyValue` – A data point for the asset property configured in this alarm.
- **AlarmStatus** – Contains the status of an alarm:
  - `alarmId` – The ID of the alarm.
  - `status` – The status of the alarm: **ON** or **OFF**.

In this procedure, you create these inputs from sample messages that define the shape of each input.
To create inputs for the alarm detection system

1. Navigate to the AWS IoT Events console.
2. In the left navigation pane, choose Inputs.
3. To create an input for alarm definitions, do the following:
   a. Create a file called `alarm_definition_message.json` and copy the following JSON object into the file.

   ```json
   {
   "assetModelId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
   "assetModelName": "Fabricator Model",
   "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
   "assetName": "Fabricator 3",
   "propertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
   "propertyName": "Nozzle Temperature",
   "alarmId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE-a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
   "threshold": {
   "high": 260,
   "low": 230
   },
   "thresholdCount": 5
   }
   ```
   
   b. Choose Create input.
   c. In Input name, enter AlarmDefinition.
   d. (Optional) Enter a description to describe this input.
   e. Choose Upload file, and upload the file `alarm_definition_message.json`.
   f. Choose Create.
4. To create an input for asset property values in alarms, do the following:

   a. Create a file called `alarm_property_value_message.json` and copy the following JSON object into the file.

```
{
```

Configuring an alarm detection system

1. Create a file called `alarm_property_value_message.json` and copy the following JSON object into the file.

   ```json
   {
     "alarmId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
     "propertyValue": 241
   }
   ```

   b. Choose Create input.
   c. In Input name, enter AlarmPropertyValue.
   d. (Optional) Enter a description to describe this input.
   e. Choose Upload file, and upload the file `alarm_property_value_message.json`.
   f. Choose Create.

2. To create an input for alarm status, do the following:
   a. Create a file called `alarm_status_message.json` and copy the following JSON object into the file.

   ```json
   {
     "alarmId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
     "status": "OFF"
   }
   ```

   b. Choose Create input.
   c. In Input name, enter AlarmStatus.
   d. (Optional) Enter a description to describe this input.
   e. Choose Upload file, and upload the file `alarm_status_message.json`.
   f. Choose Create.

You should have three inputs in AWS IoT Events.

Creating an alarm detection model in AWS IoT Events

Detector models define how AWS IoT Events processes data from inputs. In the alarm detection system, AWS IoT Events creates a detector from the detector model for each alarm. Each detector has a state, and this state changes when certain conditions are met. When data exceeds an alarm threshold, the detector transitions to an alarm state and sends an alert to Amazon SNS. For more information, see Create a detector model in the AWS IoT Events Developer Guide.

To create a detector model for alarms

1. Create a file called `AlarmThresholdMonitor.json` and copy the following JSON object into the file. Replace the three instances of `sns targetArn` with the ARN of the Amazon SNS topic you created earlier.
```json
{
    "detectorModelDefinition": {
        "states": [
        {
            "stateName": "VALUE_TOO_LOW",
            "onInput": {
                "events": [
                {
                    "eventName": "normalValueCheck",
                    "condition": "($input.AlarmPropertyValue.propertyValue < $input.AlarmDefinition.threshold.high) && ($input.AlarmPropertyValue.propertyValue > $input.AlarmDefinition.threshold.low)",
                    "actions": [
                    {
                        "setVariable": {
                            "variableName": "breachLowCount",
                            "value": "$variable.breachLowCount - 1"
                        }
                    },
                    {
                        "setVariable": {
                            "variableName": "returnToNormal",
                            "value": "1"
                        }
                    }
                ]
            }
        },
        {
            "eventName": "highValueCheck",
            "condition": "$input.AlarmPropertyValue.propertyValue >= $input.AlarmDefinition.threshold.high",
            "actions": [
            {
                "setVariable": {
                    "variableName": "breachHighCount",
                    "value": "$variable.breachHighCount + 1"
                }
            },
            {
                "setVariable": {
                    "variableName": "receivedHighValue",
                    "value": "1"
                }
            }
            ]
        }
        ],
        "transitionEvents": [
        {
            "eventName": "returnToNormal",
            "condition": "($variable.breachLowCount == 0) || (!$variable.receivedHighValue == 1)"
        },
        {
            "eventName": "turnOff",
            "condition": "#$input.AlarmStatus.status == \"OFF\"",
            "actions": [],
            "nextState": "OFF"
        }
        ]
    }
},
    "onEnter": {
```
"events": [
  {
    "eventName": "sendAlarmLowMessage",
    "condition": "true",
    "actions": [
      {
        "sns": {
          "payload": {
            "type": "STRING"
          }
        }
      }
    ]
  },
  {
    "eventName": "initializeVariable",
    "condition": "true",
    "actions": [
      {
        "setVariable": {
          "variableName": "receivedHighValue",
          "value": "0"
        }
      },
      {
        "setVariable": {
          "variableName": "returnToNormal",
          "value": "0"
        }
      }
    ]
  }
],
"onExit": {
  "events": []
}
},
{
  "stateName": "VALUE_TOO_HIGH",
  "onInput": {
    "events": [
      {
        "eventName": "normalValueCheck",
      },
      {
        "setVariable": {
          "variableName": "breachHighCount",
          "value": "#variable.breachHighCount - 1"
        }
      },
      {
        "setVariable": {
          "variableName": "returnToNormal",
          "value": "0"
        }
      }
    ]
  }
}
"value": "1"
}
},
{
  "eventName": "lowValueCheck",
  "condition": "\$input.AlarmPropertyValue.propertyValue <= $input.AlarmDefinition.threshold.low",
  "actions": [
    {
      "setVariable": {
        "variableName": "breachLowCount",
        "value": \"$\variable.breachLowCount + 1\"
      }
    },
    {
      "setVariable": {
        "variableName": "receivedLowValue",
        "value": "1"
      }
    }
  ]
},
"transitionEvents": [
  {
    "eventName": "returnToNormal",
    "condition": "($\variable.breachHighCount == 0) || ($\variable.receivedLowValue == 1)"
  },
  {
    "eventName": "turnOff",
    "condition": "$\input.AlarmStatus.status == \"OFF\"\",
    "actions": [],
    "nextState": "OFF"
  }
],
"onEnter": {
  "events": [
    {
      "eventName": "sendAlarmHighMessage",
      "condition": "true",
      "actions": [
        {
          "sns": {
            "payload": {
              "type": "STRING"
            }
          }
        }
      ]
    }
  ]
}
"eventName": "initializeVariable",
"condition": "true",
"actions": [  
  
  ]

"setVariable": {  
  "variableName": "returnToNormal",
  "value": "0"
  
  },  

"setVariable": {  
  "variableName": "receivedLowValue",
  "value": "0"
  
  }

]  

"onExit": {  
  "events": []
  
  }

{"  
  "stateName": "ALARM_NOT_DEFINED",
  "onInput": {  
    "events": [],
    "transitionEvents": [  
    
    ]
    
  },  

  "onEnter": {  
    "events": []
    
  },  

  "onExit": {  
    "events": []
    
  }

"stateName": "OFF",
"onInput": {  
  "events": [],
  "transitionEvents": [  
  
  ]
  
  },  

  "onEnter": {  
    "events": []
    
  },  

  "onExit": {  
    "events": []
    
  }

"stateName": "NORMAL",
"onInput": {  
  
  }}
"events": [ 
  {
    "eventName": "setVariables",
    "condition": "currentInput("AlarmPropertyValue")",
    "actions": [
      {
        "setVariable": {
          "variableName": "countValues",
          "value": "#variable.countValues + 1"
        }
      }
    ]
  },
  {
    "eventName": "checkHighThresholdBreach",
    "condition": "$input.AlarmPropertyValue.propertyValue >= $input.AlarmDefinition.threshold.high",
    "actions": [
      {
        "setVariable": {
          "variableName": "breachHighCount",
          "value": "#variable.breachHighCount + 1"
        }
      }
    ]
  },
  {
    "eventName": "checkLowThresholdBreach",
    "condition": "$input.AlarmPropertyValue.propertyValue <= $input.AlarmDefinition.threshold.low",
    "actions": [
      {
        "setVariable": {
          "variableName": "breachLowCount",
          "value": "#variable.breachLowCount + 1"
        }
      }
    ]
  },
  {
    "eventName": "checkNoThresholdBreach",
    "condition": "($input.AlarmPropertyValue.propertyValue < $input.AlarmDefinition.threshold.high) && ($input.AlarmPropertyValue.propertyValue > $input.AlarmDefinition.threshold.low)",
    "actions": [
      {
        "setVariable": {
          "variableName": "breachHighCount",
          "value": "0"
        }
      },
      {
        "setVariable": {
          "variableName": "breachLowCount",
          "value": "0"
        }
      }
    ]
  }
],
"transitionEvents": [
  {
    "eventName": "AlarmHigh",
    "condition": "#variable.breachHighCount >= $input.AlarmDefinition.thresholdCount",
    "actions": []
  }
]
"nextState": "VALUE_TOO_HIGH"
},
{
  "eventName": "AlarmLow",
  "condition": "#variable.breachLowCount >= $input.AlarmDefinition.thresholdCount",
  "actions": [],
  "nextState": "VALUE_TOO_LOW"
},
{
  "eventName": "turnOff",
  "condition": "$input.AlarmStatus.status == "OFF"",
  "actions": [],
  "nextState": "OFF"
}
]
}
"onEnter": {
  "events": [
    {
      "eventName": "initializeVariables-1",
      "condition": "true",
      "actions": [
        {
          "setVariable": {
            "variableName": "breachLowCount",
            "value": "0"
          }
        },
        {
          "setVariable": {
            "variableName": "breachHighCount",
            "value": "0"
          }
        },
        {
          "setVariable": {
            "variableName": "countValues",
            "value": "0"
          }
        }
      ]
    },
    {
      "eventName": "normalcyCheck",
      "condition": "#variable.returnToNormal == 1",
      "actions": [
        {
          "sns": {
            "payload": {
              "contentExpression": "An asset property has returned to normal.
Asset: $input.AlarmDefinition.assetName
Property: $input.AlarmDefinition.propertyName
Latest value: ${$input.AlarmPropertyValue.propertyValue}
Low threshold: ${$input.AlarmDefinition.threshold.low}
High threshold: ${$input.AlarmDefinition.threshold.high}
Threshold count: ${$input.AlarmDefinition.thresholdCount}",
              "type": "STRING"
            }
          }
        }
      ]
    }
  ]
}
Note

This JSON object doesn’t include `roleArn` for a service role that AWS IoT Events assumes to execute detectors. In this tutorial, AWS IoT Events creates a role for you when you import and publish the detector model in the AWS Management Console. If you instead use the AWS CLI to create the detector model, `roleArn` is required. For more information about how to define a service role for AWS IoT Events, see Setting up permissions for AWS IoT Events in the AWS IoT Events Developer Guide.

2. Navigate to the AWS IoT Events console.
3. In the left navigation pane, choose Detector models.
4. Choose Action, and then choose Import detector model from the list.
5. In the Import detector model dialog, choose Import, and then choose the AlarmThresholdMonitor.json file.

AWS IoT Events imports the detector model and displays it in the visual editor.
6. Choose **Publish** in the upper right.

7. In the **Publish detector model** dialog, do the following:
   
   a. (Optional) Edit the **Detector model name** or **Description**.
   
   b. In **Role**, enter a name for a new service role that AWS IoT Events creates for you, such as **IoTSiteWiseAlarmRole**. AWS IoT Events requires permissions to send messages to the Amazon SNS topic that you created earlier. AWS IoT Events assumes this role when it executes the detectors for this model.
   
   c. For **Detector evaluation method**, choose **Serial evaluation**.
d. **Choose Save and publish.**

Your alarm detector model publishes.
Creating a Lambda function to send asset property values to AWS IoT Events

In AWS IoT SiteWise you can enable asset property value notifications to send an MQTT message to AWS IoT Core for every new value received. In this procedure, you create a Lambda function that parses this message and sends property value updates to an input in AWS IoT Events. When AWS IoT Events receives the data, it sends the value through an alarm detector to check if the values exceed alarm thresholds.

To create a Lambda function for the alarm detection system

1. Navigate to the Lambda console.
2. Choose Create function.
3. On the Create function page, do the following:
   a. Choose Author from scratch.
   b. In Function name, enter IoTSiteWiseAlarmPayloadConverter.
   c. For Runtime, choose Python 3.7.
   d. For Execution role, choose Create a new role with basic Lambda permissions.
   e. Choose Create function.
The page for your new function opens.

4. In the inline code editor under **Function code**, paste the following Lambda function code.

```python
import boto3
import json
import uuid

# Flattens messages produced by AWS IoT SiteWise to feed to AWS IoT Events.
def lambda_handler(event, context):
    print("Received raw message from AWS IoT Core.")

    messages = []
    # Flatten the list of values from AWS IoT SiteWise into messages for AWS IoT Events.
    for value in event["values"]:
        # Include only good quality values.
        if value["quality"] == "GOOD":
            payload = {
                "alarmId": event["alarmId"]
            }

            if "doubleValue" in value["value"]:  
                payload["propertyValue"] = float(value["value"])["doubleValue"]
            elif "integerValue" in value["value"]:  
                payload["propertyValue"] = int(value["value"])["integerValue"]
            else:
                continue  # Filter out non-numeric values.

            message = {
                "messageId": str(uuid.uuid4()),
                "inputName": "AlarmPropertyValue",
                "payload": json.dumps(payload)
            }

            messages.append(message)

    # Send the flattened messages to AWS IoT Events.
iote_client = boto3.client("iotevents-data")
    response = iote_client.batch_put_message(messages=messages)
    status = int(response["ResponseMetadata"]["HTTPStatusCode"])

    if status == 200:
        print("Successfully wrote %d values to AWS IoT Events.\n" % (len(messages)))
    else:
        print("Failed to write values to AWS IoT Events. Status code: %d\n" % (status))
```

5. Choose **Save**.

Lambda requires permissions to send messages to AWS IoT Events when the function runs. In this procedure, you add the required permissions to the function’s role so that Lambda can successfully run the function.
To allow Lambda to send messages to AWS IoT Events when the function runs

1. On the function's page, choose the Permissions tab.

2. Choose the Role name, such as IoTSiteWiseAlarmPayloadConverter-role-r98v76r9, to open the role in the IAM console.

3. On the page for the Lambda execution role, choose Attach policies.

4. Choose Create policy.

   The Create policy page opens in a new tab.

5. On the Create policy page, do the following:

   a. Choose the JSON tab to open the JSON editor.
   b. Enter the following policy that allows the role to send messages to AWS IoT Events. Replace \textit{region} and \textit{account-id} with your Region and AWS account ID.

   
   
   ```json
   {   "Version": "2012-10-17",   "Statement": [   {   "Sid": "PutAlarmPropertyValues",   ```
c. Choose **Review policy**.

6. On the **Review policy** page, do the following:
   
a. In **Name** enter **IoTSiteWiseAlarmPropertyValuePolicy**.
   
b. (Optional) Enter a description for the policy.
   
c. Choose **Create policy**.
Your policy is created but not added to the role.

7. Return to the browser tab with IAM open to add permissions to the function's execution role.

8. Choose the box for the new policy, `IoTSiteWiseAlarmPropertyValuePolicy`. You might need to filter or refresh the policies table to find the policy.

9. Choose Attach policy.

The policy attaches to the Lambda function's execution role.
Your alarm detection system is configured. You can now configure alarms for your asset properties in AWS IoT SiteWise. For more information, see Configuring alarms (p. 33).

**Configuring alarms**

After you configure the alarm detection system, you can configure alarms for asset properties in AWS IoT SiteWise. To configure an alarm, you must enable property value notifications in AWS IoT SiteWise. Then, you configure AWS IoT rules to send property data to the alarm detection system’s AWS Lambda function. The Lambda function sends the data to AWS IoT Events, which checks if an alarm should trigger.

To define an alarm for an asset property, you send a message to AWS IoT Events that contains an alarm definition. You specify high and low thresholds for an asset property, and the number of data points after which the alarm triggers. You can also send a message to AWS IoT Events to turn an alarm on or off. Use the scripts in this tutorial to easily create and send these messages to inputs in AWS IoT Events.

**Important**

Before you can configure alarms, you must meet the prerequisites for this tutorial and configure an alarm detection system. For more information, see Prerequisites (p. 13) and Configuring an alarm detection system (p. 14).

**Topics**

- Enabling asset property notifications in AWS IoT SiteWise (p. 33)
- Creating an AWS IoT rule that acts on AWS IoT SiteWise notifications (p. 34)
- Defining an alarm for an asset property (p. 36)
- Subscribing to Amazon SNS topics to receive alerts (p. 38)
- Enabling and disabling alarms (p. 40)

**Enabling asset property notifications in AWS IoT SiteWise**

To send property value updates from AWS IoT SiteWise to AWS IoT Events, you must enable asset property notifications for each property for which you define an alarm. Then, each time a property receives or computes a new data point, AWS IoT SiteWise sends an MQTT message to AWS IoT Core that contains the new value. Later in this tutorial, you configure an AWS IoT rule to send the property value update to AWS IoT Events through the Lambda function that you created earlier. For more information, see Interacting with other AWS services (p. 184).

**To enable asset property notifications**

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Assets**.
3. Choose the asset for which you want to configure an alarm.
4. Choose Edit.
5. Find the property for which you want to configure an alarm, and then choose ENABLED for its Notification status.

6. Choose Save.
7. On the page for the asset, find the property for which you enabled notifications.
8. Choose the copy icon next the notification topic. AWS IoT SiteWise publishes MQTT messages to this topic for this property. You use this notification topic later in this tutorial.

The notification topic should look like the following example.

```
/aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
```

Creating an AWS IoT rule that acts on AWS IoT SiteWise notifications

AWS IoT rules query and act on MQTT messages. In this procedure, you create an AWS IoT rule that acts on asset property value updates from AWS IoT SiteWise. The rule sends values to the Lambda function, which sends messages to the property value input in AWS IoT Events.

To create an AWS IoT rule with a Lambda action

1. Navigate to the AWS IoT console. If a Get started button appears, choose it.
2. In the left navigation pane, choose Act, and then choose Rules.
3. If a You don't have any rules yet dialog box appears, choose Create a rule. Otherwise, choose Create.
4. On the Create a rule page, do the following:
   a. Enter a Name for your rule.
   b. (Optional) Enter a Description for your rule.
   c. In Rule query statement, enter the following SQL statement. Replace the FROM topic with the property value notification topic for your asset property.

   ```
   SELECT
   concat(topic(6), "-", topic(8)) as alarmId,
   payload.values as values
   FROM
   ```
This query statement uses `topic(index)` to parse asset ID and property ID from the topic. Then, it concats asset ID and property ID to form the alarm ID key for AWS IoT Events.

**Note**
This rule query statement acts on messages from only one asset property. If you want to configure alarms for a specific property for all assets with that property, you can modify the `FROM` clause to reuse this AWS IoT rule for other alarms. To do so, update the `FROM` topic to the following topic filter.

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

The `+` character is a single-level wildcard. In this topic filter, the `+` character matches all assets. For more information, see Topics in the *AWS IoT Core Developer Guide*.

d. Under **Set one or more actions**, choose **Add action**.

e. On the **Select an action** page, choose **Send a message to a Lambda function**, and then choose **Configure action**.

f. On the **Configure action** page, do the following:

i. In **Function name**, choose **Select**, and then choose the Lambda function that you created earlier, `IoTSiteWiseAlarmPayloadConverter`.

ii. Choose **Add action**.

g. Choose **Create rule**.
Defining an alarm for an asset property

To define an alarm in AWS IoT Events, you can send a message to the alarm definition input. This message must contain the asset property information and the thresholds for the alarm. Each alarm has a high and a low value threshold. Each alarm also has a threshold count, which defines how many data points must exceed the threshold before the alarm triggers. You can increase the threshold count to avoid false alarms if your sensor readings bounce often. For more information about the structure of the alarm definition input, see Creating inputs for your alarm detection model in AWS IoT Events (p. 15).

In this procedure, you download and run a script that creates and sends the alarm definition message for you. This script inputs an asset ID, property ID, and threshold configuration. Then, it fetches additional asset property information from AWS IoT SiteWise to complete the message that the input expects.

Before you can run the script, you must configure your AWS credentials on your local machine. The script uses these credentials to authenticate with AWS. You must also install version 1.12.49 or later of the SDK for Python (Boto 3). Previous versions don’t include AWS IoT SiteWise APIs that this script uses. For more information, see Prerequisites (p. 13).

You can also run this script to update the threshold configuration for an existing alarm.

To define or update an alarm

1. Create a file called define_alarm.py and copy the following Python code into the file.

```python
# Define an alarm detector for an asset property in AWS IoT SiteWise.
import argparse
import boto3
import json
import sys
import uuid

# Configures the argument parser for this program.
def configureArgumentParser():
    parser = argparse.ArgumentParser()
    parser.add_argument("-a", "--asset-id", action="store",
                        dest="asset_id", required=True, help="The ID of the asset.")
    parser.add_argument("-p", "--property-id", action="store",
                        dest="property_id", required=True, help="The ID of the property.")
    parser.add_argument("-ht", "--threshold-high", action="store",
                        dest="threshold_high", required=True, type=float, help="The upper threshold value.")
    parser.add_argument("-lt", "--threshold-low", action="store",
                        dest="threshold_low", required=True, type=float, help="The lower threshold value.")
    parser.add_argument("-tc", "--threshold-count", action="store",
                        dest="threshold_count", required=True, type=int, help="The number of data points that exceed a threshold before the alarm triggers.")
    parser.add_argument("-c", "--profile", action="store", dest="profile",
                        help="The AWS credentials profile to use. Defaults to the 'default' profile.")
    parser.add_argument("-r", "--region", action="store", dest="region",
                        help="The AWS Region to use. Defaults to the Region configured for the chosen profile.")
    return parser

# Fetches asset, model, and property names and IDs for an asset property.
def fetch_asset_property_details(session, asset_id, property_id):
    iot_sitewise_client = session.client("iotsitewise")
```

```python
```

```python
```
# Fetch the asset details.
try:
    describe_asset_response = iot_sitewise_client.describe_asset(assetId=asset_id)
except iot_sitewise_client.exceptions.ResourceNotFoundException:
    raise Exception("Asset not found for asset ID: %s" % asset_id)

# Find the property name from the asset details.
property_name = None
for asset_property in describe_asset_response["assetProperties"]:  
    if asset_property["id"] == property_id:  
        property_name = asset_property["name"]
if property_name is None:
    raise Exception("Property not found for property ID: %s" % property_id)

# Fetch the asset model details.
asset_model_id = describe_asset_response["assetModelId"]
describe_asset_model = iot_sitewise_client.describe_asset_model(assetModelId=asset_model_id)
return {
    "assetModelId": asset_model_id,
    "assetModelName": describe_asset_model["assetModelName"],
    "assetId": asset_id,
    "assetName": describe_asset_response["assetName"],
    "propertyId": property_id,
    "propertyName": property_name
}

# Sends an alarm definition message to AWS IoT Events for an asset property and thresholds.
def send_alarm_definition_message(session, asset_id, property_id, threshold_high, threshold_low, threshold_count):
    asset_property = fetch_asset_property_details(session, asset_id, property_id)
    iot_events_client = session.client("iotevents-data")
    # Create the alarm definition.
    payload = {
        "alarmId": "%s-%s" % (asset_property["assetId"], asset_property["propertyId"])
        "threshold": {
            "high": threshold_high,
            "low": threshold_low
        },
        "thresholdCount": threshold_count
    }
    # Add the asset property details to the alarm definition.
    payload.update(asset_property)
    # Send the alarm definition to AWS IoT Events.
    message = {
        "messageId": str(uuid.uuid4()),
        "inputName": "AlarmDefinition",
        "payload": json.dumps(payload)
    }
    print("Sending alarm definition message to AWS IoT Events:")
    print(message)
    response = iot_events_client.batch_put_message(messages=[message])
    print("Status code: %d" % int(response["ResponseMetadata["HTTPStatusCode"]))

# Sends an alarm definition message with user arguments.
if __name__ == "__main__":
    parser = configureArgumentParser()
    args = parser.parse_args()
    try:
        session = boto3.Session(profile_name=args.profile, region_name=args.region)
AWS IoT SiteWise User Guide
Configuring alarms

```python
send_alarm_definition_message(session, args.asset_id, args.property_id,
    args.threshold_high, args.threshold_low,
    args.threshold_count)
except Exception as e:
    print(str(e), file=sys.stderr)
exit(1)
```

2. Run `define_alarm.py` from the command line with the following parameters:
   - `-a, --asset-id` – The ID of the asset.
   - `-p, --property-id` – The ID of the property.
   - `-ht, --threshold-high` – The upper threshold value. The alarm triggers if asset property values are higher than this value.
   - `-lt, --threshold-low` – The lower threshold value. The alarm triggers if asset property values are lower than this value.
   - `-tc, --threshold-count` – The number of data points that exceed a threshold before the alarm triggers.
   - `-c, --profile` – (Optional) The AWS credentials profile to use. If you omit this parameter, the script uses the default profile. For more information, see Credentials in the AWS SDK for Python (Boto 3) Getting Started.
   - `-r, --region` – (Optional) The Region to use. Specify the Region in which your assets and alarm detection system resides. If you omit this parameter, the script uses the Region configured for the chosen AWS profile.

Your command should look similar to the following example.

```
python3 define_alarm.py \
    -r us-east-1 \
    -a a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
    -p a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
    -ht 260 \
    -lt 230 \
    -tc 5
```

The script outputs the message payload and the result of the API request. The request succeeded if the output ends with **Status code: 200**.

Subscribing to Amazon SNS topics to receive alerts

To receive alarm alert messages, you must subscribe to the Amazon SNS topic that you created earlier in this tutorial. You can create subscriptions to various types of endpoints, including email, SMS, and HTTP/S. In this procedure, you create an email subscription to receive an email when your alarm triggers.

**To subscribe to an Amazon SNS topic**

1. Navigate to the Amazon SNS console.
2. In the left navigation pane, choose Topics.
3. Choose your topic, **IoTSiteWiseAlarmTopic**.
4. On the page for your new topic, choose Create subscription.
5. On the **Create subscription** page, do the following:
   a. If **Topic ARN** doesn't contain the ARN of your topic, choose your topic's ARN from the list.
   b. For **Protocol**, choose **Email**.
   c. In **Endpoint**, enter your email address. You must confirm your email address after you create the subscription.
   d. Choose **Create subscription**.
6. You’ll receive an email that asks you to confirm your subscription. In the email, choose **Confirm subscription**.

After you confirm your subscription, you should see a page that looks like the following screenshot.

---

**Enabling and disabling alarms**

The alarm detection system in this tutorial lets you turn an alarm on or off. For example, you can use this capability to disable an alarm while you calibrate a sensor. For more information about the structure of the alarm status input, see Creating inputs for your alarm detection model in AWS IoT Events (p. 15).

In this procedure, you download and run a script that creates and sends the alarm status message for you.
Before you can run the script, you must configure your AWS credentials on your local machine. The script uses these credentials to authenticate with AWS. For more information, see Prerequisites (p. 13).

To change the status of an alarm

1. Create a file called `set_alarm_status.py` and copy the following Python code into the file.

```
# Enable or disable an alarm detector for an asset property in AWS IoT SiteWise.
import argparse
import boto3
import json
import sys
import uuid

# Configures the argument parser for this program.
def configureArgumentParser():
    parser = argparse.ArgumentParser()
    parser.add_argument("-a", "--asset-id", action="store",
                        dest="asset_id", required=True, help="The ID of the asset.")
    parser.add_argument("-p", "--property-id", action="store",
                        dest="property_id", required=True, help="The ID of the property.")
    parser.add_argument("-s", "--status", action="store",
                        dest="alarm_status", required=True, choices=["ON", "OFF"],
                        help="The status of the alarm.")
    parser.add_argument("-c", "--profile", action="store", dest="profile",
                        help="The AWS credentials profile to use. Defaults to the 'default' profile.")
    parser.add_argument("-r", "--region", action="store", dest="region",
                        help="The AWS Region to use. Defaults to the Region configured for the chosen profile.")
    return parser

# Sends an alarm status message to AWS IoT Events for an asset property.
def send_alarm_status_message(session, asset_id, property_id, alarm_status):
    iot_events_client = session.client("iotevents-data")
    payload = {
        "alarmId": "%s-%s" % (asset_id, property_id),
        "status": alarm_status
    }
    message = {
        "messageId": str(uuid.uuid4()),
        "inputName": "AlarmStatus",
        "payload": json.dumps(payload)
    }
    print("Sending alarm status message to AWS IoT Events:")
    print(message)
    response = iot_events_client.batch_put_message(messages=[message])
    print("Status code: %d" % int(response["ResponseMetadata"]['HTTPStatusCode']))

# Sends an alarm status message with user arguments.
if __name__ == "__main__":
    parser = configureArgumentParser()
    args = parser.parse_args()
    try:
        session = boto3.Session(profile_name=args.profile, region_name=args.region)
        send_alarm_status_message(session, args.asset_id, args.property_id,
                                   args.alarm_status)
    except Exception as e:
        print(str(e), file=sys.stderr)
        exit(1)
```

2. Run `set_alarm_status.py` from the command line with the following parameters:
Visualizing and sharing wind farm data in SiteWise Monitor

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

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

You, and your users with sufficient permissions, can create dashboards in each project to visualize your industrial data in meaningful ways. Then, your users can view these dashboards to quickly gain insights into your data and monitor your operation. You can configure administrative or read-only permissions to each project for every user in your company. For more information, see Monitoring data with AWS IoT SiteWise Monitor (p. 163).

In this tutorial, you build on the AWS IoT SiteWise demo that provides a sample set of data for a wind farm. You configure a portal in SiteWise Monitor and create a project and dashboards to visualize the wind farm data. Then, you create additional users who you give permissions to own or view the project and its dashboards.

Note
When you use SiteWise Monitor, you're charged per AWS SSO 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 (p. 43)
• Creating a portal in SiteWise Monitor (p. 43)
Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don’t have one, see Setting up an AWS account (p. 7).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- A running AWS IoT SiteWise wind farm demo. When you set up the demo, it defines models and assets in AWS IoT SiteWise and streams data to them to represent a wind farm. For more information, see Using the AWS IoT SiteWise demo (p. 8).

Creating a portal in SiteWise Monitor

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

To create a portal

1. Sign in to the AWS IoT SiteWise console.
2. Review the AWS Regions (p. 7) where AWS IoT SiteWise is supported and switch Regions, if needed. If you already set up AWS SSO, you must switch to the Region where you have AWS SSO configured. You must run the AWS IoT SiteWise demo in the same Region. For more information, including instructions in case you set up AWS SSO in a Region that AWS IoT SiteWise doesn’t support, see Enabling AWS SSO (p. 167).
3. In the left navigation pane, choose Portals.
4. Choose Create portal.
5. If you already enabled AWS SSO in the current Region, skip to step 6. Otherwise, complete the following steps to enable AWS SSO:
   a. On the Enable AWS Single Sign-On (SSO) page, enter your Email address, First name, and Last name to create an AWS SSO user for yourself to be the portal administrator. Use an email address you can access so that you can receive an email to set a password for your new AWS SSO user.

In a portal, the portal administrator creates projects and assigns users to projects. You can create more users later.
b. Choose **Create user**.

6. On the **Portal configuration** page, complete the following steps:
   a. Enter a name for your portal, such as **WindFarmPortal**.
   b. (Optional) Enter a description for your portal. If you have multiple portals, use meaningful descriptions to keep track of what each portal contains.
   c. 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.
   d. Choose **Create portal**.
7. On the **Invite administrators** page, complete the following steps:

   a. Choose a user to be the portal administrator. If you're using SiteWise Monitor for the first time, choose the user that you created when you enabled AWS SSO earlier in this tutorial.

   b. (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 will be the portal administrator, you don't need to email yourself.

   c. Choose **Next**.

8. On the **Assign users** page, choose **Assign users** to create the portal. You can assign portal users that are the end users of your portal. The portal administrator can later assign these users as project owners, who can create dashboards in projects, or project viewers, who have read-only access to the projects that they're assigned. You can create a portal without assigning portal users.

The portals page opens with your new portal listed.
Signing in to a portal

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

To sign in to a portal

1. On the **Portals** page, choose your new portal's **Link** to open your portal in a new tab.

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

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

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

   All portal users, including the portal administrator, must sign in with their AWS SSO user credentials. These credentials are typically not the same credentials that you use to sign in to the AWS Management Console.
Creating a wind farm project

In this procedure, you create a project in your portal. Projects are resources that define a set of permissions, assets, and dashboards, which you can configure to visualize asset data in that project. You can assign other portal users as owners or viewers of each project. Project owners create and maintain dashboards to visualize data. Project owners also assign viewers to projects to grant access to dashboards and data. With projects, you define who has access to which subsets of your operation and how those subsets’ data is visualized.

**To create a wind farm project**

1. In the left navigation pane in your portal, choose the **Asset library** tab. In the asset library, 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 in the asset library, 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 right. 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, choose **Create new project**, and then choose **Next**.

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

Your new project's page opens.
Creating dashboards to visualize wind farm data

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

To create a dashboard with visualizations

1. On your new project's page, choose Create dashboard.
2. In the dashboard, choose Edit in the upper right.

   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.

3. In the edit view, rename your dashboard.

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

5. Choose the arrow next to Demo Wind Farm Asset to expand the wind farm asset hierarchy and view all of the wind turbine assets that you made available to your project earlier.
6. Under Demo Turbine Asset 1, drag Wind Speed to the space next to the Total Average Power visualization to create a visualization for wind speed.

7. 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.
8. Repeat steps 6 and 7 for the wind turbines' Torque (KiloNewton Meter) properties to create a visualization for wind turbine torque. Create this visualization in the space below the Total Average Power visualization.

9. Choose the visualization type icon for the Torque (KiloNewton Meter) visualization, and then choose the bar chart icon.
10. Repeat steps 6 and 7 for the wind turbines’ Wind Direction properties to create a visualization for wind direction. Create this visualization in the space below the Wind Speed visualization.

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

13. Choose **Done** in the upper right to save your dashboard.

Your dashboard should look similar to the following screenshot.
14. (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.
Creating AWS SSO users

In this procedure, you create additional AWS SSO users that you can add to your portal. After you have additional users in your portal, you can add them as owners or viewers of your projects to share your operational data.

To create AWS SSO users

1. Navigate to the AWS SSO console.
2. In the left navigation pane, choose Users.
3. Choose Add user.
4. Enter details for your new AWS SSO user, then choose Next: Groups. Later in this tutorial, you make this user an owner of your wind farm project.
5. On the Add users to groups page, choose Add user.

You should see a confirmation similar to that shown in the following screenshot.

6. Repeat steps 3 through 5 to create another AWS SSO user. Later in this tutorial, you make this user a viewer of your wind farm project.
Adding AWS SSO users to a portal

In this procedure, you add your new AWS SSO users to your wind farm portal. After you add users to a portal, you can add them to projects as owners or viewers within that portal.

To add AWS SSO users to a portal

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Portals.
3. Choose your portal, WindFarmPortal.
5. Choose the two AWS SSO users that you created in the previous procedure, and then choose **Assign users**.

In the portal, you can now assign these AWS SSO users to projects as owners or viewers.

**Assigning users to a project**

In this procedure, you assign your new AWS SSO users as owners or viewers of your wind farm project. Project owners can edit dashboards and share the project with other users. Project viewers can view dashboards but not edit them. For more information about roles in SiteWise Monitor, see SiteWise Monitor roles (p. 164).

**To assign users to a project as owners or viewers**

1. Navigate to your portal, **WindFarmPortal**, and sign in as your portal administrator (for example, **john.doe@example.com**).
2. In the left navigation pane, choose **Projects**.
3. Choose your project, **Wind Farm 1**.
4. Under **Project owners**, choose **Add owners** or **Edit users**.
5. Choose the user to add as a project owner (for example, Mary Major), and then choose the >> icon.

![Project owners](image)

6. Choose Save.

![Project owners](image)

Your AWS SSO 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.

7. Under Project viewers, choose Add viewers.

8. Choose the user to add as a project viewer, and then choose the >> icon.
9. Choose Save.

Your other AWS SSO user can sign in to this portal to view, but not edit, the dashboards in the wind farm project.

10. (Optional) Sign in to the portal as your new project owner or project viewer accounts to explore the portal as a user with fewer permissions than a portal administrator.

   **Note**
   You're charged for each AWS SSO user that signs in to a portal, so you incur charges if you sign in as these users. For more information, see AWS IoT SiteWise Pricing.

11. Now that you completed the tutorial, continue to explore your demo wind farm in SiteWise Monitor. When you're done, follow the next procedure to clean up your resources.

### Cleaning up resources after the tutorial

After you complete the tutorial, you can clean up your resources. You aren't charged for SiteWise Monitor if users don't sign in to your portal, but you can delete your portal and AWS SSO users. Your demo wind farm assets are deleted at the end of the duration that you chose when you created the demo, or you can delete the demo manually. For more information, see Deleting the AWS IoT SiteWise demo (p. 10).

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

**To delete a portal**

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

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

4. In the **Delete portal** dialog, choose Remove administrators and users.
5. Enter **delete** to confirm deletion, and then choose **Delete**.

![Delete portal]

To delete AWS SSO users

1. Navigate to the AWS SSO console.
2. In the left navigation pane, choose **Users**.
3. Select the check box for each user to delete, and then choose **Delete users**.

![Delete users dialog]

4. In the **Delete users** dialog, enter **DELETE**, and then choose **Delete users**.
Ingesting data to AWS IoT SiteWise from AWS IoT things

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

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

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

In this tutorial, you learn how to ingest data from AWS IoT things’ device shadows to assets in AWS IoT SiteWise. To do so, you create one or more AWS IoT things and run a script that updates each thing’s device shadow with CPU and memory usage data. You use CPU and memory usage data in this tutorial to imitate realistic sensor data. Then, you create a rule with an AWS IoT SiteWise action that sends this data.
Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Setting up an AWS account (p. 7).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- Python 3 installed on your development computer or installed on the device that you want to register as an AWS IoT thing.

Creating an AWS IoT policy

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

To create an AWS IoT policy

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

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

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

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

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

8. Choose Create.

Creating and configuring an AWS IoT thing

In this procedure, you create and configure an AWS IoT thing. You can register your development computer as an AWS IoT thing to easily complete this tutorial. When you apply this tutorial's concepts to a real-world application, you can create and configure AWS IoT things on any device that can run an AWS IoT SDK, including AWS IoT Greengrass and FreeRTOS. For more information, see AWS IoT SDKs in the AWS IoT Developer Guide.

To create and configure an AWS IoT thing

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

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

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

   ```bash
   mkdir device1
   ```

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

3. Navigate to the AWS IoT console.

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

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

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

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

   Important
   The thing name must match the name used in the policy that you created earlier in this tutorial. Otherwise, your device can't connect to AWS IoT.
8. On the **Add a certificate for your thing** page, choose **Create certificate**. Certificates enable AWS IoT to securely identify your devices.

9. Choose the **Download** links to download your thing's certificate, public key, and private key. Save all three files to the directory that you created for your thing's certificates (for example, `iot-sitewise-rule-tutorial/device1`).

**Important**  
This is the only time that you can download your thing's certificate and keys, which you need for your device to successfully connect to AWS IoT.
10. Choose the root CA Download link to open a documentation page where you choose and download a root CA certificate. Save the root CA certificate to the iot-sitewise-rule-tutorial. We recommend downloading Amazon Root CA 1.

11. Choose Activate.

12. Choose Attach a policy.

13. On the Add a policy for your thing page, choose the policy that you created earlier in this tutorial (SiteWiseTutorialDevicePolicy), then choose Register Thing.

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

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

Creating a device asset model

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

To create an asset model that represents a device

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

Measurement properties represent a device's raw data streams. For more information, see Measurements (p. 143).
6. Choose Add measurement to add a second measurement property.
7. In the second row under Measurement definitions, do the following:
   a. In Name, enter Memory Usage.
   b. In Unit, enter %.
   c. Leave the Data type as Double.

8. Under Metric definitions, do the following:
   a. In Name, enter Average CPU Usage.
   b. In Formula, enter \( \text{avg}(\text{CPU Usage}) \). 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 Metrics (p. 144).

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

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

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

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

   c. In **Time interval**, enter **5 minutes**.

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

11. (Optional) Add other additional metrics that you're interested in calculating per device. Some interesting functions include `min` and `max`. For more information, see Using formula expressions (p. 146). In the next section, you create a parent asset that can calculate metrics using data from your entire fleet of devices.

12. Choose **Create model**.

## Creating a device fleet asset model

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

### To create an asset model that represents a device fleet

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

   a. Choose **Add hierarchy**.

   b. In **Hierarchy name**, enter **Device**.

   c. In **Hierarchy model**, choose your device asset model (**SiteWise Tutorial Device Model**).
Creating and configuring a device asset

In this procedure, you create a device asset from your device asset model. Then, you define property aliases for each measurement property. A property alias is a string that uniquely identifies an asset property. You can later use these aliases, rather than asset ID and property ID, to identify a property to which to upload data. For more information, see Mapping industrial data streams to asset properties (p. 152).

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. In **Asset model**, choose your device asset model, **SiteWise Tutorial Device Model**.
5. In **Name**, enter **SiteWise Tutorial Device 1**.
6. Choose **Create asset**.

![Create a SiteWise Tutorial Device Model](image)

7. For your new device asset, choose **Edit**.

![SiteWise Tutorial Device 1](image)

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

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

![Measurement](image)

10. Choose **Save asset**.
11. If you created multiple AWS IoT things earlier, repeat steps 3 through 10 for each device, and increment the number in the asset name and property aliases accordingly. For example, the
Creating and configuring a device fleet asset

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

To create a device fleet asset and associate device assets

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

7. For your new device fleet asset, choose Edit.

8. Under Assets associated to this asset, choose Add associated asset.
9. Under Hierarchy, choose Device. This hierarchy identifies the hierarchical relationship between device and device fleet assets. You defined this hierarchy in the device fleet asset model earlier in this tutorial.
10. Under **Asset**, choose your device asset, **SiteWise Tutorial Device 1**.

![Hierarchy of device assets](image)

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

12. Choose **Save asset**.

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

![Hierarchy of device assets](image)

---

### Creating a rule in AWS IoT Core to send data to device assets

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

**To create a rule with an AWS IoT SiteWise action**

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

```
SELECT *
FROM '$aws/things/+/shadow/update/accepted'
WHERE startsWith(topic(3), "SiteWiseTutorialDevice")
```

This rule query statement works because the device shadow service publishes shadow updates to `$aws/things/thingName/shadow/update/accepted`. For more information about device shadows, see Device shadow service in the AWS IoT Developer Guide.

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

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

7. On the Select an action page, choose Send message data to asset properties in AWS IoT SiteWise to create an AWS IoT SiteWise rule action.
8. Choose **Configure action** at the bottom of the page.

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

   a. Choose **By property alias**.

   ![Configure action](image)

   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](https://aws.amazon.com/iot).

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

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

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

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

g. In Data type, choose Double.

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

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

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

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

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

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

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

vi. In Data type, choose Double.

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

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

k. Enter a role name and choose Create role.

![Create a new role](image)

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

l. Choose Add action.
10. (Optional) Configure an error action that you can use to troubleshoot your rule. For more information, see Troubleshooting a rule (p. 82).
11. Choose Create rule at the bottom of the page to finish creating the rule.

Running the device client script

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

To configure and run the device client script

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

Your custom endpoint should look like the following example.

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

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

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

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

```
pip3 install AWSIoTPythonSDK
```

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

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

```
pip3 install psutil
```

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

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

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

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

77
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 {}...
        while True: performance = self.readPerformance()
        print("[{}]".format(self.thingName))
        print("CPU:\t{}\%

            performance["cpu"]")
        print("Memory:\t{}\%

            performance["memory"]")
        payload = { "state": { "reported": performance } }
        deviceShadowHandler.shadowUpdate(json.dumps(payload), 
                                          self.shadowUpdateCallback, 5)
        time.sleep(args.requestDelay)

    # Configures the MQTT shadow client for this thing.
def configureMQTTClient(self):
        mqttClient = AWSIoTMQTTShadowClient(self.thingName)
        mqttClient.configureEndpoint(self.host, self.port)
        mqttClient.configureCredentials(self.rootCAPath, self.privateKeyPath, 
                                         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 {} {}
".format(token, responseStatus))

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

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

8. Run thing_performance.py from the command line with the following parameters:

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

Your command should look similar to the following example.

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

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

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

10. Follow these steps to verify that the script is updating the device shadow:

a. Navigate to the AWS IoT console.
b. In the left navigation pane, choose Manage and then choose Things.
c. Choose your thing, SiteWiseTutorialDevice1.
d. In the left navigation pane on your thing's page, choose Shadow.
e. Verify that the Shadow state looks like the following example.

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

f. If your thing's shadow state is empty or doesn't look like the previous example, check that the script is running and successfully connected to AWS IoT. If the script continues to time out when connecting to AWS IoT, check that your thing policy (p. 63) 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.

f. If the CPU Usage and Memory Usage properties don't have the latest values, refresh the page. If values don't appear after a few minutes, see Troubleshooting a rule (p. 82).

12. You have completed this tutorial. If you want to explore live visualizations of your data, you can configure a portal in AWS IoT SiteWise Monitor. For more information, see Monitoring data with AWS IoT SiteWise Monitor (p. 163). Otherwise, you can press CTRL+C in your command prompt to stop the device client script. It's unlikely the Python program will send enough messages to incur charges, but it's a best practice to stop the program when you're done.
Cleaning up resources after the tutorial

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

**To delete hierarchical assets in AWS IoT SiteWise**

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

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

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

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

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

   ![](image.png)

   When you delete an asset, AWS IoT SiteWise discards all data from that asset's properties.
7. Repeat steps 4 through 6 for each device asset and the device fleet asset (**SiteWise Tutorial Device Fleet 1**).

**To delete hierarchical asset models in AWS IoT SiteWise**

1. Navigate to the AWS IoT SiteWise console.
2. If you haven't already, delete your device and device fleet assets. For more information, see the previous procedure (p. 81). You can't delete a model if you have assets that were created from that model.
3. In the left navigation pane, choose **Models**.
4. Choose your device fleet asset model (**SiteWise Tutorial Device Fleet Model**).
When you delete hierarchical asset models, you must delete the parent asset model first.

5. Choose **Delete**.

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

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

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

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

**Troubleshooting a rule**

Follow the steps in this procedure to troubleshoot your rule if the CPU and memory usage data isn't appearing in AWS IoT SiteWise as expected. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client. You can also configure logging to CloudWatch Logs to troubleshoot. For more information, see Troubleshooting an AWS IoT SiteWise rule action (p. 235).
To add a republish error action to a rule

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

4. Under Error action, choose Add action.
5. Choose Republish a message to an AWS IoT topic.
6. Choose Configure action at the bottom of the page.
7. In Topic, enter sitewise/rule/tutorial/error. AWS IoT Core will republish error messages to this topic.
8. Choose **Select** to grant AWS IoT Core access to perform the error action.

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

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

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

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

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

**To subscribe to the error action topic**

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

4. When error messages appear, view the `failures` array in any error message to diagnose issues. For more information about possible issues and how to resolve them, see [Troubleshooting an AWS IoT SiteWise rule action](p. 235).

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
```

---

**Publishing property value updates to Amazon DynamoDB**

You can store your data in Amazon DynamoDB to easily access historical asset data without needing to repeatedly query the AWS IoT SiteWise API, which returns paginated value histories. After you complete this tutorial, you can easily create custom software that consumes your asset data, such as a live map of wind speed and direction over an entire wind farm. If you're looking to monitor and visualize your data without implementing a custom software solution, see [Monitoring data with AWS IoT SiteWise Monitor](p. 163).

In this tutorial, you build on the AWS IoT SiteWise demo that provides a sample set of data for a wind farm. You configure property value updates from the wind farm demo to send data, through AWS IoT
Core rules, to a DynamoDB table that you create. When you enable property value updates, AWS IoT SiteWise sends your data to AWS IoT Core in MQTT messages. Then, you can define AWS IoT Core rules that perform actions, such as the DynamoDB action, depending on the contents of those messages. For more information, see Interacting with other AWS services (p. 184).

Topics

- Prerequisites (p. 86)
- Configuring AWS IoT SiteWise to publish property value updates (p. 86)
- Creating a rule in AWS IoT Core (p. 88)
- Creating a DynamoDB table (p. 90)
- Configuring the DynamoDB rule action (p. 91)
- Exploring data in DynamoDB (p. 92)
- Cleaning up resources after the tutorial (p. 93)
- Troubleshooting a rule (p. 96)

Prerequisites

To complete this tutorial, you need the following:

- An AWS account. If you don't have one, see Setting up an AWS account (p. 7).
- A development computer running Windows, macOS, Linux, or Unix to access the AWS Management Console. For more information, see Getting Started with the AWS Management Console.
- An IAM user with administrator permissions.
- A running AWS IoT SiteWise wind farm demo. When you set up the demo, it defines models and assets in AWS IoT SiteWise and streams data to them to represent a wind farm. For more information, see Using the AWS IoT SiteWise demo (p. 8).

Configuring AWS IoT SiteWise to publish property value updates

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

To enable property value update notifications on asset properties

1. Sign in to the AWS IoT SiteWise console.
2. Review the **AWS Regions** (p. 7) where AWS IoT SiteWise is supported and switch AWS Regions, if necessary. Switch to a Region where you’re running the AWS IoT SiteWise demo.
3. In the left navigation pane, choose **Assets**.
4. Choose the arrow next to **Demo Wind Farm Asset** to expand the wind farm asset's hierarchy.

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

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

7. Choose **Save asset** at the bottom of the page.
8. Repeat steps 5 through 7 for each demo turbine asset.
9. Choose a demo turbine (for example, **Demo Turbine Asset 1**).
10. Choose **Measurements**.
11. Choose the copy icon next to the **Wind Speed** property to copy the notification topic to your clipboard. Save the notification topic to use later in this tutorial. You only need to record the notification topic from one turbine.

![Notification Topic Copy](image)

The notification topic should look like the following example.

```
/aws/siteWISE/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/
/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-
cdef-33333EXAMPLE
```

### Creating a rule in AWS IoT Core

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

**To create a rule with a DynamoDB action**

1. Navigate to the **AWS IoT console**. If a **Get started** button appears, choose it.
2. In the left navigation pane, choose **Act** and then choose **Rules**.

![AWS IoT Console](image)

3. If a **You don't have any rules yet** dialog box appears, choose **Create a rule**. Otherwise, choose **Create**.
4. Enter a name and description for the rule.
5. Find the notification topic that you saved earlier in this tutorial.

```
$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
```

Replace the asset ID (the ID after `assets/`) in the topic with a `+` to select the wind speed property for all demo wind turbine assets. The `+` topic filter accepts all nodes from a single level in a topic. Your topic should look the following example.

```
$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/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 **Set one or more actions**, choose **Add action**.

   - **Add action**

      Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*required*)

      - **Error action**

        Optionally set an action that will be executed when something goes wrong with processing your rule.
8. On the **Select an action** page, choose **Split message into multiple columns of a DynamoDB table (DynamoDBv2)**.

   ![Select an action](image)

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

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

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

## Creating a DynamoDB table

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

**To create a DynamoDB table**

1. In the DynamoDB console dashboard, choose **Create table**.
2. Enter a name for your table.
3. For **Primary key**, do the following:
   a. Enter **timestamp** as the partition key.
   b. Choose the **Number** type.
   c. Select the **Add sort key** check box.
   d. Enter **asset** as the sort key, and leave the default sort key type of **String**.

4. Choose **Create**.

   When the **Table is being created** notice disappears, your table is ready.

5. Return to the tab with the **Configure action** page. Keep the DynamoDB tab open while you complete the following procedures.

### Configuring the DynamoDB rule action

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

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

2. Choose Create role to create an IAM role that grants AWS IoT Core access to perform the rule action.
3. Enter a role name and choose Create role.

4. Choose Add action.
5. Choose Create rule at the bottom of the page to finish creating the rule.

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

Exploring data in DynamoDB

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

To explore asset data in DynamoDB

1. Return to the tab with the DynamoDB table open.
2. In the table you created earlier, choose the Items tab to view the data in the table. Refresh the page if you don't see rows in the table. If rows don't appear after a few minutes, see Troubleshooting a rule (p. 96).
3. In a row in the table, choose the edit icon to expand the data.

4. Choose the arrow next to the windspeed structure to expand the list of wind speed data points. Each list reflects a batch of wind speed data points sent to AWS IoT SiteWise by the wind farm demo. You might want a different data format if you set up a rule action for your own use. For more information, see Querying asset property notification messages (p. 185).

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

**Cleaning up resources after the tutorial**

After you complete the tutorial, clean up your resources to avoid incurring additional charges. Your demo wind farm assets are deleted at the end of the duration that you chose when you created the
demo, or you can delete the demo manually. For more information, see Deleting the AWS IoT SiteWise demo (p. 10).

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

**To disable property value update notifications on asset properties**

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

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

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

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

**To delete a DynamoDB table**

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

---

**Troubleshooting a rule**

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

**To add a republish error action to a rule**

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

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

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

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

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

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

**To subscribe to the error action topic**

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

![MQTT client](image)

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

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

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

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

• Use an AWS IoT SiteWise gateway (p. 100) to upload data from OPC-UA servers. The gateway serves as the intermediary between AWS IoT SiteWise and your OPC-UA servers. AWS IoT SiteWise provides an AWS IoT Greengrass connector that you can deploy on any platform that can run AWS IoT Greengrass to set up a gateway.
• Use AWS IoT Core rules (p. 131) to upload data from MQTT messages published by an AWS IoT thing or another AWS service.
• Use AWS IoT Events actions (p. 137) to upload data from AWS IoT Events when an event occurs.
• Use the AWS IoT SiteWise API (p. 137) to upload data from any other source.

Ingesting data using a gateway

An AWS IoT SiteWise gateway connects to OPC-UA servers to deliver your industrial data streams to the AWS Cloud. You can deploy the AWS IoT SiteWise connector on any device or platform that can run AWS IoT Greengrass.

Note
This documentation describes how to set up the latest version of the AWS IoT SiteWise connector, version 6. For more information and the connector version changelog, see the AWS IoT SiteWise connector in the AWS IoT Greengrass Developer Guide.

Topics
• Choosing a gateway platform (p. 100)
• Configuring a gateway (p. 101)
• Configuring data sources (p. 116)
• Upgrading a connector (p. 129)

Choosing a gateway platform

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

• Supports AWS IoT Greengrass Core software v1.10.0 or later. For more information, see Supported platforms and requirements in the AWS IoT Greengrass Developer Guide.
• Has at least 1 GB of RAM.
• Has at least 10 GB of free disk space.
• Supports a Java 8 virtual machine (JVM).
Choose a gateway with sufficient disk, networking, and compute capacity for your workload.

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

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

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

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

### Configuring a gateway

A gateway serves as the intermediary between your OPC-UA servers and AWS IoT SiteWise. You can easily deploy the AWS IoT SiteWise gateway software on any platform that can run AWS IoT Greengrass. For more information, see [Choosing a gateway platform](p. 100).

To configure a gateway that runs on Amazon EC2, you can create the required dependencies from an AWS CloudFormation template. For more information, see [Configuring gateway dependencies on Amazon Elastic Compute Cloud](p. 115).

**Note**

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

#### Topics

- Setting up the gateway environment (p. 101)
- Creating an IAM policy and role (p. 103)
- Configuring an AWS IoT Greengrass group (p. 109)
- Configuring the AWS IoT SiteWise connector (p. 111)
- Adding the gateway to AWS IoT SiteWise (p. 114)
- Configuring gateway dependencies on Amazon Elastic Compute Cloud (p. 115)

### Setting up the gateway environment

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

**Note**

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

#### To set up the gateway

1. As appropriate, modify the BIOS settings of the gateway as follows.
a. Ensure that the gateway automatically restarts after a potential power failure, if applicable.
b. Ensure that the gateway won’t hibernate or sleep, if applicable.

2. Ensure that the gateway connects to the internet.

3. (Optional) To use the gateway without the mouse, keyboard, and monitor, do the following steps to set up ssh on the gateway:

   a. If you haven’t already installed the SSH package, run the following command.

   ```
   sudo apt install ssh
   ```

   b. Run the following command.

   ```
   service ssh status
   ```

   c. Search for `Active: active (running)` in the output to confirm that the SSH server is running,

   d. Press Q to exit.

   You can use SSH to connect to the gateway from another computer by running the command `ssh username@IPv4`, where `username` is the user login and `IPv4` is the IP address of the gateway.

   Run the following command to use a port other than the default port 22.

   ```
   ssh username@IPv4 -p port-number
   ```

4. Download and install AWS IoT Greengrass Core software v1.10.0 or later, and create an AWS IoT Greengrass group for your gateway. To do so, follow the instructions in Getting started with AWS IoT Greengrass in the AWS IoT Greengrass Developer Guide.

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

   **Important**

   Review the AWS Regions (p. 7) where AWS IoT SiteWise is supported. When you choose a Region for AWS IoT Greengrass, make sure that the Region also supports AWS IoT SiteWise. Otherwise, you can’t connect your gateway to AWS IoT SiteWise.

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

5. Run the following commands to install Java 8.

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

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

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

   ```
   java -version
   ```

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

   ```
   sudo ln -s /usr/bin/java /usr/bin/java8
   ```
8. Run the following command to create a `/var/sitewise` data directory and give the `ggc_user` permissions for that directory. AWS IoT SiteWise stores data in this directory. You created the `ggc_user` when you set up AWS IoT Greengrass earlier in this procedure.

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

The `/var/sitewise` is the default directory that AWS IoT SiteWise uses. You can customize the directory path (for example, replace `/var/sitewise` with `/var/custom/path/`), but doing so requires extra steps after the AWS IoT SiteWise gateway is created. For more information, see step 6 in Configuring the AWS IoT SiteWise connector (p. 111).

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

- **Ports:** 443, 8443, and 8883
  
  **Important**
  
  You can configure AWS IoT Greengrass Core to use only port 443 for all network communications. For more information, see Connect on port 443 or through a network proxy in the AWS IoT Greengrass Developer Guide.
  
- **The IP address of your gateway (port 443).** To obtain the IP address, run the `ip address` or `ifconfig` command and note the `inet` value (for example, `203.0.113.0`).
  
- **The AWS IoT SiteWise data endpoint:** `data.iotsitewise.region.amazonaws.com` (port 443).
  
- **The AWS endpoints that the gateway uses.** You can find these in the `/greengrass/config/config.json` file:
  
  - `iotHost`: `greengrass-id.iot.region.amazonaws.com` (ports 443, 8443, and 8883).
  
  - Using the same `greengrass-id` found for `iotHost` in the previous step, add `greengrass-id.credentials.iot.region.amazonaws.com` (port 443) to the allow list.

If your firewall can’t use DNS names, you must resolve the names (endpoints) to their associated IP addresses first (for example, `nslookup greengrass.iot.region.amazonaws.com`).

10. If the AWS IoT Greengrass Core software isn’t already running, run the following command to start the AWS IoT Greengrass Core software.

```
cd /greengrass/ggc/core
sudo ./greengrassd start
```

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

11. Configure the AWS IoT Greengrass Core software to automatically start when your gateway turns on. Consult the documentation for your gateway’s operating system.

## Creating an IAM policy and role

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

### To create an IAM policy and role

1. Navigate to the IAM console.
2. In the navigation pane, choose Policies, and then choose Create policy.
3. On the JSON tab, delete the current contents of the policy field, and paste the following policy into the field.

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

**Note**

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

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

4. Choose **Review policy**.
5. Enter a name and description for the policy, and then choose **Create policy**.

6. In the navigation pane, choose **Roles**, and then choose **Create role**.

   ![Roles](image)

   **What are IAM roles?**
   IAM roles are a secure way to grant permissions to entities that you trust. Examples of entities include the following:
   - IAM user in another account
   - Application code running on an EC2 instance that needs to perform actions on AWS resources
   - An AWS service that needs to act on resources in your account to provide its features
   - Users from a corporate directory who use identity federation with SAML

   IAM roles issue keys that are valid for short durations, making them a more secure way to grant access.

   **Additional resources:**
   - IAM Roles FAQ
   - IAM Roles Documentation
   - Tutorial: Setting Up Cross Account Access
   - Common Scenarios for Roles

   ![Create role](image)

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

   ![Create role page]

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

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

**Configuring an AWS IoT Greengrass group**

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

1. Navigate to the **AWS IoT Greengrass console**.
2. In the left navigation pane, under **Greengrass**, choose **Groups**, and then choose the group that you created in **Setting up the gateway environment** (p. 101).
3. In the left navigation pane, choose **Settings**. In the **Group Role** section, choose **Add Role**.

4. Choose the role that you created in Creating an IAM policy and role (p. 103), and then choose **Save**.
5. On the Settings page, in the Stream manager section, choose Edit.

Stream manager is a feature of AWS IoT Greengrass that enables your AWS IoT Greengrass Core to stream data to the AWS Cloud. AWS IoT SiteWise gateways require that stream manager is enabled. For more information, see Manage data streams on the AWS IoT Greengrass Core in the AWS IoT Greengrass Developer Guide.

6. Choose Enable, and then choose Save.

7. In the upper-left corner, choose Services to prepare for the next procedure.

**Configuring the AWS IoT SiteWise connector**

In this procedure, you configure the AWS IoT SiteWise connector on your Greengrass group. Connectors are prebuilt modules that accelerate the development lifecycle for common edge scenarios. For more information, see AWS IoT Greengrass connectors in the AWS IoT Greengrass Developer Guide.

**To configure the AWS IoT SiteWise connector**

1. Navigate to the AWS IoT Greengrass console.
2. In the left navigation pane, under Greengrass, choose Groups, and then choose the group that you created in Setting up the gateway environment (p. 101).
3. In the left navigation page, choose Connectors. On the Connectors page, choose Add a connector.

4. Choose IoT SiteWise from the list and choose Next.
5. If your OPC-UA servers require authentication, you can create AWS Secrets Manager secrets with the server’s user name and password. Then, you can attach each secrets to your Greengrass group and choose them under List of ARNs for OPC-UA username/password secrets. For more information about how to create and configure secrets, see Configuring source authentication (p. 124). You can also add secrets to your connector later.

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

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

8. Choose Add.

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

10. Choose Automatic detection to start the deployment.

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

Adding the gateway to AWS IoT SiteWise

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

To add the gateway to AWS IoT SiteWise

1. Navigate to the AWS IoT SiteWise console.

2. Choose Add gateway.

3. On the Add SiteWise gateway page, do the following:

   1. Enter a Name for the gateway. Consider including the location of the gateway in the name so that you can easily identify it.

   2. For Greengrass group ID, choose the Greengrass group that you created earlier.

   Example

   ![Add SiteWise gateway page](image)

   3. Choose Add gateway.

   Gateway creation might take some time. Wait 5 minutes and then refresh the page.
Configuring a gateway

After your gateway creates, you can add a source for each OPC-UA server from which you want your gateway to ingest data. For more information, see Configuring data sources (p. 116).

Configuring gateway dependencies on Amazon Elastic Compute Cloud

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

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

The AWS CloudFormation template creates the following resources:

• An Amazon EC2 instance
• An Amazon Virtual Private Cloud
• An AWS IoT Greengrass group
• AWS Identity and Access Management (IAM) roles

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

Prerequisites

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

• The AWS IoT SiteWise service-linked role in your AWS account. This role creates automatically when you use the AWS IoT SiteWise console. For more information, see Using service-linked roles for AWS IoT SiteWise (p. 210).
• An AWS IoT Greengrass service role attached to your AWS account in the Region where you create this stack. For more information, see AWS IoT Greengrass service role in the AWS IoT Greengrass Developer Guide.

Creating the AWS CloudFormation stack

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

To create gateway dependencies on Amazon EC2

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

   The stack takes around 5 minutes to create. If the stack fails to create, your account might have insufficient permissions, or you might not have the prerequisite IAM roles. Follow these steps to delete the stack and try again:
   a. Choose **Delete** in the upper-right corner.
      
      The stack takes a few minutes to delete.
   b. If the stack fails to delete, choose **Delete** again.
   c. If the stack fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.
9. After the stack creates successfully, you can create a gateway with the AWS IoT Greengrass group that deploys to the Amazon EC2 instance. For more information, see *Adding the gateway to AWS IoT SiteWise* (p. 114).

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

### Configuring data sources

After you set up a gateway, you can configure data sources so that your gateway can ingest data from local servers to AWS IoT SiteWise. Each source represents a local server, such as an OPC-UA server, that your gateway connects and retrieves industrial data streams. For more information about setting up a gateway, see *Configuring a gateway* (p. 101).

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

After you create sources, you can associate your data streams with an asset. For more information about how to create and use assets, see *Modeling industrial assets* (p. 139).

**Topics**
- Configuring data sources (console) (p. 116)
- Configuring data sources (AWS CLI) (p. 118)
- Enabling your source servers to trust the gateway (p. 123)
- Configuring source authentication (p. 124)
- Using OPC-UA node filters (p. 128)

### Configuring data sources (console)

You can use the AWS IoT SiteWise console to add OPC-UA sources to your gateway.
To create a data source (console)

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose Gateways.
3. On the gateway you want to create a source for, choose Manage, and then choose View details.
4. Choose New source in the upper-right corner.
5. Enter a Nickname for the source.
6. Enter the Local endpoint of the data source server. For example, your local endpoint might look like opc.tcp://203.0.113.0:49320.
7. (Optional) Enter a Data stream prefix. The gateway adds this prefix to all data streams from this source. Use a data stream prefix to distinguish between data streams that have the same name from different sources. Each data stream should have a unique name within your account.
8. Choose a Message security mode for data in transit between your source server and your gateway. This should be the same message security mode that you choose on your data server.
   - If you choose a "sign" message security mode, the data in transit between the gateway and the source is signed but not encrypted.
   - If you choose an "encrypt" message security mode, the data in transit between the gateway and the source is signed and encrypted.

   **Important**
   If you choose a message security mode other than None, you must enable your source server to trust the gateway. For more information, see Enabling your source servers to trust the gateway (p. 123).
9. If your source requires authentication, choose an AWS Secrets Manager secret from the Authentication configuration list. The gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your gateway's IoT SiteWise connector to use them for source authentication. For more information, see Configuring source authentication (p. 124).

   **Tip**
   Your data server might have an option named Allow anonymous login. If this option is Yes, then your source doesn't require authentication.
10. (Optional) Add OPC-UA node filters to limit which OPC-UA paths are uploaded to AWS IoT SiteWise. You can use node filters to reduce your gateway's startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with /Server/. To define OPC-UA node filters, you can use node paths and the * and ** wildcard characters. For more information, see Using OPC-UA node filters (p. 128).
11. Choose Save.

    AWS IoT SiteWise deploys the gateway configuration to your AWS IoT Greengrass core. You don't need to manually trigger a deployment.
Configuring data sources (AWS CLI)

You can use the AWS IoT SiteWise API and AWS Command Line Interface to add sources to your gateway. You define sources in gateway capabilities. A gateway capability represents a software feature that runs on the gateway, such as the capability to collect industrial data from OPC-UA sources.

Gateway capabilities have the following components:

- A configuration – A JSON document that defines all of the data sources for a capability.
- A namespace – A unique string that identifies the type and version of a capability. For example, the OPC-UA source capability namespace is `iotsitewise:opcuacollector:` followed by `version`, where `version` is the version of the OPC-UA capability. All OPC-UA sources are defined in one capability with this namespace.
- A synchronization status – A status that indicates if a capability is synchronized between the AWS Cloud and the gateway. The sync status can be one of the following:
  - **IN_SYNC** – The gateway is running the capability configuration.
  - **OUT_OF_SYNC** – The gateway hasn't received the capability configuration.
  - **SYNC_FAILED** – The gateway rejected the capability configuration.

After you update a capability configuration, its sync status is **OUT_OF_SYNC** until the gateway receives and applies or rejects the updated configuration.

Use the following actions to query and update your gateway sources and capability configurations:

- **DescribeGateway** – Retrieves information about a specific gateway. The response includes a list of capability summaries, including capability namespaces.
- **DescribeGatewayCapabilityConfiguration** – Retrieves the configuration of a specific capability. Use this action to retrieve a capability configuration to update.
- **ListGateways** – Lists information about all gateways. The response includes a list of capability summaries for each gateway, including capability namespaces.
- **UpdateGatewayCapabilityConfiguration** – Updates a gateway capability configuration or defines a new capability configuration. This action identifies capabilities by a capability namespace. If you provide a namespace that already exists, this action updates the capability for that namespace. Otherwise, this action creates a new capability.

Warning
The **UpdateGatewayCapabilityConfiguration** action overwrites the existing capability configuration with the configuration that you provide in the payload. To avoid deleting your capability's configuration, you must add to the existing configuration when you update the capability.

Gateway capabilities

- **OPC-UA sources** (p. 118)

OPC-UA sources

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

This capability has the following versions.
## Capability configuration parameters

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

**sources**

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

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

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

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

    - **type**
      
      The certificate trust mode for the source. Choose one of the following:

      - **TrustAny** — The gateway trusts any certificate when it connects to the OPC-UA source.
      - **X509** — The gateway trusts an X.509 certificate when it connects to the OPC-UA source. If you choose this option, you must define `certificateBody` and `securityPolicy` 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 cryptographic algorithm to use so that you can secure messages that are read from the OPC-UA source. If you choose an algorithm other than `None`, you must choose `Sign` or `SignAndEncrypt` for `messageSecurityMode`. Choose one of the following:
• **NONE** – The gateway doesn’t secure messages from the OPC-UA source. If you choose this option, you must also choose **NONE** for `messageSecurityMode`.

• **BASIC128_RSA15** – The `Basic128Rsa15` algorithm.

• **BASIC256** – The `Basic256` algorithm.

• **BASIC256_SHA256** – The `Basic256Sha256` algorithm.

• **AES128_SHA256_RSAOAEP** – The `Aes128_Sha256_RsaOaep` algorithm.

• **AES256_SHA256_RSAPSS** – The `Aes256_Sha256_Rsapss` algorithm.

`messageSecurityMode`

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

• **NONE** – The gateway doesn’t secure connections to the OPC-UA source. If you choose this option, you must also choose **NONE** for `securityPolicy`.

• **SIGN** – Data in transit between the gateway and the OPC-UA source is signed but not encrypted. If you choose this option, you must choose an algorithm other than **NONE** for `securityPolicy`, and you must enable your source server to trust the gateway. For more information, see Enabling your source servers to trust the gateway (p. 123).

• **SIGN_AND_ENCRYPT** – Data in transit between the gateway and the OPC-UA source is signed and encrypted. If you choose this option, you must choose an algorithm other than **NONE** for `securityPolicy`, and you must enable your source server to trust the gateway. For more information, see Enabling your source servers to trust the gateway (p. 123).

`identityProvider`

An identity provider structure that contains the following information:

`type`

The type of authentication credentials required by the source. Choose one of the following:

• **Anonymous** – The source doesn’t require authentication to connect.

• **Username** – The source requires a user name and password to connect. If you choose this option, you must define `usernameSecretArn` in `identityProvider`.

`usernameSecretArn`

(Optional) The ARN of an AWS Secrets Manager secret. The gateway uses the authentication credentials in this secret when it connects to this source. You must attach secrets to your gateway’s IoT SiteWise connector to use them for source authentication. For more information, see Configuring source authentication (p. 124). This field is required if you choose **Username** for `type` in `identityProvider`.

`nodeFilterRules`

A list of node filter rule structures that define the OPC-UA data stream paths to send to the AWS Cloud. You can use node filters to reduce your gateway’s startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with `/Server/`. To define OPC-UA node filters, you can use node paths and the `*` and `**` wildcard characters. For more information, see Using OPC-UA node filters (p. 128).

Each structure in the list must contain the following information:

`action`

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

Definition

A node filter rule structure that contains the following information:

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

- **rootPath**
  
  The node filter path to evaluate against the root of the OPC-UA path hierarchy.

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

### Capability configuration examples

The following example defines a gateway capability configuration from a payload stored in a JSON file.

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

**Example**

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

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

**Example**

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 algorithm to secure messages.
• Uses the SIGN_AND_ENCRYPT mode to secure connections.
• Uses authentication credentials stored in a Secrets Manager secret.
• Filters out data streams except those whose path starts with /WindFarm/2/WindTurbine/.
• Adds /Washington to the start of every data stream path to distinguish between this "Wind Farm #2" and a "Wind Farm #2" in another area.

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

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

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

```json
{
  "sources": [
    {
      "name": "Wind Farm #3",
      "endpoint": {
        "certificateTrust": {
          "type": "X509",
          "certificateBody": "-----BEGIN CERTIFICATE-----
MIICiTCCAfIICCQD6m7oRw0uXOjANBgkqhkiG9w
-----END CERTIFICATE-----"
        }
      }
    }
  ]
}
```
Enabling your source servers to trust the gateway

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

AWS IoT SiteWise User Guide
Configuring data sources

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

---

**Enabling your source servers to trust the gateway**

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

The procedure might be similar to the following steps.

**To enable an OPC-UA server to trust the gateway**

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

If your OPC-UA servers require authentication credentials to connect, you can define a user name and password in a secret for each source in AWS Secrets Manager. Then, you add the secret to your Greengrass group and IoT SiteWise connector to make the secret available to your gateway. For more information, see Deploy secrets to the AWS IoT Greengrass core in the AWS IoT Greengrass Developer Guide.

After a secret is available to your gateway, you can choose it when you configure a source. Then, the gateway uses the authentication credentials from the secret when it connects to the source. For more information, see Configuring data sources (p. 116).

Topics
- Creating source authentication secrets (p. 124)
- Adding secrets to a Greengrass group (p. 125)
- Adding secrets to an IoT SiteWise connector (p. 126)

Creating source authentication secrets

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

To create a source authentication secret

1. Navigate to the Secrets Manager console.
2. Choose Store a new secret.
3. Under Select secret type, choose Other type of secrets.
4. Enter username and password key-value pairs for your OPC-UA server's authentication values, and then choose Next.
5. Enter a **Secret name** that begins with `greengrass-`, such as `greengrass-factory1-auth`.

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

6. Enter a **Description** and choose **Next**.

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

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

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

### Adding secrets to a Greengrass group

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

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

1. Navigate to the AWS IoT Greengrass console.
2. In the left navigation pane, under **Greengrass**, choose **Groups**, and then choose your group.
3. In the left navigation page, choose Resources. On the Resources page, choose the Secret tab, and then choose Add a secret resource.

![Image of adding a secret resource](image)

4. Choose Select and choose your secret from the list.

5. Choose Next.

6. In Secret resource name, enter a name for your secret resource and choose Save.

![Image of naming a secret resource](image)

**Adding secrets to an IoT SiteWise connector**

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

**To add a secret to your IoT SiteWise connector**

1. Navigate to the AWS IoT Greengrass console.
2. In the left navigation pane, under Greengrass, choose Groups, and then choose your group.
3. In the left navigation page, choose Connectors.

4. Choose the ellipsis icon for the IoT SiteWise connector to open the options menu, and then choose Edit.

5. Under List of ARNs for OPC-UA username/password secrets, choose Select, and then select each secret to add to this gateway. If you need to create secrets, see Creating source authentication secrets (p. 124).
If your secret doesn't appear, choose **Refresh**. If your secret still doesn't appear, check that you added the secret to your Greengrass group.

6. Choose **Save**.

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

8. Choose **Automatic detection** to start the deployment.

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

After your group deploys, you can configure a source that uses the new secret. For more information, see **Configuring data sources (p. 116)**.

### Using OPC-UA node filters

When you define OPC-UA data sources for an AWS IoT SiteWise gateway, you can define node filters. Node filters let you limit which data stream paths the gateway sends to the cloud. You can use node filters to reduce your gateway’s startup time and CPU usage by only including paths to data that you model in AWS IoT SiteWise. By default, gateways upload all OPC-UA paths except those that start with `/Server/`. You can use the `*` and `**` wildcard characters in your node filters to include multiple data stream paths with one filter. To learn how to set up OPC-UA sources for your gateway, see **Adding the gateway to AWS IoT SiteWise (p. 114)**.

**Note**

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

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

### OPC-UA node filter wildcards

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>Matches a single level in a data stream path.</td>
</tr>
<tr>
<td><code>**</code></td>
<td>Matches multiple levels in a data stream path.</td>
</tr>
</tbody>
</table>
**Note**
If you configure a source with a broad filter and then later change the source to use a more restrictive filter, AWS IoT SiteWise stops storing data that doesn't match the new filter.

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

- `/WA/Factory 1/Line 1/PCL1`
- `/WA/Factory 1/Line 1/PCL2`
- `/WA/Factory 1/Line 2/Counter1`
- `/WA/Factory 1/Line 2/PCL1`
- `/OR/Factory 1/Line 1/PCL1`
- `/OR/Factory 1/Line 2/Counter2`

Using the previous data streams, you can define node filters to limit what data to include from your OPC-UA source.

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

**Upgrading a connector**

**Important**
Version 6 of the IoT SiteWise connector introduces new requirements: AWS IoT Greengrass Core software v1.10.0 and stream manager. Before you upgrade your connector, check that your gateway meets these requirements, or you won't be able to deploy your gateway.

You can easily upgrade your gateway's connector after a new IoT SiteWise connector version is released.

**Note**
In this procedure, you redeploy your Greengrass group and restart your gateway. Your gateway won't ingest data while it's restarting. The time to restart your gateway depends on the number of OPC-UA tags on your gateway's OPC-UA sources. Restart time can range from a few seconds (for a gateway with few tags) to several minutes (for a gateway with many tags).

**To upgrade an IoT SiteWise connector**

1. Navigate to the AWS IoT Greengrass console.
2. In the left navigation pane, under Greengrass, choose Groups, and then choose the group that you created when you set up your gateway.
3. In the left navigation pane, choose **Connectors**.
4. On the **Connectors** page, choose **Available** next to the **IoT SiteWise** connector.

If you don't see the **Available** element, your connector is already the latest version.

5. On the **Upgrade connector** page, enter your connector's parameters and then choose **Upgrade**.
6. In the upper-right corner, in the **Actions** menu, choose **Deploy**.
7. Choose **Automatic detection** to start the deployment.

If the deployment fails, choose **Deploy** again. If the deployment continues to fail, see [AWS IoT Greengrass deployment troubleshooting](#).
Ingesting data using AWS IoT Core rules

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

You can follow a tutorial that walks through the steps required to set up a rule that ingests data from AWS IoT things through their device shadows. For more information, see Ingesting data to AWS IoT SiteWise from AWS IoT things (p. 62).

You can also send data from AWS IoT SiteWise to other AWS services. For more information, see Interacting with other AWS services (p. 184).

Topics
- Granting AWS IoT the required access (p. 131)
- Configuring the AWS IoT SiteWise rule action (p. 132)
- Reducing costs with basic ingest (p. 136)
- Troubleshooting the AWS IoT SiteWise rule action (p. 137)

Granting AWS IoT the required access

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

If you create the rule action in the AWS IoT console, you can choose a root asset to easily 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/**
          ]
        }
      }
    }
  ]
}
```
You can 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": "*"
        }
    ]
}
```

**Configuring the AWS IoT SiteWise rule action**

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

**Note**

When you send data to AWS IoT SiteWise with the rule action, your data must meet all of the requirements of the `BatchPutAssetPropertyValue` action. For example, your data can’t have a timestamp older than 15 minutes from current Unix epoch time. For more information, see Ingesting data with the AWS IoT SiteWise API.

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

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

- The **Asset ID** (`assetId`) and **Property ID** (`propertyId`) of the asset property that you’re sending data to. If you choose this option in the AWS IoT console, you can use a drop-down list to choose an asset model and property from AWS IoT SiteWise in the current Region.
- The **Property alias** (`propertyAlias`), which is a data stream alias (for example, `/company/ windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 152).

For the timestamp in each entry, you can use the timestamp that your sensor or equipment reported. The timestamp is composed of 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 isn’t in seconds, or has a decimal portion, you must convert it to seconds and nanosecond offset. Otherwise, AWS IoT SiteWise rejects the request.

If your sensor or equipment reports time data in string format (for example, `2020-03-03T14:57:14.699Z`), you must configure an AWS Lambda function that your rule calls to convert the timestamp into numerical values. For more information, see Configuring a rule that parses timestamp strings (p. 135).
If your sensor or equipment doesn't report accurate time data, you can get the current Unix epoch time from the AWS IoT rules engine with `timestamp()`. This function reports time in milliseconds, so you must convert the value to time in seconds and offset in nanoseconds:

- 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.

The above conversions use substitution templates to calculate values when a rule is triggered. You can also use substitution templates to pull values from the message payload for several parameters in the action. 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.

**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. You could use this example to define one rule for ingesting double-type data to all wind turbines in all wind farms, where you've defined property aliases on all turbine assets' properties. You would need to define a second, similar rule for ingesting 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.

```json
{
  "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+' WHERE type = 'double'",
  "description": "Sends data to the wind turbine asset property with the same alias as the topic",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
    { "iotSiteWise": { "putAssetPropertyValueEntries": [ { "propertyAlias": "${topic()}", "propertyValues": [ { "timestamp": { "timeInSeconds": "${timeInSeconds}" }, "value": { "doubleValue": "${value}" } ] } ] }, "roleArn": "arn:aws:iam::account-id:role/MySiteWiseActionRole" }
  ]
}
```
With this rule action, you can send the following message to a wind turbine property alias (for example, /company/windfarm/3/turbine/7/temperature) as a topic to ingest data.

```json
{
    "type": "double",
    "value": "38.3",
    "timeInSeconds": "1581368533"
}
```

**Example rule action that uses timestamp() to determine time**

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

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

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

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

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

```json
{
    "type": "double",
    "value": "38.3"
}
```
Configuring the AWS IoT SiteWise rule action

Configuring a rule that parses timestamp strings

If your device sends timestamp information in string format (for example, 2020-03-03T14:57:14.699Z), use the following steps to configure your rule action. To handle timestamp strings, you can create an AWS Lambda function that converts the timestamp from a string into **Time in seconds** (timeInSeconds) and **Offset in nanos** (offsetInNanos). Then, you can use `aws_lambda(functionArn, inputJson)` in your rule action parameters to invoke that Lambda function and use the output in your rule.

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

- Lambda functions. For more information, see Create a Lambda function with the console or Using Lambda with the AWS CLI in the AWS Lambda Developer Guide.
- AWS IoT rules with the AWS IoT SiteWise rule action. For more information, see Ingesting data using AWS IoT Core rules (p. 131).

**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, `ConvertTimestampFromString`).
   - **Runtime** – Use a Python 3 runtime, such as Python 3.7 (`python3.7`).
   - **Permissions** – Create a role with basic Lambda permissions (`AWSLambdaBasicExecutionRole`).
   - **Function code** – Use the following function code, which consumes a string argument named `timestamp` and outputs `timeInSeconds` and `offsetInNanos` values for that timestamp.

   ```python
   import datetime
   import json
   import math

   # Converts a timestamp string into timeInSeconds and offsetInNanos in Unix epoch time.
   def lambda_handler(event, context):
       timestamp_str = event['timestamp']
       dt = datetime.datetime.strptime(timestamp_str, '%Y-%m-%dT%H:%M:%S.%fZ')
       return {
           'timeInSeconds': math.floor(dt.timestamp()),
           'offsetInNanos': dt.microsecond * 1000
       }
   ```

   This function consumes timestamp strings that look like 2020-03-03T14:57:14.699Z. You can change the format string (`%Y-%m-%dT%H:%M:%S.%fZ`) in the function if needed. For more information, see strftime() and strptime() Format Codes in the Python 3 Documentation.

   **Note**
The `datetime` module supports only up to microsecond precision in timestamps. If your timestamp string has nanosecond precision, this function throws an error. To support nanosecond precision, modify the above function to use a library that supports nanoseconds (such as pandas), or remove the nanosecond portion of the string.

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.
Reducing costs with basic ingest

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

To use Basic Ingest, you send messages directly to a specific rule using a special topic, \$aws/rules/rule-name. For example, to send a message to a rule named SiteWiseWindFarmRule, you send a message to the topic \$aws/rules/SiteWiseWindFarmRule.

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

\$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 **Time in seconds** (timeInSeconds), use the following substitution template.

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

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

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

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

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

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

If your rule doesn't work as expected, see Troubleshooting an AWS IoT SiteWise rule action (p. 235).

**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 executed. Consider the pricing of each service if you apply this solution to your use case.
For more information, see Reducing messaging costs with basic ingest in the AWS IoT Developer Guide.

Troubleshooting the AWS IoT SiteWise rule action

To troubleshoot your AWS IoT SiteWise rule action in AWS IoT Core, you can configure CloudWatch Logs or you can configure a republish error action for your rule. For more information, see Troubleshooting an AWS IoT SiteWise rule action (p. 235).

Ingesting data from AWS IoT Events

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

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

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

Ingesting data using the AWS IoT SiteWise API

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

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

**Important**

The BatchPutAssetPropertyValue action is subject to quotas on the number of entries per request and the number of TQV data points per entry. AWS IoT SiteWise also rejects any data with a timestamp dated to more than 15 minutes in the past or more than 5 minutes in the future. For more information about these quotas, see BatchPutAssetPropertyValue in the AWS IoT SiteWise API Reference.

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

- The assetId and propertyId of the asset property that you are sending data to.
- The propertyAlias, which is a data stream alias (for example, /company/windfarm/3/turbine/7/temperature). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 152).

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

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

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

```json
{
```
Each entry in the payload contains an entryId that you can define as any unique string. If any request entries fail, each error will contain the entryId of the corresponding request so that you know which requests to retry.

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

- value – A structure that contains one of the following fields, depending on the type of the property being set:
  - booleanValue
  - doubleValue
  - integerValue
  - stringValue
- timestamp – A structure that contains the current Unix epoch time in seconds, timeInSeconds. You can also set the offsetInNanos key in the timestamp structure if you have temporally precise data. AWS IoT SiteWise rejects any data points with timestamps older than 15 minutes in the past or newer than 5 minutes in the future.
- quality – (Optional) One of the following quality strings:
  - GOOD – (Default) The data isn't affected by any issues.
  - BAD – The data is affected by an issue such as sensor failure.
  - UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.
Modeling industrial assets

You can create virtual representations of your industrial operation with AWS IoT SiteWise assets. An asset represents a device, a piece of equipment, or a process that uploads one or more data streams to the AWS Cloud, such as a wind turbine that sends air temperature, propeller rotation speed, and power output time-series measurements to asset properties in AWS IoT SiteWise. Each data stream corresponds to unique property alias. For example, the alias /company/windfarm/3/turbine/7/temperature uniquely identifies the temperature data stream coming from turbine #7 in wind farm #3. You can configure AWS IoT SiteWise assets to transform incoming measurement data using mathematical expressions, such as to convert temperature data from Celsius to Fahrenheit.

An asset can also represent a logical grouping of devices, such as an entire wind farm. You can associate assets to other assets to create asset hierarchies that represent complex industrial operations. Assets can access the data within their associated child assets so that you can use AWS IoT SiteWise expressions to calculate aggregate metrics, such as the net power output of a wind farm.

You must create every asset from an asset model. Asset models are declarative structures that standardize the format of your assets. Asset models enforce consistent information across multiple assets of the same type, so that you can process data in assets that represent groups of devices. In the preceding diagram, you use the same asset model for all three turbines because all turbines share a common set of properties.

After you define your asset models, you can create your industrial assets. To create an asset, select an active asset model to create an asset from that model. Then, you can populate asset-specific information such as data stream aliases and attributes. In the preceding diagram, you create three turbine assets from one asset model and then associate data stream aliases like /company/windfarm/3/turbine/7/temperature for each turbine.

You can also update and delete existing assets and asset models. When you update an asset model, every asset based on that asset model reflects any changes that you make to the underlying model.

Topics
- Asset and model states (p. 140)
- Defining asset models (p. 141)
- Creating assets (p. 151)
- Mapping industrial data streams to asset properties (p. 152)
- Updating attribute values (p. 152)
Asset and model states

When you create, update, or delete an asset or an asset model, the changes take time to propagate. AWS IoT SiteWise resolves these actions asynchronously and updates the status of each resource. Each asset and asset model has a status field that contains the state of the resource and any error message, if applicable. The state can be one of the following values:

- **ACTIVE** – The asset or asset model is active. This is the only state in which you can query and interact with assets and asset models.
- **CREATING** – The asset or asset model is being created.
- **UPDATING** – The asset or asset model is being updated.
- **DELETING** – The asset or asset model is being deleted.
- **PROPAGATING** – (Asset models only) The asset model's changes are propagating to all of its assets.
- **FAILED** – The asset or asset model failed to validate during a create or update action, possibly due to a circular reference in an expression. You can delete assets and asset models that are in the **FAILED** state.

Some of the create, update, and delete actions in AWS IoT SiteWise place an asset or asset model in a state other than **ACTIVE** while the action resolves. If you need to query or interact with an asset or asset model after you perform one of these actions, you must wait until the state changes to **ACTIVE**. Otherwise, your requests will fail.

To check the status of an asset, use the **DescribeAsset** action with the **assetId** parameter. In the response, look for the **state** value in the **assetStatus** object. This object looks like the following structure.

```json
{
  ...
  "assetStatus": {
    "state": "String",
    "error": {
      "code": "String",
      "message": "String"
    }
  }
}
```

To check the status of an asset model, use the **DescribeAssetModel** action with the **assetModelId** parameter. In the response, look for the **state** value in the **assetModelStatus** object. This object looks like the following structure.

```json
{
  ...
  "assetModelStatus": {
    "state": "String",
    "error": {
      "code": "String",
      "message": "String"
    }
  }
}
```
Defining asset models

AWS IoT SiteWise asset models drive standardization of your industrial data. An asset model contains a name, description, asset properties, and asset hierarchy definitions. For example, you can define a wind turbine model with temperature, rotations per minute (RPM), and power properties, and 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. Asset hierarchy definitions contain references to existing asset models. By following this approach, you can define asset hierarchies as you create your models.

You can use the AWS IoT SiteWise API to define asset models programmatically. Use the `CreateAssetModel` action to create an asset model based on the `AssetModelProperty` and `AssetModelHierarchyDefinition` types. Asset hierarchies are optional when you create an asset model.

The following example demonstrates how to create an asset model from a payload stored in a JSON file.

```
aws iotsitewise create-asset-model --cli-input-json file://model-payload.json
```

The payload in `model-payload.json` uses the following syntax.

```
{
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": Array of AssetModelProperty,
  "assetModelHierarchies": Array of AssetModelHierarchyDefinition
}
```

Give your asset model a unique, friendly name that you can use to easily identify it, such as *Wind Turbine* or *Wind Turbine Model*.

The `CreateAssetModel` action returns a structure that contains the unique `assetModelId` that you refer to when creating an asset. The response also contains the `assetModelStatus`, which contains a state which will initially be CREATING. The asset model’s status is CREATING until the changes propagate.

**Important**

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` action by specifying the `assetModelId`. After the asset model status is ACTIVE, you can use the asset model to create assets. For more information, see *Asset and model states* (p. 140).

The following sections describe the different types of asset properties and asset hierarchies. Each section also covers how you can define the respective `AssetModelProperty` or `AssetModelHierarchyDefinition` structures that you can include in the `CreateAssetModel` action payload.

**Topics**

- Asset properties (p. 142)
- Asset hierarchies (p. 150)
Asset 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.
- **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.
- **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. All transforms are **DOUBLE** type.
- **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 access associated assets' metric properties. All metrics are **DOUBLE** type.

Each asset property has a basic data type, which you must define as one of the following types:

- **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.

Transforms and metrics must have the **DOUBLE** data type.

You can also define a scientific unit for each asset property, such as mm or Celsius.

Topics

- Attributes (p. 142)
- Measurements (p. 143)
- Transforms (p. 143)
- Metrics (p. 144)
- Using formula expressions (p. 146)

For an example of how to use measurements, transforms, and metrics to calculate Overall Equipment Effectiveness (OEE), see Calculating OEE in AWS IoT SiteWise (p. 11).

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.

The **Attribute** structure contains an optional `defaultValue` field that you can define to create an attribute with a default value. To set an asset's attributes to a value other than the asset model's default value, see Updating attribute values (p. 152).

Example Example attribute structure

The following example demonstrates an attribute that represents an asset's region and defaults to **North America**. This structure is an example of an **AssetModelProperty** that contains an **Attribute**,
which you then specify as a part of the `CreateAssetModel` request payload to create an attribute property.

```json
{
  ...
  "assetModelProperties": [
    {
      "name": "Serial Number",
      "dataType": "STRING",
      "type": {
        "attribute": {
          "defaultValue": "BLT123"
        }
      }
    },
    ...
  ]
}
```

### Measurements

A measurement represents a device's raw sensor data stream, such as timestamped temperature values or timestamped rotations per minute (RPM) values.

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. To set an asset's measurement property to a device's sensor data stream, see the Mapping industrial data streams to asset properties (p. 152).

**Example Example measurement structure**

The following example demonstrates a measurement property that represents an asset's temperature sensor readings. This structure is an example of an `AssetModelProperty` that contains a `Measurement`, which you then specify as a part of the `CreateAssetModel` request payload to create a measurement property.

```json
{
  ...
  "assetModelProperties": [
    {
      "name": "Temperature_C",
      "dataType": "DOUBLE",
      "type": {
        "measurement": {}
      },
      "unit": "Celsius"
    }
  ],
  ...
}
```

### Transforms

Transforms are mathematical expressions that map an asset property's data points from one form to another. A transform expression consists of asset property variables, mathematical operators, and common functions. The transformed data points hold a one-to-one relationship with the input data points.

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 $\text{Temperature}_F = \frac{9}{5}\text{Temperature}_C + 32$. 

---

143
* 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 and available as the Temperature_F property within a few seconds.

The Transform structure has two components:

1. An expression string. For more information about the mathematical constructs that you can use to transform your data, see Using formula expressions (p. 146).

2. A list of variables that defines the other properties of your asset 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.

   **Important**
   Transforms can only input properties that are INTEGER or DOUBLE type. Transforms must input one non-attribute property and any number of attribute properties.

**Example transform structure**

The following example demonstrates a transform property that converts an asset's temperature measurement data from Celsius to Fahrenheit. This structure is an example of an AssetModelProperty that contains a Transform, which you then specify as a part of the CreateAssetModel request payload to create a transform property. Transforms must have a dataType of DOUBLE.

```json
{
  "assetModelProperties": [
    {
      "name": "Temperature_F",
      "dataType": "DOUBLE",
      "type": {
        "transform": {
          "expression": "9/5 * temp_c + 32",
          "variables": [
            {
              "name": "temp_c",
              "value": {
                "propertyId": "Temperature_C"
              }
            }
          ]
        }
      },
      "unit": "Fahrenheit"
    }
  ]
}
```

**Metrics**

Metrics are mathematical expressions that use aggregate 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.

The Metric structure has three components:
1. An **expression** string. Metric expressions can use the same mathematical constructs as transform expressions, and the AWS IoT SiteWise aggregation functions. For more information about the mathematical constructs that you can use to aggregate your data, see Using formula expressions (p. 146).

2. A **window** time interval. AWS IoT SiteWise supports the following tumbling window time intervals, where each interval starts when the previous one ends:
   - **1m** – 1 minute, computed at the end of each minute (12:00:00 AM, 12:01:00 AM, 12:02:00 AM, and so on).
   - **5m** – 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).
   - **15m** – 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).
   - **1h** – 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).
   - **1d** – 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).
   - **1w** – 1 week (7 days), computed at the end of every Sunday in UTC (every 12:00:00 AM Monday).

   When aggregating data points, the start of each interval is exclusive and the end of each interval is inclusive. AWS IoT SiteWise places the computed data point at the end of the interval.

3. A 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 input properties that are **INTEGER** or **DOUBLE** type. If you define any metric input variables in a metric's expression, those inputs must have the same **window** as the output metric.

### Example metric structure

The following example demonstrates a metric property that aggregates an asset's temperature measurement data to calculate average hourly temperature in Fahrenheit. This structure is an example of an **AssetModelProperty** that contains a **Metric**, which you then specify as a part of the CreateAssetModel request payload to create a metric property. Metrics must have a **dataType** of **DOUBLE**.

```json
{
  "assetModelProperties": [
    {
      "name": "Average_Hourly_Temperature_F",
      "dataType": "DOUBLE",
      "type": {
        "metric": {
          "expression": "avg(temp_f)",
          "variables": [
            {
              "name": "temp_f",
```
Using formula expressions

With formula expressions, you can define the mathematical functions to transform and aggregate your raw industrial data to gain insights about your operation. For more information about how to define properties that use formula expressions, see Transforms (p. 143) and Metrics (p. 144).

Topics

- Operators (p. 146)
- Common functions (p. 146)
- Relational functions (p. 147)
- Metric aggregation functions (p. 148)
- Undefined and infinite values (p. 150)

Operators

You can use the following common mathematical operators in your expressions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Adds the left and right operands.</td>
</tr>
<tr>
<td>−</td>
<td>Subtracts the right operand from the left operand.</td>
</tr>
<tr>
<td>/</td>
<td>Divides the left operand by the right operand.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies the left and right operands.</td>
</tr>
<tr>
<td>^</td>
<td>Raises the left operand to the power of the right operand (exponentiation).</td>
</tr>
<tr>
<td>%</td>
<td>Returns the remainder from dividing the left operand by the right operand (modulo). The result has the same sign as the left operand.</td>
</tr>
</tbody>
</table>

Common functions

You can use the following functions to calculate common mathematical functions.
### 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.
exp(x) | Returns e to the power of x.
floor(x) | Returns the nearest integer less than x.
log(x) | Returns the \( \log_e \) (base e) of x.
log10(x) | Returns the \( \log_{10} \) (base 10) of x.
log2(x) | Returns the \( \log_2 \) (base 2) of x.
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.
signum(x) | Returns the sign of x ( -1 for negative inputs, 0 for zero inputs, +1 for positive inputs).

### Relational functions

In transform properties (p. 143) only, you can use the following relational functions to compare two values and output 1 (true) or 0 (false).

| Function | Description |
---|---|
gt(x, y) | Returns 1 if x is greater than y, otherwise 0 ( \( x > y \)).
gte(x, y) | Returns 1 if x is greater than or equal to y, otherwise 0 ( \( x \geq y \)).
eq(x, y) | Returns 1 if x is equal to y, otherwise 0 ( \( x == y \)).
lt(x, y) | Returns 1 if x is less than y, otherwise 0 ( \( x < y \)).
lte(x, y) | Returns 1 if x is less than or equal to y, otherwise 0 ( \( x \leq y \)).
### Metric aggregation functions

In metric properties (p. 144) only, you can use the following functions that aggregate input values over each time interval and calculate a single output value. Some aggregation functions can’t aggregate data from associated assets.

<table>
<thead>
<tr>
<th>Function</th>
<th>Supports hierarchies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{avg}(x_0, \ldots, x_n) )</td>
<td>Yes</td>
<td>Returns the mean of the values of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{sum}(x_0, \ldots, x_n) )</td>
<td>Yes</td>
<td>Returns the sum of the values of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{min}(x_0, \ldots, x_n) )</td>
<td>Yes</td>
<td>Returns the minimum value of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{max}(x_0, \ldots, x_n) )</td>
<td>Yes</td>
<td>Returns the maximum value of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{count}(x_0, \ldots, x_n) )</td>
<td>Yes</td>
<td>Returns the total number of data points for the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{first}(x_0, \ldots, x_n) )</td>
<td>No</td>
<td>Returns the value with the earliest timestamp of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{last}(x_0, \ldots, x_n) )</td>
<td>No</td>
<td>Returns the value with the latest timestamp of the given variables over the current time interval.</td>
</tr>
<tr>
<td>( \text{earliest}(x_0, \ldots, x_n) )</td>
<td>No</td>
<td>Returns the value with the earliest timestamp of the given variables, not including the current time interval. The ( \text{earliest} ) function doesn't support metric properties as input variables.</td>
</tr>
<tr>
<td>( \text{latest}(x_0, \ldots, x_n) )</td>
<td>No</td>
<td>Returns the value with the latest timestamp of the given variables, not limited to the current time interval. The ( \text{latest} ) function doesn't support metric properties as input variables.</td>
</tr>
<tr>
<td>( \text{statetime}(x_0, \ldots, x_n) )</td>
<td>No</td>
<td>Returns the amount of time in seconds that the given variables are 1 (true) over the current</td>
</tr>
<tr>
<td>Function</td>
<td>Supports hierarchies</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>time interval. You can use the relational functions (p. 147) 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 information, see the example statetime scenario (p. 149). The statetime function doesn't support metric properties as input variables.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>2:00:00 PM</th>
<th>2:00:30 PM</th>
<th>2:01:15 PM</th>
<th>2:02:45 PM</th>
<th>2:04:00 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AWS IoT SiteWise calculates the Idle Time property every minute from the values of Idle. After this calculation completes, the Idle Time property has the following data points.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>2:00:00 PM</th>
<th>2:01:00 PM</th>
<th>2:02:00 PM</th>
<th>2:03:00 PM</th>
<th>2:04:00 PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle Time</td>
<td>N/A</td>
<td>30</td>
<td>60</td>
<td>45</td>
<td>0</td>
</tr>
</tbody>
</table>

AWS IoT SiteWise performs the following calculations for Idle Time at the end of each minute.

- At 2:00 PM (for 1:59 PM to 2:00 PM)
  - There is no data for Idle before 2:00 PM, so no data point is calculated.
- At 2:01 PM (for 2:00 PM to 2:01 PM)
  - At 2:00:00 PM, the machine is active (Idle is 0).
  - At 2:00:30 PM, the machine is idle (Idle is 1).
  - Idle doesn't change again before the end of the interval at 2:01:00 PM, so Idle Time is 30 seconds.
### Asset hierarchies

You can define asset hierarchies to create logical representations of your industrial operations. For example, you can define a wind farm composed of several windmill assets or an assembly line comprising multiple machine assets. Parent assets' metric expressions can access child assets' metric property data, so you can calculate statistics that provide insight to your operation or a subset of your operation.

To create an asset hierarchy, you must have an asset model for a child asset. To do so, see Defining asset models (p. 141). The CreateAssetModel action returns an `assetModelId` that uniquely identifies each asset model. Use this identifier to define a child asset hierarchy when you create a parent asset model.

Use the following payload when you create the parent asset model with `CreateAssetModel`.

```json
{
   "assetModelName": "String",
   "assetModelDescription": "String",
   "assetModelProperties": [
   ...
   ],
   "assetModelHierarchies": [
   {
      "name": "name for your hierarchy",
      "childAssetModelId": "the child asset model ID"
   },
   ...
   ]
}
```

Give your hierarchy definition a friendly name that you can use to easily identify it, such as *Wind Turbines*.

After you create your parent and child assets, you can associate them. To associate and disassociate your assets, see Associating and disassociating assets (p. 154).
Creating assets

You can create an asset from an asset model. If you haven’t already created an asset model, see Defining asset models (p. 141). You must have an assetModelId to create an asset. If you created an asset model but don’t know its assetModelId, you can use the ListAssetModels action to view all of your asset models.

Note
You can only create assets from ACTIVE models. If your model's state isn't ACTIVE, you may need to wait for up to a few minutes before you can create assets from that model. For more information, see Asset and model states (p. 140).

To create an asset from an asset model, use the CreateAsset action. Specify the following parameters:

- assetName – The new asset's name. Give your asset a unique, friendly name that you can use to easily identify it.
- assetModelId – The ID of the model to create the asset from.

The following example demonstrates how to create an asset from an asset model.

```
aws iotsitewise create-asset \
  --asset-name "Wind Turbine 7" \ 
  --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE
```

The action returns a response that contains your new asset's assetId and status in the following format.

```
{
  "assetId": "String",
  "assetArn": "String",
  "assetStatus": {
    "state": "String",
    "error": {
      "code": "String",
      "message": "String"
    }
  }
}
```

The action’s response also contains an assetStatus with a state string, which is initially CREATING. The asset’s status is CREATING until the changes propagate.

Important
The asset creation process can take up to a minute. To check your asset's status, use the DescribeAsset action with your asset's ID as the assetId parameter. After the assetStatus's state is ACTIVE, you can perform update actions on your asset. For more information, see Asset and model states (p. 140).

Finish configuring your asset with any the following optional actions:

- Mapping industrial data streams to asset properties (p. 152) if your asset has measurement properties.
- Updating attribute values (p. 152) if your asset has unique attribute values.
- Associating and disassociating assets (p. 154) if your asset is a parent asset.
Mapping industrial data streams to asset properties

If your asset has measurement properties, you can define the property aliases to map your data streams to those properties. You can also set property aliases so that you can easily identify an asset property when you ingest or retrieve asset data.

This step requires that you know your asset's assetId and the property's propertyId. If you created an asset but don't know its assetId, use the ListAssets action to view all of your assets for a specific model. Then, use the DescribeAsset action to view your asset's properties including property IDs. To learn how to use these actions, see Discovering your existing asset resources (p. 156). This step also requires that you know your data stream's path, such as /company/windfarm/3/turbine/7/temperature.

Note
Property aliases are unique within a Region and AWS account. AWS IoT SiteWise returns an error if you set a property alias to one that already exists on another asset property.

To map a data stream to your asset's property, use the UpdateAssetProperty action. Specify the following parameters:

- assetId – The asset's ID.
- propertyId – The asset property's ID.
- propertyAlias – The data stream's path to alias to the property.
- propertyNotificationState – The property value notification state: ENABLED or DISABLED. Specify the property's existing notification state when you update the property alias. If you omit this parameter, the new notification state is DISABLED. For more information about property notifications, see Interacting with other AWS services (p. 184).

The following example demonstrates how to set an asset property's alias that has notifications enabled.

```
aws iotsitewise update-asset-property \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --property-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
  --property-alias "/company/windfarm/3/turbine/7/temperature" \
  --property-notification-state ENABLED
```

If you view your asset in the AWS IoT SiteWise console or query your asset's data (p. 180), you should now see new data sent to your asset through your data stream alias.

Updating attribute values

Assets inherit the attributes of their asset model, including the default value of the attribute. In some cases, you will want to leave the asset model's default attribute, such as for an asset manufacturer property. In other cases, you will want to overwrite the inherited attribute, such as for an asset's latitude and longitude.

This step requires that you know your asset's assetId and the property's propertyId. If you created an asset but don't know its assetId, use the ListAssets action to view all of your assets for a specific model. Then, use the DescribeAsset action to view your asset's properties including property IDs. To learn how to use these actions, see Discovering your existing asset resources (p. 156).

Use the BatchPutAssetPropertyValue action to assign attribute values to your asset. You can use this action to set multiple attributes at once. This action's payload contains a list of entries, and each entry contains the asset ID, property ID, and attribute value.
The following example demonstrates how to set a wind turbine's latitude and longitude attributes 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` contains the following content.

```json
{
    "entries": [ 
        { 
            "entryId": "windfarm3-turbine7-latitude",
            "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "propertyId": "a1b2c3d4-5678-90ab-cdef-33333EXAMPLE",
            "propertyValues": [ 
                { 
                    "value": {  
                        "doubleValue": 47.6204  
                    },
                    "timestamp": {  
                        "timeInSeconds": 1575691200  
                    }
                }
            ]
        },
        { 
            "entryId": "windfarm3-turbine7-longitude",
            "assetId": "a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE",
            "propertyValues": [ 
                { 
                    "value": {  
                        "doubleValue": 122.3491  
                    },
                    "timestamp": {  
                        "timeInSeconds": 1575691200  
                    }
                }
            ]
        }
    ]
}
```

Each entry in the payload contains an `entryId` that you can define as any unique string. If any request entries fail, each error will contain the `entryId` of the corresponding request so that you know which requests to retry.

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

- **value** – A structure that contains one of the following fields, depending on the type of the property being set:
  - booleanValue
  - doubleValue
  - integerValue
  - stringValue

- **timestamp** – A structure that contains the current Unix epoch time in seconds, `timeInSeconds`. You can also set the `offsetInNanos` key in the `timestamp` structure if you have temporally precise data. AWS IoT SiteWise rejects any data points with timestamps older than 15 minutes in the past or newer than 5 minutes in the future.
• quality – (Optional) One of the following quality strings:
  • GOOD – (Default) The data isn't affected by any issues.
  • BAD – The data is affected by an issue such as sensor failure.
  • UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.

Associating and disassociating assets

If your asset's model defines any child asset model hierarchies, you can associate child assets to your asset. To create hierarchical asset models, see Asset hierarchies (p. 150).

After you have two assets, where one asset model refers to the other asset model, you can associate them to create the asset hierarchy. This step requires that you know the hierarchyId from the parent asset model to the child asset model. Use the DescribeAsset action to find the hierarchy ID in the response, which looks like the following structure.

```
{
... 
  "assetHierarchies": [ 
    { 
      "id": "hierarchy ID that you need", 
      "name": "hierarchy definition name"
    },
    ...
  }
}
```

Next, to associate a child asset to a parent asset, use the AssociateAssets action. Specify the following parameters:

• assetId – The parent asset's ID.
• hierarchyId – The hierarchy ID in the parent asset.
• childAssetId – The child asset's ID.

The following example demonstrates how to associate a child asset to a parent asset.

```
aws iotsitewise associate-assets \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --hierarchy-id a1b2c3d4-5678-90ab-cdef-44444EXAMPLE \
  --child-asset-id a1b2c3d4-5678-90ab-cdef-66666EXAMPLE
```

You can also disassociate two associated assets. Use the DisassociateAssets action with the same parameters as the previous example.

```
aws iotsitewise disassociate-assets \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --hierarchy-id a1b2c3d4-5678-90ab-cdef-44444EXAMPLE \
  --child-asset-id a1b2c3d4-5678-90ab-cdef-66666EXAMPLE
```

Updating assets and models

You can update your assets and asset models in AWS IoT SiteWise to modify their names and definitions. These update actions are asynchronous and take time to propagate through AWS IoT SiteWise. Check the
status of the asset or asset model before you make additional changes. You must wait until the changes propagate before you can use the resources.

### Updating assets

You can update an asset's name using the `UpdateAsset` action. Specify the following parameters:

- `assetId` – The asset's ID.
- `assetName` – The asset's new name.

The following example demonstrates how to update an asset's name.

```
aws iotsitewise update-asset \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --asset-name "Wind Turbine 8"
```

The asset's status is UPDATING until the changes propagate. For more information, see Asset and model states (p. 140).

### Updating asset models

You can update an asset model's name, description, properties, and hierarchies using the `UpdateAssetModel` action. Specify the following parameter:

- `assetModelId` – The asset model's ID.

Specify the updated asset model in the payload. To learn about the expected format of an asset model, see Defining asset models (p. 141). You can't change the type or data type of an existing property.

**Warning**

The `UpdateAssetModel` action overwrites the existing model with the model that you provide in the payload. To avoid deleting your asset model's properties or hierarchies, you must include their IDs and definitions in the updated asset model payload. To learn how to query your model's existing structure, see Describing a specific asset model (p. 157).

The following example demonstrates how to update an asset model.

```
aws iotsitewise update-asset-model \
  --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE \
  --cli-input-json file://model-payload.json
```

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 of those assets have the UPDATING state. You must wait until those assets return to the ACTIVE state before you interact with them. During this time, the updated asset model's status will be PROPAGATING. For more information, see Asset and model states (p. 140).

**Important**

If you remove a property from an asset model or update a property's formula expression, AWS IoT SiteWise deletes all previous data for that property. If you remove a hierarchy definition from an asset model, AWS IoT SiteWise disassociates all assets in that hierarchy.
Deleting assets and models

You can delete your assets and models from AWS IoT SiteWise when you're done with them. The delete actions are asynchronous and take time to propagate through AWS IoT SiteWise.

Deleting assets

You can delete your assets using the DeleteAsset action. Before you can delete an asset, you must first disassociate its child assets. You can use the DescribeAsset action with the assetId parameter to list your asset's hierarchies. Next, you can use the ListAssociatedAssets action with the assetId and hierarchyId parameters to find your asset's child assets. Then, to remove child assets from an asset, see Associating and disassociating assets (p. 154).

To use the DeleteAsset action, specify the following parameter:

- assetId – The asset's ID.

The following example demonstrates how to delete an asset.

```
aws iotsitewise delete-asset --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE
```

The asset's status will be DELETING until the changes propagate. For more information, see Asset and model states (p. 140). After the asset is deleted, you can't query that asset. If you do, the API returns an HTTP 404 response.

Important

AWS IoT SiteWise deletes all property data for deleted assets.

Deleting asset models

You can delete your asset models using the DeleteAssetModel action. Before you can delete an asset model, you must first delete all assets created from the asset model. You can use the ListAssets action with the assetModelId parameter to list all assets that use your asset model.

To use the DeleteAssetModel action, specify the following parameter:

- assetModelId – The asset model's ID.

The following example demonstrates how to delete an asset model.

```
aws iotsitewise delete-asset-model --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE
```

The asset model's status will be DELETING until the changes propagate. For more information, see Asset and model states (p. 140). After the asset model is deleted, you can't query that asset model. If you do, the API returns an HTTP 404 response.

Discovering your existing asset resources

You can explore your existing asset resources and discover the structure of your virtual industrial operation using the AWS IoT SiteWise API. Use these actions to find model IDs, asset IDs, property IDs, reference IDs, and other information about your assets and asset models. To explore asset property data, see Querying asset property values and aggregates (p. 180).
Listing all asset models

You can list all of your asset models using the `ListAssetModels` action.

You can specify any of the following optional parameters:

- `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 action.

The following example demonstrates how to list all asset models in the current AWS Region.

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

The action returns a response that contains your asset model summaries in the following format.

```json
{
   "assetModelSummaries": [
      {
         "id": "String",
         "arn": "String",
         "name": "String",
         "description": "String",
         "creationDate": Number,
         "lastUpdateDate": Number,
         "status": {
            "state": "String",
            "error": {
               "code": "String",
               "message": "String"
            }
         }
      },
      ...
   ],
   "nextToken": "String"
}
```

If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to `ListAssetModels`.

Describing a specific asset model

You can explore the structure of an individual asset model using the `DescribeAssetModel` action. Specify the following parameter:

- `assetModelId` – The asset model's ID.
The following example demonstrates how to query the structure of a specific asset model.

```
aws iotsitewise describe-asset-model --asset-model-id a1b2c3d4-5678-90ab-cdef-11111EXAMPLE
```

The action returns a response that contains the asset model structure in the following format.

```json
{
  "assetModelId": "String",
  "assetModelArn": "String",
  "assetModelName": "String",
  "assetModelDescription": "String",
  "assetModelProperties": [
    {
      "id": "String",
      "name": "String",
      "dataType": "String",
      "unit": "String",
      "type": {
        "attribute": {
          "defaultValue": "String"
        },
        "measurement": {},
        "transform": {
          "expression": "String",
          "variables": [
            {
              "name": "String",
              "value": {
                "propertyId": "String"
              }
            }
          ],
          "metric": {
            "expression": "String",
            "variables": [
              {
                "name": "String",
                "value": {
                  "propertyId": "String",
                  "hierarchyId": "String"
                }
              }
            ],
            "window": {
              "tumbling": {
                "interval": "String"
              }
            }
          }
        }
      }
    }
  ],
  "assetModelHierarchies": [
    {
      "id": "String",
      "name": "String",
      "childAssetModelId": "String"
    }
  ],
  "assetModelCreationDate": Number,
  "assetModelLastUpdateDate": Number,
  "assetModelStatus": {
    "state": "String",
  
```
Listing assets

You can list your existing assets using the ListAssets action.

Specify at least one of the following required parameters:

- **assetModelId** – The asset model ID to filter assets by. This parameter is required if you choose **ALL** for filter.
- **filter** – The asset filter: **ALL** or **TOP_LEVEL**. Defaults to **ALL**. If you choose **ALL**, you must specify an asset model ID to filter assets by.

You can also specify any of the following optional parameters:

- **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 action.

The following example lists all top-level assets in the current AWS Region.

```shell
aws iotsitewise list-assets --filter TOP_LEVEL
```

The action returns a response that contains your asset summaries in the following format.

```json
{
  "assetSummaries": [
    {
      "id": "String",
      "arn": "String",
      "name": "String",
      "assetModelId": "String",
      "creationDate": Number,
      "lastUpdateDate": Number,
      "status": {
        "state": "String",
        "error": {
          "code": "String",
          "message": "String"
        }
      },
      "hierarchies": [
        {
          "id": "String",
          "name": "String"
        }
      ],
      "nextToken": "String"
    }
  ],
  "nextToken": "String"
}
```

If more value entries exist, you can pass the pagination token from the **nextToken** field to a subsequent call to ListAssets.
Describing a specific asset

You can explore an individual asset using the DescribeAsset action. Specify the following parameter:

- **assetId** – The asset’s ID.

The following example demonstrates how to query the structure of a specific asset.

```bash
aws iotsitewise describe-asset --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE
```

The action returns a response that contains the asset summary in the following format.

```json
{
  "assetId": "String",
  "assetArn": "String",
  "assetName": "string",
  "assetModelId": "String",
  "assetProperties": [
    {
      "id": "String",
      "name": "String",
      "alias": "String",
      "notification": {
        "topic": "String",
        "state": "String"
      },
      "dataType": "String",
      "unit": "String"
    }
  ],
  "assetHierarchies": [
    {
      "id": "String",
      "name": "String"
    }
  ],
  "assetCreationDate": Number,
  "assetLastUpdateDate": Number,
  "assetStatus": {
    "state": "String",
    "error": {
      "code": "String",
      "message": "String"
    }
  }
}
```

Listing assets associated to a specific asset

You can list assets associated to a parent asset in a specific hierarchy using the ListAssociatedAssets action.

Specify the following required parameters:

- **assetId** – The parent asset’s ID.
- **hierarchyId** – The hierarchy ID to query in the parent asset. To learn how to find your asset’s hierarchies, see Describing a specific asset (p. 160). The DescribeAsset action returns an array of assetHierarchies where each structure contains an id that is a hierarchyId.
You can also specify any of the following optional parameters:

- `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 action.

The following example demonstrates how to list all assets associated to a parent asset in a specific hierarchy.

```
aws iotsitewise list-associated-assets \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --hierarchy-id a1b2c3d4-5678-90ab-cdef-44444EXAMPLE
```

The action returns a response that contains the associated assets' summaries in the following format.

```
{
  "assetSummaries": [ 
    {
      "id": "String",
      "arn": "String",
      "name": "String",
      "assetModelId": "String",
      "creationDate": Number,
      "lastUpdateDate": Number,
      "status": {
        "state": "String",
        "error": {
          "code": "String",
          "message": "String"
        }
      },
      "hierarchies": [ 
        {
          "id": "String",
          "name": "String"
        }
      ]
    },
    "nextToken": "String"
  }
}
```

If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to `ListAssociatedAssets`.

### Describing a specific asset property

You can explore a specific asset property and that property's alias and notification status using the `DescribeAssetProperty` action. Specify the following parameters:

- `assetId` – The asset's ID.
- `propertyId` – The asset property's ID.

The following example demonstrates how to query the structure and status of a specific asset property.

```
aws iotsitewise describe-asset-property \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --property-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE
```
The action returns a response that contains the asset property information in the following format.

```json
{
    "assetId": "String",
    "assetName": "String",
    "assetModelId": "String",
    "assetProperty": {
        "id": "String",
        "name": "String",
        "alias": "String",
        "notification": {
            "state": "String",
            "topic": "String"
        },
        "dataType": "String",
        "unit": "String",
        "type": {
            "attribute": {
                "defaultValue": "String"
            },
            "measurement": {},
            "transform": {
                "expression": "String",
                "variables": [
                    {
                        "name": "String",
                        "value": {
                            "propertyId": "String"
                        }
                    }
                ]
            },
            "metric": {
                "expression": "String",
                "variables": [
                    {
                        "name": "String",
                        "value": {
                            "propertyId": "String",
                            "hierarchyId": "String"
                        }
                    }
                ],
                "window": {
                    "tumbling": {
                        "interval": "String"
                    }
                }
            }
        }
    }
}
```
Monitoring data with AWS IoT SiteWise Monitor

You can use AWS IoT SiteWise to monitor the data from your processes, devices, and equipment by creating SiteWise Monitor web portals. SiteWise Monitor is a feature of AWS IoT SiteWise that lets you 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 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 time frames to better understand operational data. Project viewers can only view dashboards in the projects to which they have access.

Depending on your organization, the same person might perform multiple roles.

The following image illustrates how these four roles interact in the SiteWise Monitor portal.
You can manage who has access to your data by using AWS Single Sign-On. Your portal users don't need to access your AWS account. They can sign in to SiteWise Monitor from a desktop or mobile browser using their corporate credentials or AWS SSO user credentials.

SiteWise Monitor concepts

To use SiteWise Monitor, you should be familiar with the following concepts:

portal

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

project

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

**asset**

When data is ingested into AWS IoT SiteWise from your industrial equipment, your devices, equipment, and processes are each represented as assets. Each asset has data associated with it. For example, a piece of equipment might have a serial number, a location, a make and model, and an install date. It might also have time series values for availability, performance, quality, temperature, pressure, and so on. The portal administrator assigns sets of assets to each project.

**dashboard**

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

**visualization**

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

---

**Getting started with AWS IoT SiteWise Monitor**

If you're the AWS administrator for your organization, you create portals from the AWS IoT SiteWise console. Complete the following steps to create a portal so that members of your organization can view your AWS IoT SiteWise data:

1. Enable AWS SSO if it's not already enabled
2. Configure and create a portal
3. Add portal administrators and send invitation emails
4. Add portal users

You complete these steps in the AWS IoT SiteWise console.

After you create a portal, the portal administrator can view your AWS IoT SiteWise assets and assign them to projects in the portal. Project owners can then create dashboards to visualize the properties of the assets that help project viewers understand how your devices, processes, and equipment are performing.

You can follow a tutorial that walks through the steps required to set up a portal with a project, dashboards, and multiple users for a specific scenario using wind farm data. For more information, see Visualizing and sharing wind farm data in AWS IoT SiteWise Monitor (p. 42).

**Creating a portal**

You create a SiteWise Monitor portal in the AWS IoT SiteWise console.

**To create a portal**

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose **Monitor, Getting started**.

![AWS IoT SiteWise](image)

3. Choose **Create Portal**.

![Create Portal](image)

Next, you must provide some basic information to configure your portal.

### Enabling AWS SSO

If you already enabled AWS SSO in the current AWS Region, you can skip to **Configuring your portal** (p. 168).

AWS SSO provides identity federation for SiteWise Monitor so that you can control access to your portals. With AWS SSO, your users sign in with their corporate email and password instead of an AWS account. For more information about AWS SSO, see the [AWS Single Sign-On User Guide](#).

**Important**

You can only configure AWS SSO in one AWS Region, and you can only create SiteWise Monitor portals in the Region where AWS SSO is configured. If you configured AWS SSO in another Region, do one of the following:

- Configure AWS IoT SiteWise and SiteWise Monitor in the same Region as AWS SSO.
• Delete the AWS SSO configuration from the AWS Single Sign-On console before completing the steps in this section.

In this procedure, SiteWise Monitor performs the following steps for you:

1. Enables AWS Organizations, a prerequisite for AWS SSO. For more information, see AWS Organizations User Guide.
2. Creates an AWS organization and sets your AWS account as the master account.
3. Enables AWS SSO in the current AWS Region.

To enable AWS SSO

1. Enter the Email address, First name, and Last name for the user that you want as your portal administrator. The given email address will receive an email to set a password for the new AWS SSO user. If you want to be the portal administrator, enter your email and name to create an AWS SSO identity to use with your portal. You can create more users later.

   For more information, see Manage identities in AWS SSO in the AWS Single Sign-On User Guide.

2. Choose Create User to create the portal administrator user and enable AWS SSO.

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

### Configuring your portal

Your users use portals to view your data. You can customize a portal’s name, description, branding, support contact email, and permissions.
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 that portal users can contact when they have an issue with the portal and need help to resolve it.
5. (Optional) Add tags for your portal. For more information, see Tagging your AWS IoT SiteWise resources (p. 231).
6. (Optional) Choose an existing service role for your portal. By default, SiteWise Monitor creates a new service role for each portal. This role allows your portal users to access your AWS IoT SiteWise resources. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 212).
7. Choose Create portal.

Note
At this point in the process, AWS IoT SiteWise creates your portal. If you close the console, you can finish the setup process by adding administrators and users. For more information, see Adding or removing portal administrators (p. 174). If you don't want to keep this portal, delete it so it doesn't consume resources. For more information, see Deleting a portal (p. 176).

A message appears when your portal is created.

Successfully created portal URL at https://s1b2c3d4-5678-90ab-cdef-1111EXAMPLE.app.iotsitewise.aws

Next, you must invite one or more portal administrators to the portal. So far, you created a portal but no one can access it.

Inviting administrators

To get started in your new portal, you must assign a portal administrator. The portal administrator creates projects, chooses project owners, and assigns assets to projects. Portal administrators can see all of your AWS IoT SiteWise assets.

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 haven't created users, navigate to the AWS Single Sign-On console to define at least one user that you can specify as the portal administrator. Alternatively, you can connect an external identity provider to AWS SSO. For more information, see the AWS Single Sign-On User Guide.
To invite administrators

1. Select the check boxes for the users that you want as your portal administrators. The users are added to the Selected users list.

   If you don't see the user that you want to add in the list, navigate to the AWS Single Sign-On console to add users to your identity store. You can then return to your portal configuration to add the user to the list of portal administrators. For more information about how to add users to AWS SSO, see Manage your AWS SSO directory.

2. (Optional) Choose Send invite to selected users. Your email client opens, and an invitation is populated in the message body, as shown in the following example.

   Hello,

   You have been invited to administer the example-factory-1 portal.

   As a portal administrator, you create projects and assign assets that represent your company's devices, equipment, and processes for those projects. You designate a project owner who can create dashboards and visualizations to help your company get insights into your operational data.

   To log in, use the following link (or paste it into your browser) and Amazon Single Sign-On (SSO) will prompt you for your corporate credentials (your work email and password) to authenticate your access.

   https://a1b2c3d4-5678-90ab-cdef-111111EXAMPLE-app.iotsitewise.aws

   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 account 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.

   You can change the list of portal administrators later. For more information, see Adding or removing portal administrators (p. 174).
Note
Because only a portal administrator can create projects and assign assets to them, your portal is of limited use until you specify at least one portal administrator.

As the last step, you add users who can access your new portal.

Adding portal users

You control which users have access to your portals. In each portal, the portal administrators create one or more projects and assign portal users as owners or viewers for each project. Each project owner can invite additional portal users to own or view the project.

If you see the user that you want to add in the Directory list, complete the following steps.

**To add portal users**

1. Choose users from the 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.

   If you don't see the user that you want to add in the list, navigate to the AWS Single Sign-On console to add users to your identity store. You can then return to your portal configuration to add the user to the list of portal users. For more information about how to add users to AWS SSO, see Manage your AWS SSO directory.

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 adding users, choose Add users.

Congratulations! You successfully created a portal, assigned portal administrators, and added 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.

If you need to make changes to the portal, see Administering your SiteWise Monitor portals (p. 173).

To get started in the portal, see Getting started in the SiteWise Monitor Application Guide.
Administering your SiteWise Monitor portals

You might need to update portal details, change administrators, or add users to your portals. This section explains how you can complete these basic administrative tasks for your SiteWise Monitor portals.

1. Sign in to the AWS IoT SiteWise console.
2. In the navigation pane, choose Monitor, Portals.
3. Choose a portal, and then choose View details (or choose the portal's Name).
4. You can perform any of the following administrative tasks:
   - Changing a portal's name, description, branding, support email, and permissions (p. 173)
   - Adding or removing portal administrators (p. 174)
   - Sending email invitations to portal administrators (p. 175)
   - Adding or removing portal users (p. 175)
   - Deleting a portal (p. 176)

For information about how to create a portal, see Getting started with AWS IoT SiteWise Monitor (p. 166).

Changing a portal's name, description, branding, support email, and permissions

You can change a portal's name, description, branding, support email, and permissions.

1. On the portal details page, in the Portal details section, choose Edit.
2. Update the Name, Description, Portal branding, Support contact email, or Permissions.
3. When you're finished, choose Save.

Adding or removing portal administrators

In a few steps, you can add or remove users as administrators for a portal.

To add portal administrators

1. On the portal details page, in the Portal administrators section, choose Assign users.
2. On the Assign administrators page, select the check boxes for the users, and then choose Assign administrators.

If you don't see the user that you want to add in the list, navigate to the AWS Single Sign-On console to add users to your identity store. You can then return to your portal configuration to add the user to the list of portal administrators. For more information about how to add users to AWS SSO, see Manage your AWS SSO directory.

To remove portal administrators

- On the portal details page, in the Portal administrators section, select the check box for each user to remove, and then choose Remove from portal.

Note
Leaving a portal without a portal administrator is not recommended.
Sending email invitations to portal administrators

You can send email invitations to portal administrators through their AWS SSO email address.

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, as shown in the following example.

```
Hello,

You have been invited to administer the example-factory-1 portal.

As a portal administrator, you create projects and assign assets that represent your company's devices, equipment, and processes for those projects. You designate a project owner who can create dashboards and visualizations to help your company get insights into your operational data.

To log in, use the following link (or paste it into your browser) and Amazon Single Sign-On (SSO) will prompt you for your corporate credentials (your work email and password) to authenticate your access.

https://a1b2c3d4-5678-90ab-cdef-111111EXAMPLE.app.iotsitewise.aws
```

You can customize the email before you send it to your portal administrators.

Adding or removing portal users

You choose which users have access to your portals. Portal users appear in the list of users within a SiteWise Monitor portal. From this list, portal administrators can add project owners, and project owners can add project viewers.

**Note**

Your portal administrators and portal users might contact you through a portal's support email if they need you to add or remove a user.

**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, and then choose **Assign Users**.
Deleting a portal

If you don't see the user that you want to add in the list, navigate to the AWS Single Sign-On console to add users to your identity store. You can then return to your portal configuration to add the user to the list of portal users. For more information about how to add users to AWS SSO, see Manage your AWS SSO directory.

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.

Deleting a portal

You might delete a portal if you created it for testing purposes or if you created a duplicate of a portal that already exists.

Note
You must first manually delete all dashboards and projects in a portal before you can delete a portal. For more information, see Deleting projects and Deleting dashboards in the SiteWise Monitor Application Guide.

1. On the portal details page, choose Delete.

   Important
   When you delete a portal, you lose all projects that the portal contains, and all dashboards in each project. This action can't be undone. Your asset data isn't affected.

2. In the Delete Portals dialog box, choose Remove Admins and Users.

   You must remove the administrators and users from a portal before you can delete it. If your portal doesn't have administrators or users, the button doesn't appear, and you can skip to the next step.
3. If you're sure that you want to delete the entire portal, enter `delete` in the field to confirm deletion.

4. Choose Delete.

Creating dashboards (AWS Command Line Interface)

When you define visualizations (or widgets) in dashboards using the AWS CLI, you must specify the following information in the `dashboardDefinition` JSON document. This definition is a parameter of the `CreateDashboard` and `UpdateDashboard` operations.

`widgets`

A list of widget definition structures that each contain the following information:

`type`

The type of widget. AWS IoT SiteWise provides the following widget types:

- `monitor-line-chart` – A line chart. For more information, see [Line charts](#) in the [SiteWise Monitor Application Guide](#).
- `monitor-bar-chart` – A bar chart. For more information, see [Bar charts](#) in the [SiteWise Monitor Application Guide](#).
- `monitor-kpi` – A key performance indicator (KPI) visualization. For more information, see [KPI visualizations](#) in the [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.
properties
(Optional) A dictionary of properties for the widget. The members of this structure are context-dependent. Currently, AWS IoT SiteWise doesn't provide widgets that use properties.

Example Example dashboard definition

The following example defines a dashboard from a payload stored in a JSON file.

```bash
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.
• A bar chart that visualizes wind speed for four turbines in the upper right of the dashboard.

**Note**
This example represents the line and bar chart visualizations on a dashboard. This dashboard is similar to the example wind farm dashboard (p. 163).

```json
{
  "widgets": [
    {
      "type": "monitor-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"
        }
      ]
    },
    {
      "type": "monitor-bar-chart",
      "title": "Wind Speed",
      "x": 3,
      "y": 3,
      "height": 3,
      "width": 3,
      "metrics": [
        {
          "label": "Turbine 1",
          "type": "iotsitewise",
          "assetId": "a1b2c3d4-5678-90ab-cdef-2a2a2EXAMPLE",
          "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
        },
        {
          "label": "Turbine 2",
          "type": "iotsitewise",
          "assetId": "a1b2c3d4-5678-90ab-cdef-2b2b2EXAMPLE",
          "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
        },
        {
          "label": "Turbine 3",
          "type": "iotsitewise",
          "assetId": "a1b2c3d4-5678-90ab-cdef-2c2c2EXAMPLE",
          "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
        },
        {
          "label": "Turbine 4",
          "type": "iotsitewise",
          "assetId": "a1b2c3d4-5678-90ab-cdef-2d2d2EXAMPLE",
          "propertyId": "a1b2c3d4-5678-90ab-cdef-55555EXAMPLE"
        }
      ]
    }
  ]
}
```
Querying asset property values and aggregates

You can use the AWS IoT SiteWise API actions to query your asset properties' current values, historical values, and aggregated values over specific time intervals. You can use these features to gain quick insights or develop software solutions that integrate with the industrial data stored in your AWS IoT SiteWise assets.

You can also explore your asset data live in AWS IoT SiteWise Monitor. To learn how to configure SiteWise Monitor, see Monitoring data with AWS IoT SiteWise Monitor (p. 163).

The actions described in this section return property value objects that contain timestamp, quality, value (TQV) structures.

- The timestamp contains the current Unix epoch time in seconds with nanosecond offset.
- The quality contains one of the following strings that indicate the quality of the data point:
  - GOOD – The data isn't affected by any issues.
  - BAD – The data is affected by an issue such as sensor failure.
  - UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.
- The value contains one of the following fields, depending on the type of the property:
  - booleanValue
  - doubleValue
  - integerValue
  - stringValue

Topics
- Querying current asset property values (p. 180)
- Querying historical asset property values (p. 181)
- Querying aggregated asset property values (p. 182)

Querying current asset property values

Use the GetAssetPropertyValue action to query the current value of an asset property.

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

- The assetId and propertyId of the asset property that you are sending data to.
- The propertyAlias, which is a data stream alias (for example, /company/windfarm/3/turbine/7/temperature). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 152).

The following example demonstrates how to query a property's current value by assetId and propertyId.

```
aws iotsitewise get-asset-property-value \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
```
Querying historical asset property values

Use the `GetAssetPropertyValueHistory` action to query the historical values of an asset property.

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

- The `assetId` and `propertyId` of the asset property that you are sending data to.
- The `propertyAlias`, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 152).

You must also pass the following required parameters:

- `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 action.
- `timeOrdering` – The ordering to apply to the returned values: `ASCENDING` or `DESCENDING`.
- `qualities` – The quality to filter results by: `GOOD`, `BAD`, or `UNCERTAIN`

The following example demonstrates how to query a specific 10 minutes of a property's value history by `assetId` and `propertyId`.

```bash
aws iotsitewise get-asset-property-value-history \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --property-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
  --start-date 1575216000 \
  --end-date 1575216600
```
Querying aggregated asset property values

AWS IoT SiteWise provides aggregated asset property values, which are a set of metrics continuously calculated over specific time intervals.

Use the GetAssetPropertyAggregates action to query aggregates of an asset property.

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

- The `assetId` and `propertyId` of the asset property that you are sending data to.
- The `propertyAlias`, which is a data stream alias (for example, `/company/windfarm/3/turbine/7/temperature`). To use this option, you must first set your asset property's alias. To learn how to set property aliases, see Mapping industrial data streams to asset properties (p. 152).

You must also pass the following required parameters:

- `aggregateTypes` – The list of metrics to calculate. You can specify AVERAGE, COUNT, MAXIMUM, MINIMUM, and/or SUM.
- `resolution` – The time interval over which to calculate the metric: 1m (1 minute), 1h (1 hour), or 1d (1 day).
- `startDate` – The exclusive start of the range from which to query historical data, expressed in seconds in Unix epoch time.
- `endDate` – The inclusive end of the range from which to query historical data, expressed in seconds in Unix epoch time.

You can also pass any of the following parameters to refine your results:

- `maxResults` – The maximum number of results to return in one request. Defaults to 20 results.
- `nextToken` – A pagination token returned from a previous call of this action.
- `timeOrdering` – The ordering to apply to the returned values: ASCENDING or DESCENDING.
- `qualities` – The quality to filter results by: GOOD, BAD, or UNCERTAIN.

The action returns a response that contains the historical TQVs of the property in the following format.

```json
{
  "assetPropertyValueHistory": [
    {
      "value": {
        "booleanValue": Boolean,
        "doubleValue": Number,
        "integerValue": Number,
        "stringValue": "String"
      },
      "timestamp": {
        "timeInSeconds": Number,
        "offsetInNanos": Number
      },
      "quality": "String"
    }
  ],
  "nextToken": "String"
}
```

If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to the GetAssetPropertyValueHistory action.
The following example demonstrates how to query the count and average of a property's value by assetId and propertyId over a specific 1 hour interval.

```bash
aws iotsitewise get-asset-property-aggregates /
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --property-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
  --start-date 1575216000 \
  --end-date 1575219600 \
  --aggregate-types AVERAGE COUNT \
  --resolution 1h
```

The action returns a response that contains the historical TQVs of the property in the following format.

```json
{
  "aggregatedValues": [
    {
      "timestamp": Number,
      "quality": "String",
      "value": {
        "average": Number,
        "count": Number,
        "maximum": Number,
        "minimum": Number,
        "sum": Number
      }
    }
  ],
  "nextToken": "String"
}
```

**Note**

The `GetAssetPropertyAggregates` action returns a TQV with a different format than other actions 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.

If more value entries exist, you can pass the pagination token from the `nextToken` field to a subsequent call to the `GetAssetPropertyAggregates` action.

To define custom asset metrics into your asset models, see Metrics (p. 144). Metric properties offer additional aggregation functions and time intervals that aren't precomputed for the AWS IoT SiteWise API.
Interacting with other AWS services

AWS IoT SiteWise can publish asset data to the AWS IoT MQTT publish-subscribe message broker, so that you can interact with your asset data from other AWS services. AWS IoT SiteWise assigns each asset property a unique MQTT topic that you can use to route your asset data to other AWS services using AWS IoT Core rules. For example, you can configure AWS IoT Core rules to do the following tasks:

- Identify equipment failure and notify appropriate personnel by sending data to AWS IoT Events.
- Historize select asset data for use in external software solutions by sending data to Amazon DynamoDB.
- Generate weekly reports by triggering an AWS Lambda function.

You can follow a tutorial that walks through the steps required to set up a rule that stores property values in DynamoDB. For more information, see Publishing property value updates to Amazon DynamoDB (p. 85).

For more information about how to configure a rule, see Rules in the AWS IoT Developer Guide.

You can also consume data from other AWS services back into AWS IoT SiteWise. To ingest data through the AWS IoT SiteWise rule action, see Ingesting data using AWS IoT Core rules (p. 131).

Topics
- Understanding asset properties' MQTT topics (p. 184)
- Enabling asset property notifications (p. 184)
- Querying asset property notification messages (p. 185)

Understanding asset properties' MQTT topics

Every asset property has a unique MQTT topic path in the following format.

```
/aws/sitewise/asset-models/assetModelId/assets/assetId/properties/propertyId
```

Note

AWS IoT SiteWise doesn't support the # (multi-level) topic filter wildcard in the AWS IoT Core rules engine. You can use the + (single-level) wildcard. For example, you can use the following topic filter to match all updates for a particular asset model.

```
/aws/sitewise/asset-models/assetModelId/assets/+/properties/+ 
```

To learn more about topic filter wildcards, see Topics in the AWS IoT Core Developer Guide.

Enabling asset property notifications

By default, AWS IoT SiteWise doesn't publish property value updates. Use the UpdateAssetProperty action to enable MQTT message publishing for an asset property. You then pass the following parameters:

- assetId – The asset's ID.
Querying asset property notification messages

AWS IoT SiteWise User Guide

Querying asset property notification messages

AWS IoT SiteWise publishes asset property data updates to AWS IoT Core in the following format.

```json
{
  "type": "PropertyValueUpdate",
  "payload": {
    "assetId": "String",
    "propertyId": "String",
    "values": [
      {
        "timestamp": {
          "timeInSeconds": Number,
          "offsetInNanos": Number
        },
        "quality": "String",
        "value": {
          "booleanValue": Boolean,
          "doubleValue": Number,
          "integerValue": Number,
          "stringValue": "String"
        }
      }
    ]
  }
}
```

Each structure in the `values` list is a timestamp-quality-value (TQV) structure.

- The `timestamp` contains the current Unix epoch time in seconds with nanosecond offset.
- The `quality` contains one of the following strings that indicate the quality of the data point:
  - GOOD – The data isn’t affected by any issues.
  - BAD – The data is affected by an issue such as sensor failure.
  - UNCERTAIN – The data is affected by an issue such as sensor inaccuracy.

If you don’t know your resources’ IDs, you can use the `ListAssetModels`, `ListAssets`, and `DescribeAsset` actions to look up your AWS IoT SiteWise assets. To learn how to use these actions, see Discovering your existing asset resources (p. 156).

The following example demonstrates how to enable property value notifications for an asset property with an alias.

```bash
aws iotsitewise update-asset-property \
  --asset-id a1b2c3d4-5678-90ab-cdef-22222EXAMPLE \
  --property-id a1b2c3d4-5678-90ab-cdef-33333EXAMPLE \
  --property-notification-state ENABLED \
  --property-alias "/company/windfarm/3/turbine/7/temperature"
```

You can also disable MQTT message publishing by passing `--property-notification-state DISABLED`. 
The value contains one of the following fields, depending on the type of the property:

- booleanValue
- doubleValue
- integerValue
- stringValue

To parse values out of the `values` array, you need to use complex nested object queries in your rules' SQL statements. For more information, see Nested object queries in the AWS IoT Developer Guide, or see the Publishing property value updates to Amazon DynamoDB (p. 85) tutorial for a specific example of parsing asset property notification messages.

**Example Example query to extract the array of values**

The following statement demonstrates how to query the array of updated property values for a specific double-type property on all assets with that property.

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

The previous rule query statement outputs data in the following format.

```json
{
    "windspeed": [
        26.32020195042838,
        26.28584572975477,
        26.35266977372508,
        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.

```sql
SELECT
    get((SELECT VALUE (value.doubleValue) FROM payload.values), 0) AS windspeed
FROM
    '$aws/sitewise/asset-models/a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/assets/+/
    properties/a1b2c3d4-5678-90ab-cdef-33333EXAMPLE'
WHERE
    type = 'PropertyValueUpdate'
```

The previous rule query statement outputs data in the following format.

```json
{
}
```
"windspeed": 26.32020195042838
}

**Important**
This rule query statement ignores value updates other than the first in each batch. Each batch can contain up to 10 values. If you need to include the remaining values, you must set up a more complex solution to output asset property values to other services. For example, you can set up a rule with an AWS Lambda action to republish each value in the array to another topic, and set up another rule to query that topic and publish each value to the desired rule action.
Exporting data to Amazon Simple Storage Service

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 lets you create historical reports or analyze your data with complex methods.

AWS IoT SiteWise provides this feature as a AWS CloudFormation template. When you create a stack from the template, AWS CloudFormation creates the required AWS resources to stream incoming data from AWS IoT SiteWise to an S3 bucket.

Then, the S3 bucket receives all of your asset property data sent from AWS IoT SiteWise property value update messages. The S3 bucket also receives your asset metadata, which includes asset and property names and other information.

For more information about how to enable property value update messages for the asset properties to export to Amazon S3, see Interacting with other AWS services (p. 184).

This feature stores your asset property data and asset metadata in Amazon S3 in Apache Parquet format. Parquet is a columnar data format that saves space and enables faster queries compared to row-oriented formats like JSON.

Note
When this feature retrieves asset metadata, it supports up to approximately 5,000 asset properties. This limitation applies only to asset metadata. This limitation doesn't apply to the amount of properties supported when the feature exports asset property data.

Each resource's name includes a prefix that you can customize when you create the stack. Resources include the following:

- Amazon S3 buckets
- AWS Lambda functions
- AWS IoT Core rules
- AWS Identity and Access Management roles
- Amazon Kinesis Data Firehose streams
- AWS Glue databases

For a complete list, see Resources created from the template (p. 191).

Important
You will be charged for the resources that this AWS CloudFormation template creates and consumes. These charges include data storage and data transfer for multiple AWS services.

Topics
- Creating the AWS CloudFormation stack (p. 188)
- Viewing your data in Amazon S3 (p. 189)
- Resources created from the template (p. 191)

Creating the AWS CloudFormation stack

You can create a stack in AWS CloudFormation to export your asset data to Amazon S3.
To export data to Amazon S3

1. Open the AWS CloudFormation template and sign in to the AWS Management Console.
2. On the Create stack page, choose Next at the bottom of the page.
3. On the Specify stack details page, enter a BucketName for the S3 bucket that this template creates to receive asset data. This bucket name must be globally unique. For more information, see Rules for bucket naming in the Amazon Simple Storage Service Developer Guide.
4. (Optional) Change any of the template’s other parameters:
   - GlobalResourcePrefix – A prefix for names of global resources, such as IAM roles, created from this template.
   - LocalResourcePrefix – A prefix for names of resources created from this template in the current Region.

   **Note**
   If you create this template multiple times, you might need to change the bucket name and resource prefix parameters to avoid resource name conflicts.
5. Choose Next.
6. On the Configure stack options page, choose Next.
7. At the bottom of the page, select the check box that says I acknowledge that AWS CloudFormation might create IAM resources.
8. Choose Create stack.

   The stack takes a few minutes to create. If the stack fails to create, your account might have insufficient permissions, or you might have entered a bucket name that already exists. Use the following steps to delete the stack and try again:
   a. Choose Delete in the upper-right corner.
      
   **Note**
   AWS CloudFormation doesn’t delete S3 buckets or CloudWatch log groups. You can delete these resources in the consoles for those services.
   b. If the stack fails to delete, choose Delete again.
   c. If the stack fails to delete again, follow the steps in the AWS CloudFormation console to skip the resources that failed to delete, and try again.
9. After the AWS CloudFormation stack creates successfully, follow the next procedure to explore your asset property data in Amazon S3.

**Important**
After you create the stack, you can see the new resources in your AWS account. The feature might stop working correctly if you delete or modify these resources. We recommend that you don’t modify these resources unless you want to stop sending data to the bucket or want to customize this feature.

**Viewing your data in Amazon S3**

After you create the feature, you can view your asset property data and asset metadata in Amazon S3.

**Note**
Asset metadata updates every 6 hours. You may need to wait up to 6 hours to see asset metadata appear in the S3 bucket.
This feature stores asset property data in the following columns, where each row contains a data point:

- **type** – The type of property notification (PropertyValueUpdate).
- **asset_id** – The ID of the asset that received a data point.
- **asset_property_id** – The ID of the property that received a data point for the asset.
- **time_in_seconds** – The time at which the data was received, expressed in seconds in Unix epoch time.
- **offset_in_nanos** – The nanosecond offset from timeInSeconds.
- **asset_property_quality** – The quality of the data point: GOOD, UNCERTAIN, or BAD.
- **asset_property_value** – The value of the data point.
- **asset_property_data_type** – The data type of the asset property: boolean, double, integer, or string.

This feature stores asset metadata in the following columns, where each row contains an asset property:

- **asset_id** – The ID of the asset.
- **asset_name** – The name of the asset.
- **asset_model_id** – The ID of the asset's model.
- **asset_property_id** – The ID of the asset property.
- **asset_property_name** – The name of the asset property.
- **asset_property_data_type** – The data type of the asset property: BOOLEAN, DOUBLE, INTEGER, or STRING.
- **asset_property_unit** – The unit of the asset property.
- **asset_property_alias** – The alias of the asset property.

**To view your AWS IoT SiteWise data in Amazon S3**

1. Navigate to the Amazon S3 console.
2. From the list of buckets, choose the bucket with the name you chose when you created the template.
3. In the bucket, choose one of the following folders:
   - **asset-property-updates** – This folder contains asset property data exported from AWS IoT SiteWise.
   - **asset-metadata** – This folder contains asset details exported from AWS IoT SiteWise.
4. Choose the object that you want to view.
5. On the object's page, do the following:
   a. Choose the Select from tab.
      In this panel, you can preview records from Parquet files.
   b. For File format, choose Parquet.
   c. Choose Show file preview to show the contents of the file in JSON format.

**Note**
If new data doesn't appear in the bucket, check that you enabled property value update notifications for your asset properties. For more information, see Interacting with other AWS services (p. 184).

After you have your asset property data in Amazon S3, you can use several AWS services to generate reports or analyze and query your data:
• Run SQL queries on your data using Amazon Athena.
• Perform big data analysis using Amazon EMR.
• Search and analyze your data using Amazon Elasticsearch Service.

You can find other AWS services that can interact with your data in Amazon S3 listed under Analytics in the AWS Management Console.

**Note**
The stack creates an AWS Glue database to format asset property data. You can’t query this database for asset data. To query asset data from the S3 bucket with Athena, you can create a new AWS Glue database and configure crawlers to create AWS Glue tables. For more information, see Defining crawlers in the AWS Glue Developer Guide.

Resources created from the template

When you create a stack from the template, AWS CloudFormation creates the following resources. Most resources' names include a prefix that you can customize when you create the stack.

**Resource name parameters**

• BucketName – The name of the S3 bucket created from this template that receives asset data.
• GlobalResourcePrefix – A prefix for names of global resources created from this template. Defaults to sitewise-export-to-s3.
• LocalResourcePrefix – A prefix for names of resources created from this template in the current Region. Defaults to sitewise_export_to_s3.

**Resources created by the AWS CloudFormation template**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3 bucket for processed data</td>
<td>This bucket contains two folders. One folder receives the flattened, formatted data from the Kinesis Data Firehose delivery stream, and the other folder receives asset metadata.</td>
<td><code>${BucketName}</code></td>
</tr>
<tr>
<td>AWS Glue database</td>
<td>This database contains the AWS Glue table that this stack creates.</td>
<td><code>${LocalResourcePrefix}_firehose_glue_database</code></td>
</tr>
<tr>
<td>AWS Glue table</td>
<td>The Kinesis Data Firehose delivery stream uses this table to format data to Parquet format.</td>
<td><code>${LocalResourcePrefix}_firehose_glue_table</code></td>
</tr>
<tr>
<td>AWS Lambda function that transforms data</td>
<td>This function flattens the array of values in property value notification messages sent from AWS IoT SiteWise.</td>
<td><code>${LocalResourcePrefix}_lambda_transform_function</code></td>
</tr>
<tr>
<td>IAM role for the transform Lambda function</td>
<td>This role allows Lambda to store execution logs for the transform function.</td>
<td><code>${GlobalResourcePrefix}-lambda-transform-role</code></td>
</tr>
<tr>
<td>Resource</td>
<td>Description</td>
<td>Name</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IAM policy for the transform Lambda function role</td>
<td>This policy allows Lambda to store execution logs for the transform function.</td>
<td><code>${GlobalResourcePrefix}-lambda-transform-policy</code></td>
</tr>
<tr>
<td>CloudWatch Logs log group for the transform function</td>
<td>This log group contains logs for the transform function.</td>
<td><code>/aws/lambda/${GlobalResourcePrefix}_lambda_transform_function</code></td>
</tr>
<tr>
<td>Lambda function that collects asset metadata</td>
<td>This function retrieves details about assets in AWS IoT SiteWise and stores the details in an Amazon S3 bucket that this stack creates.</td>
<td><code>${LocalResourcePrefix}_lambda_metadata_function</code></td>
</tr>
<tr>
<td>Lambda layer for the metadata function</td>
<td>This layer provides an AWS SDK that contains AWS IoT SiteWise actions that the metadata function uses.</td>
<td><code>${LocalResourcePrefix}_lambda_metadata_layer</code></td>
</tr>
<tr>
<td>IAM role for the metadata Lambda function</td>
<td>This role allows Lambda to retrieve details about assets in AWS IoT SiteWise.</td>
<td><code>${GlobalResourcePrefix}-lambda-metadata-role</code></td>
</tr>
<tr>
<td>IAM policy for the metadata Lambda function role</td>
<td>This policy allows Lambda to retrieve details about assets in AWS IoT SiteWise.</td>
<td><code>${GlobalResourcePrefix}-lambda-metadata-policy</code></td>
</tr>
<tr>
<td>EventBridge scheduled event for the metadata Lambda function</td>
<td>This scheduled event runs the metadata Lambda every 6 hours to update the asset metadata bucket.</td>
<td><code>${LocalResourcePrefix}-metadata-event</code></td>
</tr>
<tr>
<td>CloudWatch Logs log group for the metadata function</td>
<td>This log group contains logs for the metadata function.</td>
<td><code>/aws/lambda/${LocalResourcePrefix}_lambda_metadata_function</code></td>
</tr>
<tr>
<td>AWS IoT rule</td>
<td>This rule queries property value notification messages and sends asset data to an Amazon Kinesis Data Firehose delivery stream.</td>
<td><code>${LocalResourcePrefix}_iot_topic_rule</code></td>
</tr>
<tr>
<td>IAM role for the AWS IoT rule</td>
<td>This role allows AWS IoT to send data to the Kinesis Data Firehose delivery stream.</td>
<td><code>${GlobalResourcePrefix}-core-firehose-role</code></td>
</tr>
<tr>
<td>IAM policy for the AWS IoT rule role</td>
<td>This policy allows AWS IoT to send data to the Kinesis Data Firehose delivery stream.</td>
<td><code>${GlobalResourcePrefix}-core-firehose-policy</code></td>
</tr>
<tr>
<td>Kinesis Data Firehose delivery stream</td>
<td>This delivery stream consumes data from the AWS IoT rule, flattens the data with a Lambda function, and delivers the data to Amazon S3.</td>
<td><code>${LocalResourcePrefix}_firehose_delivery_stream</code></td>
</tr>
<tr>
<td>Resource</td>
<td>Description</td>
<td>Name</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IAM role for the delivery stream</strong></td>
<td>This role allows Kinesis Data Firehose to perform actions on the S3 bucket, AWS Glue table, Lambda functions, and CloudWatch Logs log group.</td>
<td><code>${GlobalResourcePrefix}-firehose-delivery-role</code></td>
</tr>
<tr>
<td><strong>CloudWatch Logs log group for the delivery stream</strong></td>
<td>This log group contains a log stream, S3 Delivery, that receives logs about the Kinesis Data Firehose delivery stream.</td>
<td><code>/aws/kinesisfirehose/${LocalResourcePrefix}_firehose_delivery_stream</code></td>
</tr>
</tbody>
</table>
Security in AWS IoT SiteWise

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS IoT SiteWise, see AWS services in scope by compliance program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS IoT SiteWise. The following topics show you how to configure AWS IoT SiteWise to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS IoT SiteWise resources.

Topics
- Data protection in AWS IoT SiteWise (p. 194)
- Identity and access management for AWS IoT SiteWise (p. 197)
- Logging and monitoring in AWS IoT SiteWise (p. 219)
- Compliance validation for AWS IoT SiteWise (p. 227)
- Resilience in AWS IoT SiteWise (p. 227)
- Infrastructure security in AWS IoT SiteWise (p. 228)
- Configuration and vulnerability analysis in AWS IoT SiteWise (p. 228)
- Security best practices for AWS IoT SiteWise (p. 229)

Data protection in AWS IoT SiteWise

AWS IoT SiteWise conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that 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.

• Set up API and user activity logging with AWS CloudTrail.
• Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a Name field. This includes when you work with AWS IoT SiteWise or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into AWS IoT SiteWise or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

Topics
• Data encryption (p. 195)
• Internetwork traffic privacy (p. 197)

Data encryption

Data encryption refers to protecting data while in-transit (as it travels to and from AWS IoT SiteWise, and between gateways and OPC-UA 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.

For more information about server-side encryption and client-side encryption, review the topics listed below.

Topics
• Encryption at rest (p. 195)
• Encryption in transit (p. 196)
• Key management (p. 197)

Encryption at rest

AWS IoT SiteWise stores your data in the AWS Cloud and on gateways.

Data at rest in the AWS Cloud

AWS IoT SiteWise stores data in other AWS services that encrypt data at rest by default.

Data at rest on gateways

AWS IoT SiteWise gateways store the following data on the local file system:
• OPC-UA source configuration information
• The set of OPC-UA data stream paths from connected OPC-UA sources
• Industrial data cached when the gateway loses connection to the internet
AWS IoT SiteWise gateways run on AWS IoT Greengrass. AWS IoT Greengrass relies on Unix file permissions and full-disk encryption (if enabled) to protect data at rest on the core. It's your responsibility to secure the file system and device.

However, AWS IoT Greengrass does encrypt local copies of your OPC-UA server secrets retrieved from Secrets Manager. For more information, see Secrets encryption in the AWS IoT Greengrass Developer Guide.

For more information about encryption at rest on AWS IoT Greengrass cores, see Encryption at rest in the AWS IoT Greengrass Developer Guide.

## Encryption in transit

AWS IoT SiteWise has three modes of communication where data is in transit:

- **Over the Internet (p. 196)** – Communication between local devices (including gateways) and AWS IoT SiteWise is encrypted.
- **Over the Local Network (p. 196)** – Communication between gateways and OPC-UA sources can be encrypted.
- **Between Components on Gateways (p. 196)** – Communication between AWS IoT Greengrass components on AWS IoT SiteWise gateways isn't encrypted.

### Data in transit over the internet

AWS IoT SiteWise uses Transport Layer Security (TLS) to encrypt all communication over the internet. All data sent to the AWS Cloud is sent over a TLS connection using MQTT or HTTPS protocols, so it's secure by default. Gateways, which run on AWS IoT Greengrass, and property value notifications use the AWS IoT transport security model. For more information, see Transport security in the AWS IoT Developer Guide.

### Data in transit over the local network

AWS IoT SiteWise gateways follow OPC-UA specifications for communication with local OPC-UA sources. It's your responsibility to configure your sources to use a message security mode that encrypts data in transit.

Gateways support the following message security modes to connect to local OPC-UA sources:

- Basic256 – Sign and encrypt
- Basic256 – Sign
- Basic128Rsa15 – Sign and encrypt
- Basic128Rsa15 – Sign
- None

If you choose a sign message security mode, data in transit between gateways and sources is signed but not encrypted. If you choose an encrypt message security mode, the data in transit between gateways and sources is signed and encrypted. For more information about configuring sources, see Adding the gateway to AWS IoT SiteWise (p. 114).

### Data in transit between local components on gateways

AWS IoT SiteWise gateways run on AWS IoT Greengrass, which doesn't encrypt data exchanged locally on the AWS IoT Greengrass core because the data doesn't leave the device. This includes communication between AWS IoT Greengrass components such as the AWS IoT SiteWise connector. For more information, see Data on the core device in the AWS IoT Greengrass Developer Guide.
Key management

AWS IoT SiteWise doesn't support customer-managed encryption keys for data encrypted in the AWS Cloud.

AWS IoT SiteWise gateways run on AWS IoT Greengrass, and AWS IoT Greengrass core devices use public and private keys to authenticate with the AWS Cloud and encrypt local secrets, such as OPC-UA authentication secrets. For more information, see Key management in the AWS IoT Greengrass Developer Guide.

Internetwork traffic privacy

Connections between AWS IoT SiteWise and on-premise applications, such as gateways, are secured over Transport Layer Security (TLS) connections. For more information, see Encryption in transit (p. 196).

AWS IoT SiteWise doesn't support connections between Availability Zones within a Region, connections between Regions, or connections between AWS accounts.

Identity and access management for AWS IoT SiteWise

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use AWS IoT SiteWise resources. IAM is an AWS service that you can use with no additional charge.

Topics
- Audience (p. 197)
- Authenticating with identities (p. 198)
- Managing access using policies (p. 199)
- How AWS IoT SiteWise works with IAM (p. 201)
- AWS IoT SiteWise identity-based policy examples (p. 207)
- Using service-linked roles for AWS IoT SiteWise (p. 210)
- Using service roles for AWS IoT SiteWise Monitor (p. 212)
- Troubleshooting AWS IoT SiteWise identity and access (p. 217)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in AWS IoT SiteWise.

Service user – If you use the AWS IoT SiteWise service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT SiteWise features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS IoT SiteWise, see Troubleshooting AWS IoT SiteWise identity and access (p. 217).

Service administrator – If you’re in charge of AWS IoT SiteWise resources at your company, you probably have full access to AWS IoT SiteWise. It’s your job to determine which AWS IoT SiteWise features and resources your employees should access. You must then submit requests to your IAM administrator to
change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT SiteWise, see How AWS IoT SiteWise works with IAM (p. 201).

IAM administrator – If you’re an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS IoT SiteWise. To view example AWS IoT SiteWise identity-based policies that you can use in IAM, see AWS IoT SiteWise identity-based policy examples (p. 207).

### Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

### AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

### IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but
roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the IAM User Guide.

IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.

- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the IAM User Guide.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the IAM User Guide.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) 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.
An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

**Identity-based policies**

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM Policies](#) in the [IAM User Guide](#).

Identity-based policies can be further categorized as **inline policies** or **managed policies**. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing Between Managed Policies and Inline Policies](#) in the [IAM User Guide](#).

**Resource-based policies**

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

**Access control lists (ACLs)**

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access Control List (ACL) Overview](#) in the [Amazon Simple Storage Service Developer Guide](#).

**Other policy types**

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity’s identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions Boundaries for IAM Entities](#) in the [IAM User Guide](#).

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account.
root user. For more information about Organizations and SCPs, see How SCPs Work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the IAM User Guide.

### Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the IAM User Guide.

### How AWS IoT SiteWise works with IAM

Before you use IAM to manage access to AWS IoT SiteWise, you should understand what IAM features are available to use with AWS IoT SiteWise.

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>Supported by AWS IoT SiteWise?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies with resource-level permissions (p. 202)</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource-based policies (p. 205)</td>
<td>No</td>
</tr>
<tr>
<td>Access control lists (ACLs) (p. 206)</td>
<td>No</td>
</tr>
<tr>
<td>Tags-based authorization (p. 206)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials (p. 206)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles (p. 206)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles (p. 206)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

To get a high-level view of how AWS IoT SiteWise and other AWS services work with IAM, see AWS services that work with IAM in the IAM User Guide.

### Contents

- **AWS IoT SiteWise identity-based policies (p. 202)
  - Actions (p. 202)
  - BatchPutAssetPropertyValue authorization (p. 202)
  - Resources (p. 203)
  - Condition keys (p. 203)
  - Examples (p. 205)
- **AWS IoT SiteWise resource-based policies (p. 205)
- Access control lists (ACLs) (p. 206)
- Authorization based on AWS IoT SiteWise tags (p. 206)
- AWS IoT SiteWise IAM roles (p. 206)
AWS IoT SiteWise identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. AWS IoT SiteWise supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the IAM User Guide.

Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in AWS IoT SiteWise use the following prefix before the action: `iotsitewise:`. For example, to grant someone permission to upload asset property data to AWS IoT SiteWise with the `BatchPutAssetPropertyValue` API operation, you include the `iotsitewise:BatchPutAssetPropertyValue` action in their policy. Policy statements must include either an Action or NotAction element. AWS IoT SiteWise defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
  "iotsitewise:action1",
  "iotsitewise:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action.

```
"Action": "iotsitewise:Describe*"
```

To see a list of AWS IoT SiteWise actions, see Actions Defined by AWS IoT SiteWise in the IAM User Guide.

BatchPutAssetPropertyValue authorization

AWS IoT SiteWise authorizes access to the BatchPutAssetPropertyValue action in an unusual way. For most actions, when you allow or deny access to an action, that action returns an error if permissions aren't granted. When you use `BatchPutAssetPropertyValue`, you can send multiple data entries to different assets and asset properties in a single API request, and AWS IoT SiteWise authorizes each data entry independently. For any individual entry that fails authorization in the request, AWS IoT SiteWise includes an AccessDeniedException in the returned list of errors. AWS IoT SiteWise receives the data for any entry that authorizes and succeeds, even if another entry in the same request fails.

**Important**

If one entry is denied permissions, all entries for the same asset are also denied. For example, consider a scenario where you allow access to a property Property1 for any asset using the propertyId condition key. If you send a BatchPutAssetPropertyValue request that contains entries for Asset1.Property1, Asset1.Property2, Asset2.Property1, and
Asset3.Property3, then the only entry that succeeds is Asset2.Property1. If you send those entries in separate BatchPutAssetPropertyValue requests, then Asset1.Property1 and Asset2.Property1 succeed.

Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

Each IAM policy statement applies to the resources that you specify using their ARNs. An ARN has the following general syntax.

```
arn:${Partition}:${Service}:${Region}:${Account}:${ResourceType}/${ResourcePath}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS service namespaces.

For example, to specify the asset with ID a1b2c3d4-5678-90ab-cdef-22222EXAMPLE in your statement, use the following ARN;

```
"Resource": "arn:aws:iotsitewise:region:123456789012:asset/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE"
```

To specify all assets that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:iotsitewise:region:123456789012:asset/*"
```

Some AWS IoT SiteWise actions, such as those for creating resources, can't be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
    "resource1",
    "resource2"
]
```

To see a list of AWS IoT SiteWise resource types and their ARNs, see Resources Defined by AWS IoT SiteWise in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by AWS IoT SiteWise.

Condition keys

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build 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.

Important
Many condition keys are specific to a resource, and some API actions use multiple resources. If you write a policy statement with a condition key, use the Resource element of the statement to specify the resource to which the condition key applies. If you don’t do so, the policy might prevent users from performing the action at all, because the condition check fails for the resources to which the condition key doesn’t apply. If you don’t want to specify a resource, or if you’ve written the Action element of your policy to include multiple API actions, then you must use the ...IfExists condition type to ensure that the condition key is ignored for resources that don’t use it. For more information, see ...IfExists conditions in the IAM User Guide.

AWS IoT SiteWise defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

AWS IoT SiteWise condition keys

<table>
<thead>
<tr>
<th>Condition key</th>
<th>Description</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotsitewise:assetHierarchyPath</td>
<td>The asset’s hierarchy path, which is a string of asset IDs each separated by a forward slash. Use this condition key to define permissions based on a subset of your hierarchy of all assets in your account.</td>
<td>String</td>
</tr>
<tr>
<td>Example value: /a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/a1b2c3d4-5678-90ab-cdef-66666EXAMPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iotsitewise:propertyId</td>
<td>The ID of an asset property. Use this condition key to define permissions based on a specified property of an asset model. This condition key applies to all assets of that model.</td>
<td>String</td>
</tr>
<tr>
<td>Example value: a1b2c3d4-5678-90ab-cdef-33333EXAMPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iotsitewise:childAssetId</td>
<td>The ID of an asset being associated as a child to another asset. Use this condition key to define permissions based on child assets. To define permissions based on parent assets, use the resource section of a policy statement.</td>
<td>String</td>
</tr>
<tr>
<td>Example value: a1b2c3d4-5678-90ab-cdef-66666EXAMPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition key</td>
<td>Description</td>
<td>Types</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>iotsitewise:group</td>
<td>The ID of an AWS SSO group when listing access policies. Use this condition key to define access policy permissions for an AWS SSO group. Example value: a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE</td>
<td>String, Null</td>
</tr>
<tr>
<td>iotsitewise:user</td>
<td>The ID of an AWS SSO user when listing access policies. Use this condition key to define access policy permissions for an AWS SSO user. Example value: a1b2c3d4e5-a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE</td>
<td>String, Null</td>
</tr>
<tr>
<td>iotsitewise:portal</td>
<td>The ID of a portal in an access policy. Use this condition key to define access policy permissions based on a portal. Example value: a1b2c3d4-5678-90ab-cdef-77777EXAMPLE</td>
<td>String, Null</td>
</tr>
<tr>
<td>iotsitewise:project</td>
<td>The ID of a project in an access policy, or the ID of a project for a dashboard. Use this condition key to define dashboard or access policy permissions based on a project. Example value: a1b2c3d4-5678-90ab-cdef-88888EXAMPLE</td>
<td>String, Null</td>
</tr>
</tbody>
</table>

To learn with which actions and resources you can use a condition key, see Actions Defined by AWS IoT SiteWise.

Examples

To view examples of AWS IoT SiteWise identity-based policies, see AWS IoT SiteWise identity-based policy examples (p. 207).

AWS IoT SiteWise resource-based policies

AWS IoT SiteWise doesn't support resource-based policies (p. 200).
**Access control lists (ACLs)**

AWS IoT SiteWise doesn't support ACLs (p. 200).

**Authorization based on AWS IoT SiteWise tags**

You can attach tags to AWS IoT SiteWise resources or pass tags in a request to AWS IoT SiteWise. To control access based on tags, you provide tag information in the condition element of a policy using the `iotsitewise:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging AWS IoT SiteWise resources, see Tagging your AWS IoT SiteWise resources (p. 231).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Viewing AWS IoT SiteWise assets based on tags (p. 209).

**AWS IoT SiteWise IAM roles**

An IAM role is an entity within your AWS account that has specific permissions.

**Using temporary credentials with AWS IoT SiteWise**

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

AWS IoT SiteWise supports using temporary credentials.

SiteWise Monitor supports federated users to access portals. When a user signs in to a portal, SiteWise Monitor generates a session policy that provides the following permissions:

- Read-only access to the assets and asset data in AWS IoT SiteWise in your account to which that portal's role provides access.
- Access to projects in that portal to which the user has administrator (project owner) or read-only (project viewer) access.

For more information about federated portal user permissions, see Using service-linked roles for AWS IoT SiteWise (p. 210).

**Service-linked roles**

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

AWS IoT SiteWise supports service-linked roles. For details about creating or managing AWS IoT SiteWise service-linked roles, see Using service-linked roles for AWS IoT SiteWise (p. 210).

**Service roles**

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

AWS IoT SiteWise uses a service role to allow SiteWise Monitor portal users to access some of your AWS IoT SiteWise resources on your behalf. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 212).
Choosing an IAM role in AWS IoT SiteWise

When you create a portal resource in AWS IoT SiteWise, you must choose a role to allow the federated users of your SiteWise Monitor portal to access AWS IoT SiteWise on your behalf. If you have previously created a service role, then AWS IoT SiteWise provides you with a list of roles to choose from. Otherwise, you can create a role with the required permissions when you create a portal. It’s important to choose a role that allows access to your assets and asset data. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 212).

AWS IoT SiteWise identity-based policy examples

By default, IAM users and roles don’t have permission to create or modify AWS IoT SiteWise resources. They also can’t perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the IAM User Guide.

Topics

• Policy best practices (p. 207)
• Using the AWS IoT SiteWise console (p. 208)
• Allowing users to view their own permissions (p. 208)
• Allowing users to ingest data to assets in one hierarchy (p. 208)
• Viewing AWS IoT SiteWise assets based on tags (p. 209)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete AWS IoT SiteWise resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

• Get Started Using AWS Managed Policies – To start using AWS IoT SiteWise quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the IAM User Guide.

• Grant Least Privilege – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the IAM User Guide.

• Enable MFA for Sensitive Operations – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

• Use Policy Conditions for Extra Security – To the extent that it’s practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the IAM User Guide.
Using the AWS IoT SiteWise console

To access the AWS IoT SiteWise console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT SiteWise resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won’t function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the AWS IoT SiteWise console, attach the AWSIoTSiteWiseConsoleFullAccess managed policy to those entities or define equivalent permissions for those entities. For more information, see Adding permissions to a user in the IAM User Guide.

You don’t need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you’re trying to perform.

Allowing users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ViewOwnUserInfo",
         "Effect": "Allow",
         "Action": [
            "iam:GetUserPolicy",
            "iam:ListGroupsForUser",
            "iam:ListAttachedUserPolicies",
            "iam:ListUserPolicies",
            "iam:GetUser"
         ],
         "Resource": ["arn:aws:iam::*:user/${aws:username}"
      },
      {
         "Sid": "NavigateInConsole",
         "Effect": "Allow",
         "Action": [
            "iam:GetGroupPolicy",
            "iam:GetPolicyVersion",
            "iam:GetPolicy",
            "iam:ListAttachedGroupPolicies",
            "iam:ListGroupPolicies",
            "iam:ListPolicyVersions",
            "iam:ListPolicies",
            "iam:ListUsers"
         ],
         "Resource": "*"
      }
   ]
}
```

Allowing users to ingest data to assets in one hierarchy

In this example, you want to grant an IAM user in your AWS account access to write data to all asset properties in a specific hierarchy of assets, starting from the root asset `a1b2c3d4-5678-90ab-cdef-22222EXAMPLE`. The policy grants the `iotsitewise:BatchPutAssetPropertyValue`
permission to the user. This policy uses the `iotsitewise:assetHierarchyPath` condition key to restrict access to assets whose hierarchy path matches the asset or its descendants.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PutAssetPropertyValuesForHierarchy",
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "arn:aws:iotsitewise:*:*:asset/**",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE",
            "/a1b2c3d4-5678-90ab-cdef-22222EXAMPLE/**
          ]
        }
      }
    }
  ]
}
```

**Viewing AWS IoT SiteWise assets based on tags**

You can use conditions in your identity-based policy to control access to AWS IoT SiteWise resources based on tags. This example shows how you might create a policy that allows viewing an asset. However, permission is granted only if the asset tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListAllAssets",
      "Effect": "Allow",
      "Action": [
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets"
      ],
      "Resource": "*
    },
    {
      "Sid": "DescribeAssetIfOwner",
      "Effect": "Allow",
      "Action": "iotsitewise:DescribeAsset",
      "Resource": "arn:aws:iotsitewise:*:*:asset/**",
      "Condition": {
        "StringEquals": {
          "iotsitewise:ResourceTag/Owner": "${aws:username}"
        }
      }
    }
  ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to view an AWS IoT SiteWise asset, the asset must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: `Condition` in the *IAM User Guide*. 

209
Using service-linked roles for AWS IoT SiteWise

AWS IoT SiteWise uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS IoT SiteWise. Service-linked roles are predefined by AWS IoT SiteWise and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS IoT SiteWise easier because you don't have to manually add the necessary permissions. AWS IoT SiteWise defines the permissions of its service-linked roles, and unless defined otherwise, only AWS IoT SiteWise can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy can’t be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS IoT SiteWise resources because you can’t inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS IoT SiteWise

AWS IoT SiteWise uses the service-linked role named AWSServiceRoleForIoTSiteWise – AWS IoT SiteWise uses this service-linked role to deploy gateways (which run on AWS IoT Greengrass) and perform logging.

The AWSServiceRoleForIoTSiteWise service-linked role trusts the following services to assume the role:

- iotsitewise.amazonaws.com

The role uses the following permissions policy to allow AWS IoT SiteWise to complete actions on other services' resources in your account:

```json
{"Version": "2012-10-17",
"Statement": [
  {
    "Action": [
      "greengrass:GetAssociatedRole",
      "greengrass:GetCoreDefinition",
      "greengrass:GetCoreDefinitionVersion",
      "greengrass:GetGroup",
      "greengrass:GetGroupVersion"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "logs:CreateLogGroup",
      "logs:DescribeLogGroups"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/iotsitewise*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "logs:CreateLogStream",
      "logs:DescribeLogStreams"
    ]
```
You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the IAM User Guide.

Creating a service-linked role for AWS IoT SiteWise

You don't need to manually create a service-linked role. When you perform any operation in the AWS IoT SiteWise console, AWS IoT SiteWise creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you perform any operation in the AWS IoT SiteWise console, AWS IoT SiteWise creates the service-linked role for you again.

You can also use the IAM console or API to create a service-linked role for AWS IoT SiteWise.

• To do so in the IAM console, create a service-linked role with the IoT SiteWise use case.
• To do so using the AWS CLI or IAM API, create a service-linked role with the iotsitewise.amazonaws.com service name.

For more information, see Creating a service-linked role in the IAM User Guide.

If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for AWS IoT SiteWise

AWS IoT SiteWise doesn't allow you to edit the AWSServiceRoleForIoTSiteWise service-linked role. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the IAM User Guide.

Deleting a service-linked role for AWS IoT SiteWise

If you no longer need a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that isn't actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note
If the AWS IoT SiteWise service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try again.

To delete AWS IoT SiteWise resources used by the AWSServiceRoleForIoTSiteWise

1. Disable logging for AWS IoT SiteWise. For more information, see Changing your logging level (console) (p. 221) or Changing your logging level (CLI) (p. 221).
2. Delete any active gateways.

To manually delete the service-linked role using IAM
Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForIoTSiteWise service-linked role. For more information, see Deleting a Service-Linked Role in the IAM User Guide.

Supported Regions for AWS IoT SiteWise service-linked roles

AWS IoT SiteWise supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS IoT SiteWise Endpoints and Quotas.

Using service roles for AWS IoT SiteWise Monitor

A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

To allow federated SiteWise Monitor portal users to access your AWS IoT SiteWise assets and asset data, 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 specific permissions (p. 212).

When you create a SiteWise Monitor portal, you must choose a role that allows users of that portal to access your AWS IoT SiteWise resources. The AWS IoT SiteWise console can create and configure the role for you. You can edit the role in IAM later. Your portal users will have issues using their SiteWise Monitor portals if you remove the required permissions from the role or delete the role.

**Note**

Portals created before April 29, 2020 didn't require service roles. If you created portals before this date, you must attach service roles to continue using them. To do so, navigate to the Portals page in the AWS IoT SiteWise console, and then choose Migrate all portals to use IAM roles.

The following sections describe how to create and manage the SiteWise Monitor service role in the AWS Management Console or the AWS Command Line Interface.

Contents

- Service role permissions for SiteWise Monitor (p. 212)
- Managing the SiteWise Monitor service role (console) (p. 213)
  - Finding a portal's service role (console) (p. 214)
  - Creating the SiteWise Monitor service role (console) (p. 214)
  - Changing a portal's service role (console) (p. 214)
- Managing the SiteWise Monitor service role (CLI) (p. 214)
  - Finding a portal's service role (CLI) (p. 215)
  - Creating the SiteWise Monitor service role (CLI) (p. 215)

Service role permissions for SiteWise Monitor

When you create a portal, AWS IoT SiteWise lets you create a role whose name starts with AWSSiteWiseMonitorServiceRole. This role allows federated SiteWise Monitor users to access your portal configuration, assets, and asset data.
The role trusts the following service to assume the role:

- monitor.iotsitewise.amazonaws.com

The role uses the following permissions policy, whose name starts with `AWSIoTSiteWiseMonitorServicePortalPolicy`, to allow SiteWise Monitor users to complete actions on resources in your account.

```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",
        "sso-directory:DescribeUsers"
      ],
      "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. Access policies define AWS SSO identities' level of access to your portals and projects. For more information about portal permissions and access policies, see Administering your SiteWise Monitor portals (p. 173) and CreateAccessPolicy.

**Managing the SiteWise Monitor service role (console)**

The AWS IoT SiteWise console makes it easy to manage the SiteWise Monitor service role for your portals. When you create a portal, the console checks if you have any existing roles that can be attached to that portal. If not, the console can create and configure a service role for you. For more information, see Creating the SiteWise Monitor service role (console) (p. 214).

**Topics**

- Finding a portal's service role (console) (p. 214)
Finding a portal's service role (console)

Use the following steps to find the service role attached to a SiteWise Monitor portal.

**To find a portal's service role**

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Portals**.
3. Choose the portal for which you want to find the service role.

   The role attached to the portal appears under **Permissions**, **Service role**.

Creating the SiteWise Monitor service role (console)

When you create a SiteWise Monitor portal, you can create a new service role for your portal. For more information, see **Creating a portal** (p. 166).

**To create a service role for an existing portal**

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 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**.

Changing a portal's service role (console)

Use the following procedure to choose a different SiteWise Monitor service role for a portal.

**To change a portal's service role**

1. Navigate to the AWS IoT SiteWise console.
2. In the left 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**.

Managing the SiteWise Monitor service role (CLI)

You can use the AWS CLI for the following portal service role management tasks:

Topics
Finding a portal’s service role (CLI) (p. 215)
Creating the SiteWise Monitor service role (CLI) (p. 215)

Finding a portal’s service role (CLI)

To find the service role attached to a SiteWise Monitor portal, run the following command to list all of your portals in the current Region.

```bash
aws iotsitewise list-portals
```

The action returns a response that contains your portal summaries in the following format.

```json
{
  "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.

Creating the SiteWise Monitor service role (CLI)

Use the following steps to create a new SiteWise Monitor service role.

**To create a SiteWise Monitor service role**

1. Create a role with a trust policy that allows SiteWise Monitor to assume the role. This example creates a role named `MySiteWiseMonitorPortalRole` from a trust policy stored in a JSON string.

   - **Linux, macOS, or Unix**
   ```bash
   aws iam create-role --role-name MySiteWiseMonitorPortalRole --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "monitor.iotsitewise.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }'
   ```

   - **Windows command prompt**
   ```bash
   aws iam create-role --role-name MySiteWiseMonitorPortalRole --assume-role-policy-document "{"Version"::"2012-10-17"},"Statement":[{"Effect":"Allow"},
   ```
Using service roles for SiteWise Monitor

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.

Create a file called portal-service-role-policy.json and copy the following JSON object into the file.

```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",
        "sso-directory:DescribeUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

To create a policy with the service role permissions, run the following command.

```
aws iam create-policy \
  --policy-name MySiteWiseMonitorPortalPolicy \
  --description "Allows access to portal and asset resources in AWS IoT SiteWise." \
  --policy-document file://portal-service-role-policy.json
```

Copy the policy ARN from the policy metadata in the output.

To attach the policy to the role, run the following command.

```
aws iam attach-role-policy \
  --role-name MySiteWiseMonitorPortalRole \
  --policy-arn arn:aws:iam::123456789012:policy/MySiteWiseMonitorPortalPolicy
```
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.

   ```bash
   aws iotsitewise describe-portal --portal-id portal-id
   ```

   The operation returns a response that contains the portal's details in the following format.

   ```json
   {
     "portalId": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
     "portalArn": "arn:aws:iotsitewise:region:account-id:portal/a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
     "portalName": "WindFarmPortal",
     "portalDescription": "A portal that contains wind farm projects for Example Corp.",
     "portalClientId": "E-1a2b3c4d5e6f_sn6tbgHVzLWVEXAMPLE",
     "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.

   ```bash
   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
   ```

Troubleshooting AWS IoT SiteWise identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT SiteWise and IAM.

**Topics**

- I am not authorized to perform an action in AWS IoT SiteWise (p. 218)
- I am not authorized to perform iam:PassRole (p. 218)
- I want to view my access keys (p. 218)
- I'm an administrator and want to allow others to access AWS IoT SiteWise (p. 219)
- I want to allow people outside of my AWS account to access my AWS IoT SiteWise resources (p. 219)
I am not authorized to perform an action in AWS IoT SiteWise

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about an asset but does not have iotsitewise:DescribeAsset permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iotsitewise:DescribeAsset on resource: a1b2c3d4-5678-90ab-cdef-22222EXAMPLE
```

In this case, Mateo asks his administrator to update his policies to allow him to access the asset resource with ID a1b2c3d4-5678-90ab-cdef-22222EXAMPLE using the iotsitewise:DescribeAsset action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to AWS IoT SiteWise.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in AWS IoT SiteWise. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the IAM User Guide.
I'm an administrator and want to allow others to access AWS IoT SiteWise

To allow others to access AWS IoT SiteWise, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS IoT SiteWise.

To get started right away, see Creating Your First IAM Delegated User and Group in the IAM User Guide.

I want to allow people outside of my AWS account to access my AWS IoT SiteWise resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS IoT SiteWise supports these features, see How AWS IoT SiteWise works with IAM (p. 201).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the IAM User Guide.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the IAM User Guide.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.

Logging and monitoring in AWS IoT SiteWise

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT SiteWise and your other AWS solutions. AWS IoT SiteWise supports the following monitoring tools to watch the service, report when something is wrong, and take automatic actions when appropriate:

- Amazon CloudWatch monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- Amazon CloudWatch Logs enables you to monitor, store, and access your log files from AWS IoT SiteWise gateways, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- AWS CloudTrail captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Topics
Monitoring AWS IoT SiteWise with Amazon CloudWatch Logs

You can configure AWS IoT SiteWise to log information to CloudWatch Logs to monitor and troubleshoot the service.

When you use the AWS IoT SiteWise console, AWS IoT SiteWise creates a service-linked role that allows the service to log information on your behalf. If you don't use the AWS IoT SiteWise console, you must create a service-linked role manually to receive logs. For more information, see Creating a service-linked role for AWS IoT SiteWise (p. 211).

By default, AWS IoT SiteWise doesn't log information to CloudWatch Logs. To enable logging, choose a logging level other than **Disabled** (OFF). AWS IoT SiteWise supports the following logging levels:

- **OFF** – Logging is disabled.
- **ERROR** – Errors are logged.
- **INFO** – Errors and informational messages are logged.

You can also configure gateways to log information to CloudWatch Logs through AWS IoT Greengrass. For more information, see Monitoring gateway logs (p. 221).

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 Troubleshooting an AWS IoT SiteWise rule action (p. 235).

Contents

- Managing logging in AWS IoT SiteWise (console) (p. 220)
  - Finding your logging level (console) (p. 220)
  - Changing your logging level (console) (p. 221)
- Managing logging in AWS IoT SiteWise (CLI) (p. 221)
  - Finding your logging level (CLI) (p. 221)
  - Changing your logging level (CLI) (p. 221)

Managing logging in AWS IoT SiteWise (console)

You can use the AWS IoT SiteWise console for the following logging configuration tasks.

**Topics**

- Finding your logging level (console) (p. 220)
- Changing your logging level (console) (p. 221)

Finding your logging level (console)

Use the following procedure to find your current logging level in the AWS IoT SiteWise console.
To find your current AWS IoT SiteWise logging level

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Logging options**.

   The current logging status appears under **Logging status**. If logging is enabled, the current logging level appears under **Level of verbosity**.

Changing your logging level (console)

Use the following procedure to change your logging level in the AWS IoT SiteWise console.

To change your AWS IoT SiteWise logging level

1. Navigate to the AWS IoT SiteWise console.
2. In the left navigation pane, choose **Logging options**.
3. Choose **Edit**.
4. Choose the **Level of verbosity** to enable.
5. Choose **Save**.

Managing logging in AWS IoT SiteWise (CLI)

You can use the AWS Command Line Interface (AWS CLI) for the following logging configuration tasks.

Topics

- Finding your logging level (CLI) (p. 221)
- Changing your logging level (CLI) (p. 221)

Finding your logging level (CLI)

Run the following command to find your current AWS IoT SiteWise logging level with the AWS CLI.

```
aws iotsitewise describe-logging-options
```

The action returns a response that contains your logging level in the following format.

```
{
    "loggingOptions": {
        "level": "String"
    }
}
```

Changing your logging level (CLI)

Run the following AWS CLI command to change your AWS IoT SiteWise logging level. Replace `logging-level` with the logging level you want.

```
aws iotsitewise put-logging-options --logging-options level=logging-level
```

Monitoring gateway logs

You can configure your gateway to log information to Amazon CloudWatch Logs or the local file system.
Topics
- Using Amazon CloudWatch Logs (p. 222)
- Using local file system logs (p. 223)

Using Amazon CloudWatch Logs

You can configure your gateway to send logs to CloudWatch Logs. You might follow this procedure if you want to use the AWS Management Console to view your gateway's log files.

To configure and access CloudWatch Logs

1. To configure CloudWatch Logs for your gateway, see Configure logging in the AWS IoT Greengrass Developer Guide.
2. Navigate to the CloudWatch console.
3. In the navigation pane, choose Log groups.
4. You can find the AWS IoT SiteWise connector logs in the following folders:
   - /aws/greengrass/Lambda/region/aws/swCollector – The logs for the gateway's Lambda function that collects data from the gateway's OPC-UA sources.
   - /aws/greengrass/Lambda/region/aws/swPublisher – The logs for the gateway's Lambda function that publishes OPC-UA data streams to AWS IoT SiteWise.

Choose the folder for the function to debug.

5. Choose a log stream that has a name that ends with the name of your AWS IoT Greengrass group. By default, CloudWatch displays the most recent log stream first.

6. To show recent log entries, choose the list in the upper-right corner labeled all with the date and time, choose Relative, and then choose 5 minutes to show logs from the last 5 minutes.
7. (Optional) To see fewer logs, you can choose 30s from the upper-right corner.
8. Scroll to the bottom of the log entries to show the most recent logs.

**Using local file system logs**

You can configure your gateway to store logs locally. You might follow this procedure if you want to use the command line or local software to view your gateway’s log files.

**To configure and access local file system logs**

1. To configure local file system logs on your gateway, see [Configure logging](aws-iot-sitewise-user-guide.html#configure-logging) in the *AWS IoT Greengrass Developer Guide*.

   **Note**
   You must have root permissions to read AWS IoT Greengrass logs on the file system.

2. On your gateway, run the following command to list all AWS IoT SiteWise connector log files and their file details.

   ```bash
   sudo ls -l /greengrass-root/ggc/var/log/user/region/aws
   ```

   You can find the most recent AWS IoT SiteWise connector logs in the following files:

   - `swCollector.log` – The most recent logs for the Lambda function that collects data from the gateway’s OPC-UA sources.
   - `swPublisher.log` – The most recent logs for the Lambda function that publishes OPC-UA data streams to AWS IoT SiteWise.

   You can view older logs in the files named `swCollector.log-timestamp` and `swPublisher.log-timestamp`.

3. Run the following command to view the publisher log file contents, for example.

   ```bash
   sudo tail -f swPublisher.log
   ```

   The `tail` command shows the end of a file, and the `-f` option shows new lines as they’re appended to the file.
Monitoring AWS IoT SiteWise with Amazon CloudWatch metrics

You can monitor AWS IoT SiteWise using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

AWS IoT SiteWise publishes the metrics and dimensions listed in the sections below to the AWS/IoTSiteWise namespace.

Tip
AWS IoT SiteWise publishes metrics on a one minute interval. When you view these metrics in graphs in the CloudWatch console, we recommend that you choose a Period of 1 minute. This lets you see the highest available resolution of your metric data.

Contents
• Gateway metrics (p. 224)

Gateway metrics

AWS IoT SiteWise publishes the following gateway metrics. All gateway metrics are published on a one minute interval.

Important
To receive gateway metrics, you must use at least version 6 of the AWS IoT SiteWise connector on your gateway. For more information, see AWS IoT SiteWise connector in the AWS IoT Greengrass Developer Guide.

Gateway metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway.Heartbeat</td>
<td>Generated every minute for each gateway (gatewayId) connected.</td>
</tr>
<tr>
<td>Gateway.PublishSuccessCount</td>
<td>The number of data points that a gateway (gatewayId) successfully published.</td>
</tr>
<tr>
<td>Gateway.PublishFailureCount</td>
<td>The number of data points that a gateway (gatewayId) failed to publish. This metric counts errors that result from the gateway's calls to the BatchPutAssetPropertyValue operation. For more information about troubleshooting gateways, see Troubleshooting an AWS IoT SiteWise gateway (p. 234).</td>
</tr>
<tr>
<td>Gateway.ProcessFailureCount</td>
<td>The number of data points that a gateway (gatewayId) failed to process. This metric count errors that occur between the gateway and the gateway's sources, including errors reported by sources. For more information about troubleshooting gateways,</td>
</tr>
</tbody>
</table>
Logging AWS IoT SiteWise API calls with AWS CloudTrail

AWS IoT SiteWise is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS IoT SiteWise. CloudTrail captures API calls for AWS IoT SiteWise as events. The calls captured include calls from the AWS IoT SiteWise console and code calls to the AWS IoT SiteWise API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS IoT SiteWise. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT SiteWise, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

AWS IoT SiteWise information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in AWS IoT SiteWise, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for AWS IoT SiteWise, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts
Most AWS IoT SiteWise actions are logged by CloudTrail and are documented in the AWS IoT SiteWise API Reference.

The following data plane actions aren't logged by CloudTrail:

- BatchPutAssetPropertyValue
- GetAssetPropertyValue
- GetAssetPropertyValueHistory
- GetAssetPropertyAggregates

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Example: AWS IoT SiteWise log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, 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 action.

```
{
  "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": "alb2c3d4-5678-90ab-cdef-11111EXAMPLE",
    "clientToken": "alb2c3d4-5678-90ab-cdef-00000EXAMPLE"
  }
}
```
Compliance validation for AWS IoT SiteWise

AWS IoT SiteWise is not in scope of any AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using AWS IoT SiteWise is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **Evaluating resources with rules** in the AWS Config Developer Guide – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS IoT SiteWise

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

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 Interacting with other AWS services (p. 184).

• You can use the AWS IoT SiteWise Get* APIs to retrieve and backup historical asset property data. For more information, see Querying historical asset property values (p. 181).

• 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 Discovering your existing asset resources (p. 156).

Infrastructure security in AWS IoT SiteWise

As a managed service, AWS IoT SiteWise is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS IoT SiteWise through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

AWS IoT SiteWise gateways, which run on AWS IoT Greengrass, use X.509 certificates and cryptographic keys to connect and authenticate to the AWS Cloud. For more information, see Device authentication and authorization for AWS IoT Greengrass in the AWS IoT Greengrass Developer Guide.

Configuration and vulnerability analysis in AWS IoT SiteWise

IoT fleets can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make fleet setup complex and error-prone. Because devices are often constrained in computational power, memory, and storage capabilities, this limits the use of encryption and other forms of security on the devices themselves. Also, devices often use software with known vulnerabilities. These factors make IoT fleets an attractive target for hackers and make it difficult to secure your device fleet on an ongoing basis.

AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. You can use AWS IoT Device Defender to analyze, audit, and monitor connected devices to detect abnormal behavior, and mitigate security risks. AWS IoT Device Defender can audit device fleets to ensure they adhere to security best practices and detect abnormal behavior on devices. This makes it possible to enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised. For more information, see AWS IoT Device Defender in the AWS IoT Developer Guide.

If you use AWS IoT SiteWise gateways to ingest data to the service, it’s your responsibility to configure and maintain your gateway’s environment. This responsibility includes upgrading to the latest versions of the gateway’s system software, AWS IoT Greengrass software, and the AWS IoT SiteWise connector. For more information, see Configure the AWS IoT Greengrass core in the AWS IoT Greengrass Developer Guide and Upgrading a connector (p. 129).
Security best practices for AWS IoT SiteWise

This topic contains security best practices for AWS IoT SiteWise.

Use authentication credentials on your OPC-UA servers

Require authentication credentials to connect to your OPC-UA servers. Consult the documentation for your servers to do so. Then, to allow your gateway to connect to your OPC-UA servers, add server authentication secrets to your gateway. For more information, see Configuring source authentication (p. 124).

Use encrypted communication modes for your OPC-UA servers

Choose an encrypted message security mode when you configure your OPC-UA sources for your gateway, so your industrial data is secure as it moves from your OPC-UA servers to the gateway. For more information, see Data in transit over the local network (p. 196) and Adding the gateway to AWS IoT SiteWise (p. 114).

Encrypt your gateway's file system

Encrypt and secure your gateway, so your industrial data is secure as it moves through the gateway. If your gateway has a hardware security module, you can configure AWS IoT Greengrass to secure your gateway. For more information, see Hardware security integration in the AWS IoT Greengrass Developer Guide. Otherwise, consult the documentation for your operating system to learn how to encrypt and secure your file system.

Grant SiteWise Monitor users minimum possible permissions

Follow the principle of least privilege by using the minimum set of access policy permissions for your AWS SSO users and groups.

- When you create a portal, define a role that allows the minimum set of assets needed for that portal. For more information, see Using service roles for AWS IoT SiteWise Monitor (p. 212).
- When you and your portal administrators create and share projects, use the minimum set of assets needed for that project.
- When a user or group no longer needs access to a portal or project, remove them from that resource. If that user or group is no longer applicable to your organization, delete that user or group from your identity store.

The least principle best practice also applies to IAM roles. For more information, see Policy best practices (p. 207).

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
Follow AWS IoT Greengrass security best practices

Follow AWS IoT Greengrass security best practices for your gateway. For more information, see Security best practices in the AWS IoT Greengrass Developer Guide.

See also

- Security best practices in the AWS IoT Developer Guide
- Ten security golden rules for IoT solutions
Tagging your AWS IoT SiteWise resources

With tags, you can organize and manage your resources in AWS IoT SiteWise. You can use tags to assign metadata to your resources, and you can use tags in IAM policies to define conditional access to your resources.

Using tags in AWS IoT SiteWise

You can use tags to categorize your AWS IoT SiteWise resources by purpose, owner, environment, or any other classification for your use case. When you have many resources of the same type, you can quickly identify a specific resource based on its tags.

Each tag consists of a key and an optional value, both of which you define. For example, you could define a set of tags for your asset models that helps you track them by the industrial processes to which assets of each model contribute. We recommend that you create a set of tag keys that meets your needs for each kind of resource. By using a consistent set of tag keys, you can more easily manage your resources.

Tagging with the AWS Management Console

The Tag Editor in the AWS Management Console provides a central, unified way for you to create and manage your tags for resources from all AWS services. For more information, see Tag Editor in the AWS Resource Groups User Guide.

Tagging with the AWS IoT SiteWise API

You can also work with tags by using the AWS IoT SiteWise API. Before you create tags, be aware of tagging restrictions. For more information, see Tag naming and usage conventions in the AWS General Reference.

- To add tags when you create a resource, define them in the `tags` property of the resource.
- To add tags to an existing resource, or to update tag values, use the TagResource action.
- To remove tags from a resource, use the UntagResource action.
- To retrieve the tags that are associated with a resource, use the ListTagsForResource action, 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 actions.

<table>
<thead>
<tr>
<th>Taggable AWS IoT SiteWise resources</th>
<th>Create action</th>
<th>Describe action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset model</td>
<td>CreateAssetModel</td>
<td>DescribeAssetModel</td>
</tr>
<tr>
<td>Asset</td>
<td>CreateAsset</td>
<td>DescribeAsset</td>
</tr>
</tbody>
</table>

231
Use the following actions to view and manage tags for resources that support tagging:

- **TagResource** – Adds tags to a resource, or updates an existing tag's value.
- **ListTagsForResource** – Lists the tags for a resource.
- **UntagResource** – Removes tags from a resource.

You can add or remove tags for a resource at any time. To change the value of a tag key, add a tag to the resource that defines the same key and the new value. The new value replaces the old value. You can set a value to an empty string, but you can't set a value to null.

When you delete a resource, tags that are associated with that resource are also deleted.

## Using tags with IAM policies

In your IAM policies, you can use resource tags to control user access and permissions. For example, policies can allow users to create only those resources that have a specific tag. Policies can also restrict users from creating or modifying resources that have certain tags.

**Note**
If you use tags to allow or deny users' access to resources, you should deny users the ability to add or remove those tags for the same resources. Otherwise, a user could circumvent your restrictions and gain access to a resource by modifying its tags.

You can use the following condition context keys and values in the `Condition` element (also called the `Condition` block) of a policy statement.

- `iotsitewise:ResourceTag/tag-key: tag-value`

  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

- Viewing AWS IoT SiteWise assets based on tags (p. 209)
Troubleshooting AWS IoT SiteWise

Follow the steps in this section to help troubleshoot and resolve issues with AWS IoT SiteWise.

Contents
• Troubleshooting an AWS IoT SiteWise gateway (p. 234)
  • Configuring and accessing AWS IoT SiteWise gateway logs (p. 234)
  • AWS IoT SiteWise doesn't receive data from OPC-UA servers (p. 235)
  • Common AWS IoT Greengrass issues (p. 235)
• Troubleshooting an AWS IoT SiteWise rule action (p. 235)
  • Configuring AWS IoT Core logs (p. 236)
  • Configuring a republish error action (p. 236)
  • Error: Member must be within 900 seconds before and 300 seconds after the current timestamp (p. 238)
  • Error: Property value does not match data type <type> (p. 238)
  • Error: User: <role-arn> is not authorized to perform: iotsitewise:BatchPutAssetPropertyValue on resource (p. 238)
  • Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: <role-arn> (p. 238)
  • Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates. (p. 239)

Troubleshooting an AWS IoT SiteWise gateway

AWS IoT SiteWise gateways run an AWS IoT Greengrass connector. You can configure your gateway to log connector events to CloudWatch and to your gateway's local filesystem. Then, you can view the log files associated with the connector to troubleshoot your gateway.

Topics
• Configuring and accessing AWS IoT SiteWise gateway logs (p. 234)
• AWS IoT SiteWise doesn't receive data from OPC-UA servers (p. 235)
• Common AWS IoT Greengrass issues (p. 235)

Configuring and accessing AWS IoT SiteWise gateway logs

Before you can view gateway logs, you must configure your gateway to send logs to Amazon CloudWatch Logs or store logs on the local file system.

• Use CloudWatch Logs if you want to use the AWS Management Console to view your gateway's log files. For more information, see Using Amazon CloudWatch Logs (p. 222).
• Use local file system logs if you want to use the command line or local software to view your gateway's log files. For more information, see Using local file system logs (p. 223).
AWS IoT SiteWise doesn't receive data from OPC-UA servers

If your AWS IoT SiteWise assets aren't receiving data sent by your OPC-UA servers, you can search your gateway's logs to troubleshoot issues. Issues can include any of the following:

- Incorrect property aliases. If your asset's property aliases don't exactly match your OPC-UA data streams, including any source prefix you defined, then AWS IoT SiteWise won't receive data for those asset properties.

  In the gateway logs, look for `WARN` level log entries that contain a `ResourceNotFoundException`. In these entries, you can find text such as `alias=/company/windfarm/3/turbine/7/temperature` to discover any data streams not received by AWS IoT SiteWise.

- Incorrect data types. If your asset's property data types don't match your OPC-UA data types, then AWS IoT SiteWise won't receive data for those asset properties. For example, if an OPC-UA stream has an integer data type, your corresponding asset property must be `INTEGER` data type. A `DOUBLE`-type asset property won't receive OPC-UA integer streams.

- Incorrect AWS IoT Greengrass role permissions. Verify that your gateway configuration meets all of the AWS IoT SiteWise connector requirements.

- Exceeding AWS IoT SiteWise service quotas. If your large or complex AWS IoT SiteWise operation isn't receiving all of your expected data, review the AWS IoT SiteWise quotas (p. 240) and verify that your setup doesn't exceed any service quotas.

Common AWS IoT Greengrass issues

To find solutions to many issues configuring or deploying your gateway on AWS IoT Greengrass, see Troubleshooting AWS IoT Greengrass in the AWS IoT Greengrass Developer Guide.

Troubleshooting an AWS IoT SiteWise rule action

To troubleshoot your AWS IoT SiteWise rule action in AWS IoT Core, you can do one of the following procedures:

- Configure CloudWatch Logs
- Configure a republish error action for your rule

Next, compare the error messages with the errors in this topic to troubleshoot your issue.

Topics

- Configuring AWS IoT Core logs (p. 236)
- Configuring a republish error action (p. 236)
- Error: Member must be within 900 seconds before and 300 seconds after the current timestamp (p. 238)
- Error: Property value does not match data type `<type>` (p. 238)
- Error: User: `<role-arn>` is not authorized to perform: iotsitewise:BatchPutAssetPropertyValue on resource (p. 238)
- Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: `<role-arn>` (p. 238)
- Info: No requests were sent. PutAssetPropertyValueEntries was empty after performing substitution templates. (p. 239)
Configuring AWS IoT Core logs

You can configure AWS IoT to log various levels of information to CloudWatch Logs.

To configure and access CloudWatch Logs

1. To configure logging for AWS IoT Core, see Monitoring with CloudWatch Logs in the AWS IoT Developer Guide.
2. Navigate to the CloudWatch console.
3. In the navigation pane, choose Log groups.
4. Choose the AWSIotLogs group.
5. Choose a recent log stream. By default, CloudWatch displays the most recent log stream first.
6. Choose a log entry to expand the log message. Your log entry might look like the following screenshot.

7. Compare the error messages with the errors in this topic to troubleshoot your issue.

Configuring a republish error action

You can configure an error action on your rule to handle error messages. In this procedure, you configure the republish rule action as an error action to view error messages in the MQTT test client.

**Note**
The republish error action outputs only the equivalent of ERROR level logs. If you want more verbose logs, you must configure CloudWatch Logs (p. 236).

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.

![Error action 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://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**.

![MQTT client screenshot]

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.
Error: Member must be within 900 seconds before and 300 seconds after the current timestamp

Your timestamp is older than 15 minutes 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 encounter 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 15 minutes 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 each entry has the correct Data type chosen.
- If you configure the rule from the API or 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 Mapping industrial data streams to asset properties (p. 152).
- 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 Granting AWS IoT the required access (p. 131).

Error: iot.amazonaws.com is unable to perform: sts:AssumeRole on resource: <role-arn>

Your IAM user isn't authorized to assume the role on your rule.

Check that your IAM user is allowed 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 doesn't 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*. 
AWS IoT SiteWise is available in the following AWS Regions:

- US East (N. Virginia) – us-east-1
- US West (Oregon) – us-west-2
- Europe (Frankfurt) – eu-central-1
- Europe (Ireland) – eu-west-1

The following tables describes quotas within AWS IoT SiteWise. For more information about quotas that can be changed, see AWS service quotas in the AWS General Reference.

To request a service quota increase, see Create a support case.

### Quotas for assets and asset models

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of asset models per Region per AWS account</td>
<td>100</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of assets per asset model</td>
<td>10,000</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of child assets per parent asset</td>
<td>100</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Depth of asset hierarchy tree</td>
<td>10</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of asset hierarchy definitions per asset model</td>
<td>10</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Number of properties per asset model</td>
<td>200</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of property variables per property formula expression</td>
<td>10</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Number of functions per property formula expression</td>
<td>10</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Depth of property tree per asset model</td>
<td>10</td>
<td>No</td>
<td>For example, a model with a transform property C that consumes a transform property B that consumes a measurement property A has a depth of 3.</td>
</tr>
<tr>
<td>Resource</td>
<td>Quota</td>
<td>Adjustable</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of asset models per hierarchy tree</td>
<td>20</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of directly dependent properties per asset model</td>
<td>20</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Number of dependent properties per asset model</td>
<td>30</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Request rate for model API actions and logging options</td>
<td>10 requests per second per Region per AWS account</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Request rate for asset API actions</td>
<td>30 requests per second per Region per AWS account</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Quotas for asset property data**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request rate for asset property data API actions</td>
<td>1,000 requests per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to API actions such as GetAssetPropertyValue and BatchPutAssetPropertyValue.</td>
</tr>
<tr>
<td>Rate of data points ingested per asset property</td>
<td>10 data points per second</td>
<td>No</td>
<td>Timestamp-quality-value (TQV) data points.</td>
</tr>
<tr>
<td>Rate of data points ingested</td>
<td>1,000 data points per second per Region per AWS account</td>
<td>Yes</td>
<td>Timestamp-quality-value (TQV) data points.</td>
</tr>
<tr>
<td>Rate of data points computed</td>
<td>10,000 data points per second per Region per AWS account</td>
<td>Yes</td>
<td>This quota applies to the number of timestamp-quality-value (TQV) data points output by transform and metric computations.</td>
</tr>
</tbody>
</table>
### Resource Quota

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of data points processed per metric computation</td>
<td>200,000</td>
<td>No</td>
<td>Timestamp-quality-value (TQV) data points.</td>
</tr>
</tbody>
</table>

### Quotas for AWS IoT SiteWise gateways

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of gateways per Region per AWS account</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of OPC-UA sources per gateway</td>
<td>100</td>
<td>No</td>
</tr>
</tbody>
</table>

### Quotas for AWS IoT SiteWise Monitor

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of portals per Region per AWS account</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of projects per portal</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of dashboards per project</td>
<td>100</td>
<td>No</td>
</tr>
<tr>
<td>Number of root assets per project</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Number of visualizations per dashboard</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>Number of metrics per dashboard visualization</td>
<td>5</td>
<td>No</td>
</tr>
</tbody>
</table>
### Document history for the AWS IoT SiteWise User Guide

The following table describes the documentation for this release of AWS IoT SiteWise.

- **API version:** 2019-12-02
- **Latest documentation update:** April 29, 2020

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT SiteWise connector version 6 released</td>
<td>Version 6 of the IoT SiteWise connector is available. This release adds support for CloudWatch metrics and automatic discovery of new OPC-UA tags. This means you don't need to restart your gateway when tags change for your OPC-UA sources. This version of the connector requires stream manager and AWS IoT Greengrass Core software v1.10.0 or higher.</td>
<td>April 29, 2020</td>
</tr>
</tbody>
</table>

- **AWS IoT SiteWise general availability**

AWS IoT SiteWise general availability feature release. You can now manage gateways with the API, add your logo to portals, view gateway metrics, and more.

- Added [a tutorial](#) (p. 13) that you can follow to configure alarms for your asset properties.
- Added the [Exporting data to Amazon S3](#) (p. 188) section with an AWS CloudFormation template that you can use to export new data values to an S3 bucket.
- Added the [Configuring data sources](#) (p. 116) section that improves gateway source documentation and includes the new gateway APIs.
- Added the [gateway metrics](#) (p. 224) section that describes the CloudWatch metrics that gateways publish.
• Added the Configuring a gateway on Amazon EC2 (p. 115) section with an AWS CloudFormation template that you can use to quickly configure gateway dependencies on an Amazon EC2 instance.

• Added the portal service roles (p. 212) section that describes the new permissions feature of SiteWise Monitor portals.

• Updated portal documentation (p. 168) for portal service roles and portal logos.

• Added the Tagging your AWS IoT SiteWise resources (p. 231) section.

• Updated the Creating dashboards (CLI) (p. 177) section for the new dashboard definition structure.

• Added the Security (p. 194) 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

Configuring a rule that parses timestamp strings

Added information about how to create a rule that parses timestamp strings into the timestamp format expected by AWS IoT SiteWise. 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 (p. 243)

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
<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Improved ingesting data using AWS IoT Core rules</strong></td>
<td>Added detailed information about how to use and how to troubleshoot the AWS IoT SiteWise rule action, which you can use to ingest data from MQTT messages through AWS IoT Core.</td>
<td>February 14, 2020</td>
</tr>
<tr>
<td><strong>IoT SiteWise connector version 5 released</strong></td>
<td>Version 5 of the IoT SiteWise connector is available. This release fixes a compatibility issue with AWS IoT Greengrass Core software v1.9.4.</td>
<td>February 12, 2020</td>
</tr>
<tr>
<td><strong>IoT SiteWise connector version 4 released</strong></td>
<td>Version 4 of the IoT SiteWise connector is available. This release fixes an issue with OPC-UA server reconnection.</td>
<td>February 7, 2020</td>
</tr>
<tr>
<td><strong>Restructured modeling industrial assets</strong></td>
<td>Restructured the Updating Assets and Models section into multiple topics within Modeling Industrial Assets.</td>
<td>February 4, 2020</td>
</tr>
<tr>
<td>- Asset and model states (p. 140)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Mapping industrial data streams to asset properties (p. 152)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Updating attribute values (p. 152)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Associating and disassociating assets (p. 154)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Updating assets and models (p. 154)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Deleting assets and models (p. 156)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ingesting data from AWS IoT things tutorial</strong></td>
<td>Added a tutorial that you can follow to learn how to configure an AWS IoT SiteWise rule action to ingest data from a new or existing fleet of AWS IoT things.</td>
<td>February 4, 2020</td>
</tr>
<tr>
<td><strong>Restructured retrieving data from AWS IoT SiteWise (p. 243)</strong></td>
<td>Restructured the Retrieving Data section into two top-level sections: Querying asset property values and aggregates and Interacting with other AWS services.</td>
<td>January 21, 2020</td>
</tr>
<tr>
<td><strong>Publishing property value updates to Amazon DynamoDB tutorial</strong></td>
<td>Added a tutorial that you can follow to learn how to use property value notifications to store asset data in DynamoDB.</td>
<td>January 8, 2020</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Using formula expressions</td>
<td>Added the formula expression reference to organize the constants and functions available for use in transform and metric properties. Restructured Asset properties into separate topics for each property type.</td>
<td>January 7, 2020</td>
</tr>
<tr>
<td>Using OPC-UA node filters</td>
<td>Added information about how to use OPC-UA node filters to improve gateway performance when adding gateway sources.</td>
<td>January 3, 2020</td>
</tr>
<tr>
<td>Upgrading a connector</td>
<td>Added information about how to upgrade a gateway when a new connector version is released.</td>
<td>December 30, 2019</td>
</tr>
<tr>
<td>IoT SiteWise connector version 3 released</td>
<td>Version 3 of the IoT SiteWise connector is available. This release removes the iot:* permissions requirement.</td>
<td>December 17, 2019</td>
</tr>
<tr>
<td>IoT SiteWise connector version 2 released</td>
<td>Version 2 of the IoT SiteWise connector is available. This release adds support for multiple OPC-UA secret resources.</td>
<td>December 10, 2019</td>
</tr>
<tr>
<td>Creating dashboards (AWS CLI)</td>
<td>Added information about how to create a dashboard in AWS IoT SiteWise Monitor using the AWS CLI.</td>
<td>December 6, 2019</td>
</tr>
<tr>
<td>Version Released</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| 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 (p. 139) section for changes to assets, asset models, and asset hierarchies.  
- Updated the data ingestion (p. 100) section to include AWS IoT Greengrass connector steps and non-gateway data ingestion sections.  
- Added the AWS IoT SiteWise Monitor (p. 163) section and a separate application guide that shows how to use the SiteWise Monitor web application.  
- Added Querying asset property values and aggregates (p. 180) and Interacting with other AWS services (p. 184) sections.  
- Rewrote the getting started (p. 7) 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 |
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.