aws

User Guide

AWS IoT TwinMaker

AWS IoT TwinMaker: User Guide

Table of Contents

What is AWS IoT TwinMaker?

AWS IoT TwinMaker is an AWS IoT service that you can use to build operational digital twins of physical and digital systems. AWS IoT TwinMaker creates digital visualizations using measurements and analysis from a variety of real-world sensors, cameras, and enterprise applications to help you keep track of your physical factory, building, or industrial plant. You can use this real-world data to monitor operations, diagnose and correct errors, and optimize operations.

A digital twin is a live digital representation of a system and all of its physical and digital components. It is dynamically updated with data to mimic the true structure, state, and behavior of the system. You can use it to drive business outcomes.

End users interact with data from your digital twin by using a user interface application.

How it works

To fulfill the minimum requirements for creating a digital twin, you must do the following.

- Model devices, equipment, spaces, and processes in a physical location.
- Connect these models to data sources that store important contextual information, such as sensor data camera feeds.
- Create visualizations that help users understand the data and insights in order to make business decisions more efficiently.
- Make digital twins available to end users to drive business outcomes.

AWS IoT TwinMaker addresses these challenges by providing the following capabilities.

- Entity component system knowledge graph: AWS IoT TwinMaker provides tools for modeling devices, equipment, spaces, and processes in a knowledge graph.

  This knowledge graph contains metadata about the system and can connect to data in different locations. AWS IoT TwinMaker provides built-in connectors for data stored in AWS IoT SiteWise and Kinesis Video Streams. You can also create custom connectors to data stored in other locations.

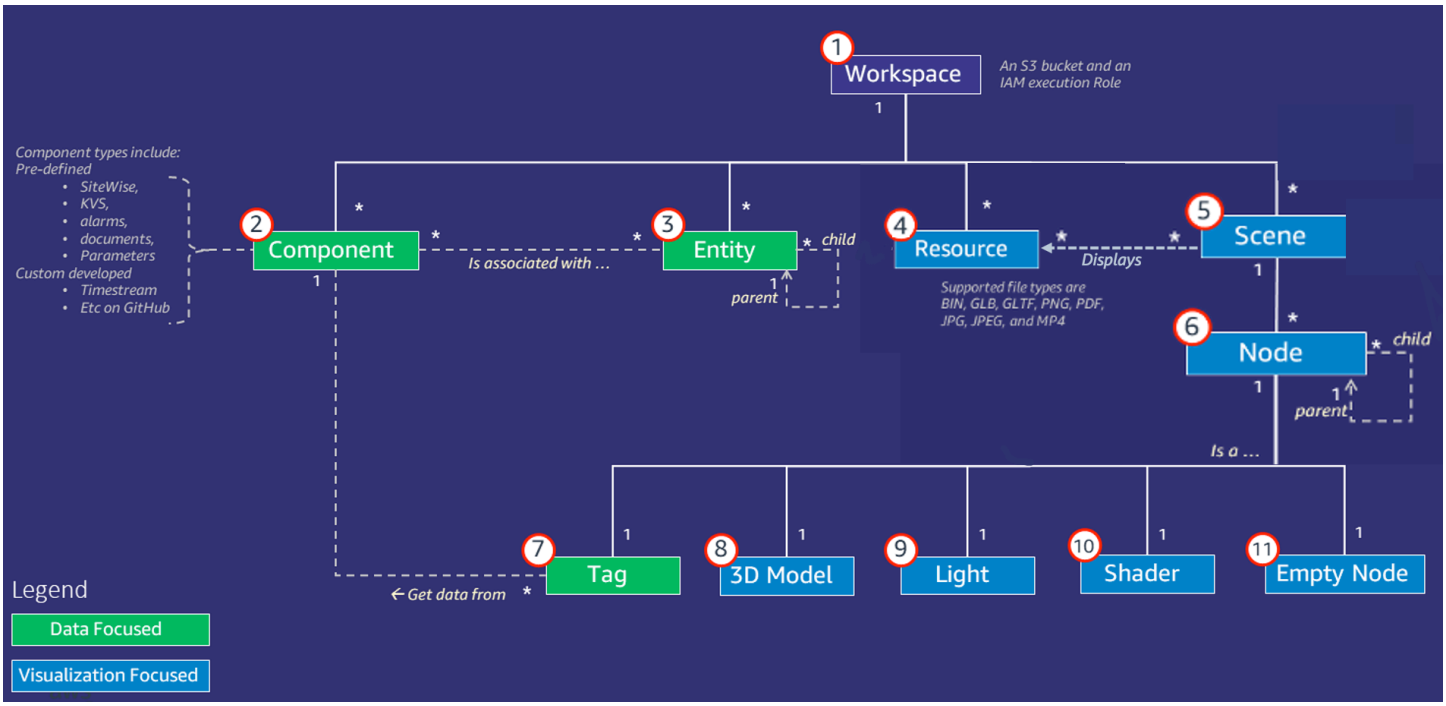
  The knowledge graph and connectors together provide a single interface for querying data in disparate locations.

- Scene composer: The AWS IoT TwinMaker console provides a scene composition tool for creating scenes in 3D. You upload your previously built 3D/CAD models, optimized for web display and converted to .gltf or .glb format. You then use the scene composer to place multiple models in a single scene, creating visual representations of their operations.

  You can also overlay data in the scene. For example, you can create a tag in a scene location that connects to temperature data from a sensor. This associates the data with the location.
- Applications: AWS IoT TwinMaker provides a plug-in for Grafana and Amazon Managed Grafana that you can use to build dashboard applications for end users.
- Third-party tools: Mendix partners with AWS IoT TwinMaker to provide complete solutions for industrial IoT. See the workshop Lean Daily Management Application with Mendix and AWS IoT TwinMaker to get started with using the Mendix Low Code Application Development Platform (LCAP) with AWS services like AWS IoT TwinMaker, Kinesis Video Streams and AWS IoT SiteWise.

Key concepts and components

The following diagram illustrates how the key concepts of AWS IoT TwinMaker fit together.



> **Note**
>
> Asterisks (*) in the diagram indicate one-to-many relationships. For the quotas for each of these relationships, see AWS IoT TwinMaker endpoints and quotas.

The following sections describe the concepts illustrated in the diagram.

Workspace

A workspace is a top-level container for your digital twin application. You create a logical set of entities, components, scene assets, and other resources for your digital twin inside this workspace. It also serves as a security boundary to manage access to the digital twin application and the resources it contains. Each workspace is linked to the Amazon S3 bucket where your workspace data is stored. You use IAM roles to restrict access to your workspace.

A workspace can contain multiple components, entities, scenes and resources. A component type, entity, scene or resource exists only within one workspace.

Entity-component model

AWS IoT TwinMaker provides tools that you use to model your system by using an entity-component-based knowledge graph. You can use the entity-component architecture to create a representation of your physical system. This entity component model consists of **entities**, **components**, and *relationships*. For more information about entity-component systems, see Entity component system.

Entity

Entities are digital representations of the elements in a digital twin that capture the capabilities of that element. This element can be a piece of physical equipment, a concept, or a process. Entities have **components** associated with them. These **components** provide data and context for the associated entity.

With AWS IoT TwinMaker, you can organize entities into custom hierarchies for more efficient management. The default view of the entity and component system is hierarchical.

Component

Components provide context and data for entities in a scene. You add components to entities. The lifetime of a component is tied to the lifetime of an entity.

Components can add static data, such as a list of documents or the coordinates of a geographic location. They can also have functions that connect to other systems, including systems that contain time series data such as AWS IoT SiteWise and other time-series cloud historians.

Components are defined by JSON documents that describe the connection between a data source and AWS IoT TwinMaker. Components can describe external data sources or data sources that are built in to AWS IoT TwinMaker. A component accesses an external datasource by using a Lambda function that is specified in the JSON document. A workspace can contain many components. Components provide data to tags through associated entities.

AWS IoT TwinMaker provides several built-in components that you can add from the console. You can also create your own custom components to connect to sources of data such as timestream telemetry and geospatial coordinates. Examples of these include TimeStream Telemetry, Geospatial components, and connectors to third party data sources such as Snowflake.

AWS IoT TwinMaker provides the following types of built-in components for common use cases:

- **Document**, such as user manuals or images located at specified URLs.
- **Time series**, such as sensor data from AWS IoT SiteWise.
- **Alarms**, such as time-series alarms from external data sources.
- **Video**, from IP cameras connected to Kinesis Video Streams.
- **Custom components** to connect to additional data sources. For example, you can create a custom connector to connect your AWS IoT TwinMaker entities to time-series data stored externally.

Data sources

A data source is the location of your digital twin's source data. AWS IoT TwinMaker supports two types of data sources:

- **Hierarchy connectors**, which allow you to continually sync an external model to AWS IoT TwinMaker.

- **Time-series connectors**, which allow you to connect to time-series databases such as AWS IoT SiteWise.

Property

Properties are the values, both static and time-series backed, contained in components. When you add components to entities, the properties in the component describe details about the current state of the entity.

AWS IoT TwinMaker supports three kinds of properties:

- **Single value, non-time-series properties**— These properties are typically static key-value pairs and are directly stored in AWS IoT TwinMaker with the metadata of the associated entity.
- **Time-series properties**— AWS IoT TwinMaker stores a reference to the time-series store for these properties. This defaults to the latest value.
- **Relationship properties**— These properties store a reference to another entity or component. For example, `seen_by` is a relationship component that might relate a camera entity to another entity that is directly visualized by that camera.

You can query property values across heterogeneous data sources by using the unified data query interface.

Visualization

You use AWS IoT TwinMaker to augment a three-dimensional representation of your digital twin, and then view it in Grafana. To create scenes, use existing CAD or other 3D file types. You then use data overlays to add relevant data for your digital twin.

Scenes

Scenes are three-dimensional representations that provide visual context for the data connected to AWS IoT TwinMaker. Scenes can be created by using a single gltf (GL Transmission Format) or glb 3D model for the entire environment, or by using a composition of multiple models. Scenes also include **tags** to denote points of interest in the scene.

Scenes are the top level containers for visualizations. A scene consists of one or more nodes.

A workspace can contain multiple scenes. For example, a workspace can contain one scene for each floor of a facility.

Resources

Scenes display resources, which are displayed as nodes in the AWS IoT TwinMaker console. A scene can contain many resources.

Resources are images and `glTF`-based, three-dimensional models used to create a scene. A resource can represent a single piece of equipment, or a complete site.

You place resources into a scene by uploading a .gltf or .glb file to your workspace resource library and then adding them to your scene.

Augmented user interface

With AWS IoT TwinMaker you can augment your scenes with data overlays that add important context and information, such as sensor data, to locations in the scene.

Nodes: Nodes are instances of tags, lights, and three-dimensional models. They can also be empty to add structure to your scene hierarchy. For example, you can group multiple nodes together under a single empty node.

Tags: A tag is a type of node that represents data from a component (through an entity). A tag can be associated with only one component. A tag is an annotation added to a specific `x,y,z` coordinate position of a scene. The tag connects this scene part to the knowledge graph by using an entity property. You can use a tag to configure the behavior or visual appearance of an item in the scene, such as an alarm.

Lights: You can add lights to a scene to bring certain objects into focus, or cast shadows on objects to indicate their physical location.

Three-dimensional models: A three-dimensional model is a visual representation of a .gltf or .glb file imported as a resource.

> **Note**
>
> AWS IoT TwinMaker is not intended for use in, or in association with, the operation of any hazardous environments or critical systems that may lead to serious bodily injury or death or cause environmental or property damage.
> Data collected through your use of AWS IoT TwinMaker should be evaluated for accuracy as appropriate for your use case. AWS IoT TwinMaker should not be used as a substitute for

human monitoring of physical systems for purposes of assessing whether such systems are operating safely.

Getting started with AWS IoT TwinMaker

The topics in this section describe how to do the following.

- Create and set up a new workspace.
- Create an entity and add a component to it.

Prerequisites:

To create your first workspace and scene, you need the following AWS resources.

- An AWS account.
- An IAM service role for AWS IoT TwinMaker. This role is automatically generated by default, when you create a new AWS IoT TwinMaker workspace in the AWS IoT TwinMaker console.

  If you don't choose to let AWS IoT TwinMaker automatically create a new IAM service role, you must specify one that you have already created.

  For instructions on creating and managing this service role, see ???.

  For more information about IAM service roles, see Creating a role to delegate permissions to an AWS service.

  > **Important**
  >
  > This service role must have an attached policy that grants permission for the service to read and write to an Amazon S3 bucket. AWS IoT TwinMaker uses this role to access other services on your behalf. You will also need to assign a trust relationship between this role and AWS IoT TwinMaker so that the service can assume the role. If your twin interacts with other AWS services, add the necessary permissions for those services as well.

Topics

Create and manage a service role for AWS IoT TwinMaker

AWS IoT TwinMaker requires that you use a service role to allow it to access resources in other services on your behalf. This role must have a trust relationship with AWS IoT TwinMaker. When you create a workspace, you must assign this role to the workspace. This topic contains example policies that show you how to configure permissions for common scenarios.

Assign trust

The following policy establishes a trust relationship between your role and AWS IoT TwinMaker. Assign this trust relationship to the role that you use for your workspace.

JSON

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

Amazon S3 permissions

The following policy allows your role to read and delete from and write to an Amazon S3 bucket. Workspaces store resources in Amazon S3, so the Amazon S3 permissions are required for all workspaces.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucket*",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::*/DO_NOT_DELETE_WORKSPACE_*"
      ]
    }
  ]
}
```

Note

When you create a workspace, AWS IoT TwinMaker creates a file in your Amazon S3 bucket that indicates it's being used by a workspace. This policy gives AWS IoT TwinMaker permission to delete that file when you delete the workspace.
AWS IoT TwinMaker places other objects related to your workspace. It's your responsibility to delete these objects when you delete a workspace.

Assign permissions to a specific Amazon S3 bucket

When you create a workspace in the AWS IoT TwinMaker console, you can choose to have AWS IoT TwinMaker create an Amazon S3 bucket for you. You can find information about this bucket by using the following AWS CLI command.

```
aws iottwinmaker get-workspace --workspace-id workspace name
```

The following example shows the format of the output of this command.

```
{
    "arn": "arn:aws:iottwinmaker:region:account Id:workspace/workspace name",
    "creationDateTime": "2021-11-30T11:30:00.000000-08:00",
    "description": "",
    "role": "arn:aws:iam::account Id:role/service role name",
    "s3Location": "arn:aws:s3:::bucket name",
    "updateDateTime": "2021-11-30T11:30:00.000000-08:00",
    "workspaceId": "workspace name"
}
```

To update your policy so that it assigns permissions for a specific Amazon S3 bucket, use the value of *bucket name*.

The following policy allows your role to read and delete from and write to a specific Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucket*",
        "s3:GetObject",
```

```
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket name",
        "arn:aws:s3:::bucket name/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::iottwinmakerbucket/DO_NOT_DELETE_WORKSPACE_*"
      ]
    }
  ]
}
```

Permissions for built-in connectors

If your workspace interacts with other AWS services by using built-in connectors, you must include permissions for those services in this policy. If you use the **com.amazon.iotsitewise.connector** component type, you must include permissions for AWS IoT SiteWise. For more information about component types, see ???.

> **Note**
>
> If you interact with other AWS services by using a custom component type, you must grant the role permission to run the Lambda function that implements the function in your component type. For more information, see ???.

The following example shows how to include AWS IoT SiteWise in your policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucket*",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket name",
        "arn:aws:s3:::bucket name/*"
      ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iotsitewise:DescribeAsset"
        ],
        "Resource": "arn:aws:s3:::bucket name"
        },
    {
        "Effect": "Allow",
        "Action": [
            "iotsitewise:DescribeAssetModel"
        ],
        "Resource": "arn:aws:s3:::bucket name"
        },
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::*/DO_NOT_DELETE_WORKSPACE_*"
      ]
    }
  ]
}
```

If you use the **com.amazon.iotsitewise.connector** component type and need to read property data from AWS IoT SiteWise, you must include the following permission in your policy.

```
...
{
    "Action": [
        "iotsitewise:GetPropertyValueHistory",
    ],
    "Resource": [
        "AWS IoT SiteWise asset resource ARN"
    ],
    "Effect": "Allow"
},
...
```

If you use the **com.amazon.iotsitewise.connector** component type and need to write property data to AWS IoT SiteWise, you must include the following permission in your policy.

```
...
{
    "Action": [
        "iotsitewise:BatchPutPropertyValues",
    ],
    "Resource": [
        "AWS IoT SiteWise asset resource ARN"
    ],
    "Effect": "Allow"
},
...
```

If you use the **com.amazon.iotsitewise.connector.edgevideo** component type, you must include permissions for AWS IoT SiteWise and Kinesis Video Streams. The following example policy shows how to include AWS IoT SiteWise and Kinesis Video Streams permissions in your policy.

```
...
{
    "Action": [
```

```
        "iotsitewise:DescribeAsset",
        "iotsitewise:GetAssetPropertyValue"
    ],
    "Resource": [
        "AWS IoT SiteWise asset resource ARN for the Edge Connector for Kinesis Video
 Streams"
    ],
    "Effect": "Allow"
},
{
    "Action": [
        "iotsitewise:DescribeAssetModel"
    ],
    "Resource": [
        "AWS IoT SiteWise model resource ARN for the Edge Connector for Kinesis Video
 Streams"
    ],
    "Effect": "Allow"
},
{
    "Action": [
        "kinesisvideo:DescribeStream"
    ],
    "Resource": [
        "Kinesis Video Streams stream ARN"
    ],
    "Effect": "Allow"
},
...
```

Permissions for a connector to an external data source

If you create a component type that uses a function that connects to an external data source, you must give your service role permission to use the Lambda function that implements the function. For more information about creating component types and functions, see ???.

The following example gives permission to your service role to use a Lambda function.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucket*",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:PutObject"
      ],
      "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    },
    {
        "Action": [
            "lambda:invokeFunction"
        ],
        "Resource": [
        "arn:aws:lambda:us-east-1:111122223333:function:example-function"
        ],
        "Effect": "Allow"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::*/DO_NOT_DELETE_WORKSPACE_*"
      ]
    }
  ]
}
```

For more information about creating roles and assigning policies and trust relationships to them by using the IAM console, the AWS CLI, and the IAM API, see Creating a role to delegate permissions to an AWS service.

Modify your workspace IAM role to use the Athena data connector

To use the AWS IoT TwinMaker Athena tabular data connector, you must update your AWS IoT TwinMaker workspace IAM role. Add the following permissions to your workspace IAM role:

> **Note**
>
> This IAM change only works for Athena tabular data stored with AWS Glue and Amazon S3. To use Athena with other data sources, you must configure an IAM role for Athena, see Identity and access management in Athena.

```
{
    "Effect": "Allow",
    "Action": [
        "athena:GetQueryExecution",
        "athena:GetQueryResults",
        "athena:GetTableMetadata",
        "athena:GetWorkGroup",
        "athena:StartQueryExecution",
        "athena:StopQueryExecution"
    ],
    "Resource": [
        "athena resouces arn"
    ]
},// Athena permission
{
    "Effect": "Allow",
    "Action": [
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetDatabase",
        "glue:GetDatabases"
    ],
    "Resource": [
        "glue resouces arn"
    ]
},// This is an example for accessing aws glue
{
    "Effect": "Allow",
    "Action": [
        "s3:ListBucket",
```

```
        "s3:GetObject"
    ],
    "Resource": [
        "Amazon S3 data source bucket resources arn"
    ]
}, // S3 bucket for storing the tabular data.
{
    "Effect": "Allow",
    "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload",
        "s3:CreateBucket",
        "s3:PutObject",
        "s3:PutBucketPublicAccessBlock"
    ],
    "Resource": [
        "S3 query result bucket resources arn"
    ]
} // Storing the query results
```

Read the Identity and access management in Athena for more information on Athena IAM configuration.

Create a workspace

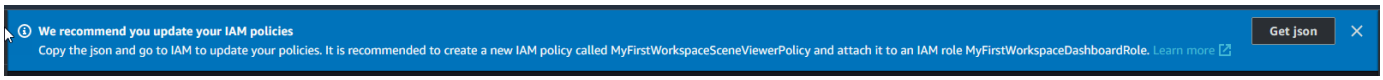
To create and configure your first workspace, use the following steps.

> **Note**
>
> This topic shows you how to create a simple workspace with a single resource. For a fully featured workspace with multiple resources, try the sample setup in the AWS IoT TwinMaker samples Github repository.

1. On the AWS IoT TwinMaker console home page, choose **Workspaces** in the left navigation pane.
2. On the **Workspaces** page, choose **Create workspace**.

3. On the **Create a Workspace** page, enter a name for your workspace.
4. (Optional) Add a description for your workspace.
5. Under **S3 resource**, choose **Create an S3 bucket**. This option creates an Amazon S3 bucket where AWS IoT TwinMaker stores information and resources related to the workspace. Each workspace requires its own bucket.
6. Under **Execution role**, choose either **Auto-generate a new role** or the custom IAM role that you created as for this workspace.

   If you choose **Auto-generate a new role**, AWS IoT TwinMaker attaches a policy to the role that grants permission to the new service role to access other AWS services, including permission to read and write to the Amazon S3 bucket that you specify in the previous step. For information about assigning permissions to this role, see ???.
7. Choose **Create Workspace**. The following banner appears at the top of the **Workspaces** page.



8. Choose **Get json**. We recommend you add the IAM policy you see to the IAM role that AWS IoT TwinMaker created for users and accounts that view the Grafana dashboard. The name of this role follows this pattern: *workspace-name*DashboardRole, For instructions on how to create a policy and attach it to a role, see Modifying a role permissions policy (console).

   The following example contains the policy to add to the dashboard role.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::iottwinmaker-workspace-workspace-name-lower-
case-123456789012",
                "arn:aws:s3:::iottwinmaker-workspace-workspace-name-lower-
case-123456789012/*"
            ]
```

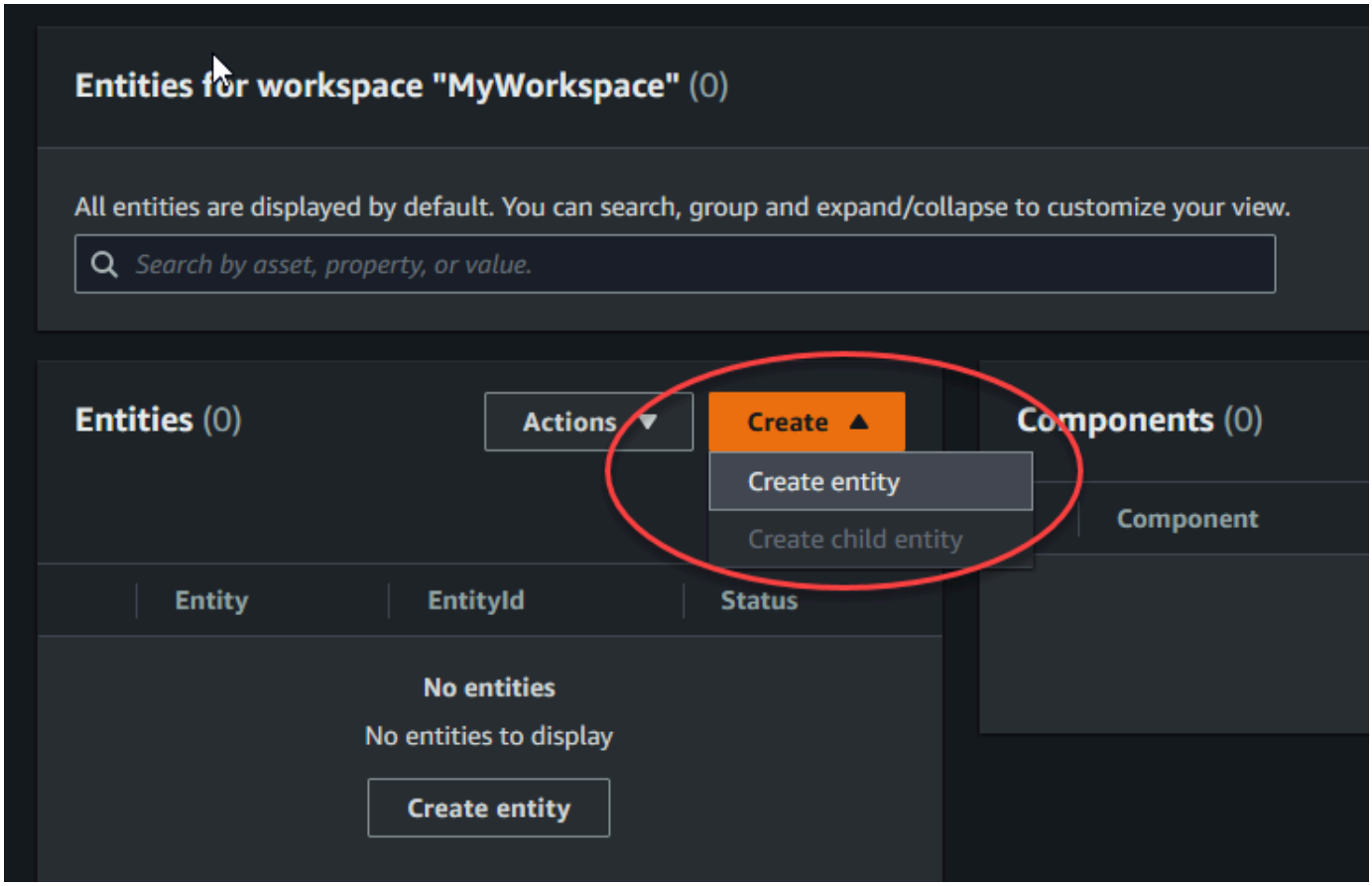
```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
                "iottwinmaker:List*"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspace-name",
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspace-name/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        }
    ]
}
```

You're now ready to start creating a data model for your workspace with your first entity. For instructions on how to do this, see Create your first entity.

Create your first entity

To create your first entity, use the following steps.

1. On the **Workspaces** page, choose your workspace, and then in the left pane choose **Entities**.
2. On the **Entities** page, choose **Create**, and then choose **Create entity**.



3. In the **Create an entity** window, enter a name for your entity. This example uses a **`CookieMixer`** entity.
4. (Optional) Enter a description for your entity.
5. Choose **Create entity**,

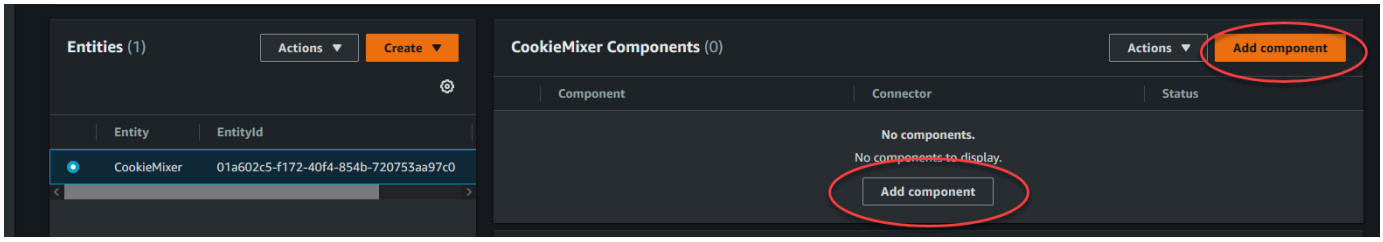
Entities contain data about each item in your workspace. You put data into entities by adding components. AWS IoT TwinMaker provides the following built-in component types.

- **Parameters**: Adds a set of key-value properties.
- **Document**: Adds a name and a URL for a document that contains information about the entity.
- **Alarms**: Connects to an alarm time-series data source.
- **SiteWise connector**: Pulls time-series properties that are defined in an AWS IoT SiteWise asset.
- **Edge Connector for Kinesis Video Streams AWS IoT Greengrass**: Pulls video data from the Edge Connector for KVS AWS IoT Greengrass. For more information, see AWS IoT TwinMaker video integration.

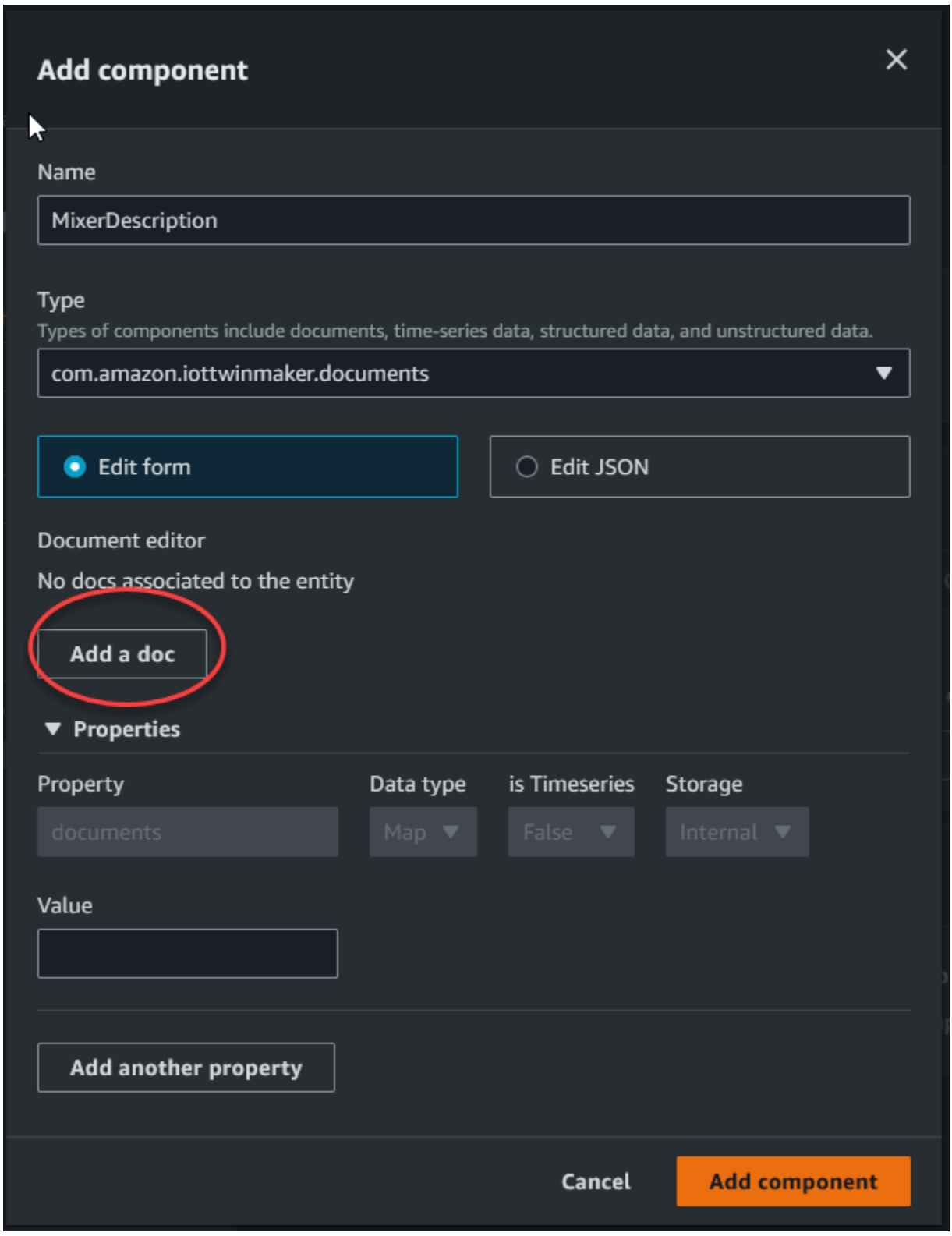
You can see these component types and their definitions by choosing **Component types** in the left pane. You can also create a new component type on the **Component types** page. For more information about creating component types, see Using and creating component types.

In this example, we create a simple document component that adds descriptive information about your entity.

1. On the **Entities** page, choose the entity, and then choose add component.



2. In the **Add component** window, enter a name for your component. Since this example uses a cookie mixer entity, we enter **`MixerDescription`** in the **Name** field.



3. Choose **Add a doc**, then enter values for the doc **Name** and **External Url**. With the documents component, you can store a list of external URLs that contain important information about the entity.

4. Choose **Add component**.

You're now ready to create your first scene. For instructions on how to do this, see Creating and editing AWS IoT TwinMaker scenes.

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

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

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

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

Sign up for an AWS account

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

To sign up for an AWS account

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

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

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

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

Create a user with administrative access

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

Secure your AWS account root user

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

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

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

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

Create a user with administrative access

1. Enable IAM Identity Center.

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

2. In IAM Identity Center, grant administrative access to a user.

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

Sign in as the user with administrative access

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

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

Assign access to additional users

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

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

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

Using and creating component types

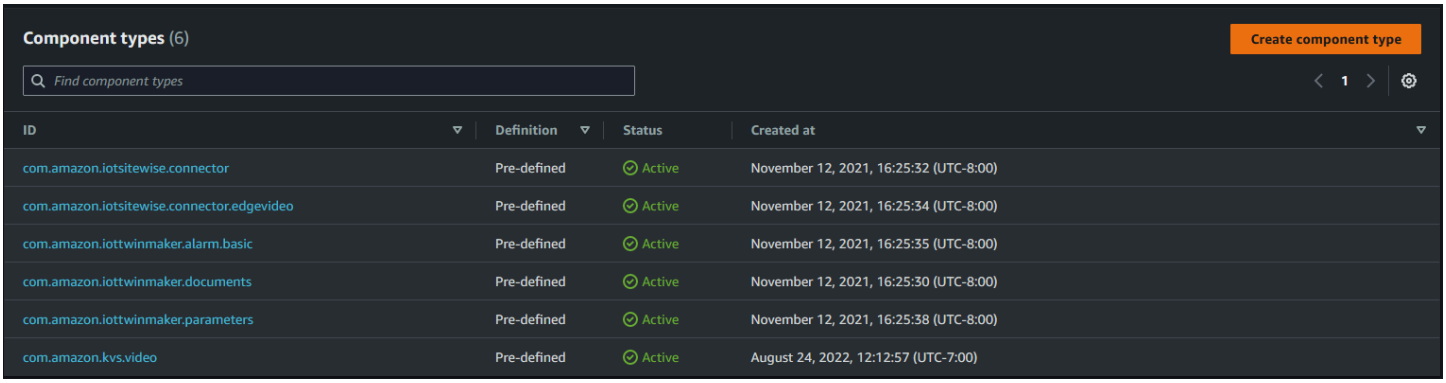
This topic walks you through the values and structures that you use to create an AWS IoT TwinMaker component type. It shows you how to create a request object that you can pass to the CreateComponentType API or by using the component type editor in the AWS IoT TwinMaker console.

Components provide context for properties and data for their associated entities.

Built-in component types

In the AWS IoT TwinMaker console, when you choose a workspace and then choose **Component types** in the left pane, you see the following component types.

- **com.amazon.iotsitewise.resourcesync**: A component type that automatically syncs your AWS IoT SiteWise assets and asset models and converts them into AWS IoT TwinMaker entities components and component types. For more information on using AWS IoT SiteWise asset sync, see Asset sync with AWS IoT SiteWise.
- **com.amazon.iottwinmaker.alarm.basic**: A basic alarm component that pulls alarm data from an external source to an entity. This component doesn't contain a function that connects to a specific data source. This means that the alarm component is abstract and can be inherited by another component type that specifies a data source and a function that reads from that source.
- **com.amazon.iottwinmaker.documents**: A simple mapping of titles to URLs for documents that contain information about an entity.
- **com.amazon.iotsitewise.connector.edgevideo**: A component that pulls video from an IoT device using the Edge Connector for Kinesis Video Streams AWS IoT Greengrass component into an entity. The Edge Connector for Kinesis Video Streams AWS IoT Greengrass component is not an AWS IoT TwinMaker component, but rather a prebuilt AWS IoT Greengrass component that is deployed locally on your IoT device.
- **com.amazon.iotsitewise.connector**: A component that pulls AWS IoT SiteWise data into an entity.
- **com.amazon.iottwinmaker.parameters**: A component that adds static key-value pairs to an entity.
- **com.amazon.kvs.video**: A component that pulls video from Kinesis Video Streams into an AWS IoT TwinMaker entity.



Core features of AWS IoT TwinMaker component types

The following list describes the core features of component types.

- **Property definitions**: The PropertyDefinitionRequest object defines a property that you can populate in the scene composer or it can be populated with data pulled from external data sources. Static properties that you set are stored in AWS IoT TwinMaker. Time-series properties and other properties that are pulled from data sources are stored externally.

  You specify property definitions inside a string to the `PropertyDefinitionRequest` map. Each string must be unique to the map.

- **Functions**: The FunctionRequest object specifies a Lambda function that reads from and potentially writes to an external data source.

  A component type that contains a property with a value that is stored externally but that doesn't have a corresponding function to retrieve the values is an abstract component type. You can extend concrete component types from an abstract component type. You can't add abstract component types to an entity. They don't appear in the scene composer.

  You specify functions inside a string to `FunctionRequest` map. The string must specify one of the following predefined function types.

  - `dataReader`: A function that pulls data from an external source.
  - `dataReaderByEntity`: A function that pulls data from an external source.

    When you use this type of data reader, the GetPropertyValueHistory API operation supports only entity-specific queries for properties in this component type. (You can only request the property value history for `componentName` + `entityId`.)
  - `dataReaderByComponentType`: A function that pulls data from an external source.

When you use this type of data reader, the GetPropertyValueHistory API operation supports only cross-entity queries for properties in this component type. (You can only request the property value history for `componentTypeId`.)

- `dataWriter`: A function that writes data to an external source.
- `schemaInitializer`: A function that automatically initializes property values whenever you create an entity that contains the component type.

One of the three types of data reader functions is required in a non-abstract component type.

For an example of a Lambda function that implements time-stream telemetry components, including alarms, see the data reader in AWS IoT TwinMaker Samples.

> **Note**
>
> Because the alarm connector inherits from the abstract alarm component type, the Lambda function must return the `alarm_key` value. If you don't return this value, Grafana won't recognize it as an alarm. This is required for all components that return alarms.

- **Inheritance**: Component types promote code reusability through inheritance. A component type can inherit up to 10 parent component types.

  Use the `extendsFrom` parameter to specify the component types from which your component type inherits properties and functions.
- **isSingleton**: Some components contain properties, such as location coordinates, that can't be included more than once in an entity. Set the value of the `isSingleton` parameter to `true` to indicate that your component type can be included only once in an entity.

Creating property definitions

The following table describes the parameters of a `PropertyDefinitionRequest`.

Parameter	Description
`isExternalId`	A Boolean that specifies whether the property is a unique identifier (such as an AWS IoT

Parameter	Description
	SiteWise asset Id) of a property value that is stored externally. The default value of this property is `false`.
`isStoredExternally`	A Boolean that specifies whether the property value is stored externally. The default value of this property is `false`.
`isTimeSeries`	A Boolean that specifies whether the property stores time-series data. The default value of this property is `false`
`isRequiredInEntity`	A Boolean that specifies whether the property must have a value in an entity that uses the component type.
`dataType`	A DataType object that specifies the data type (such as string, map, list, and unit of measure) of the property.
`defaultValue`	A DataValue object that specifies the default value of the property.
`configuration`	A string-to-string map that specifies additiona l information that you need to connect to an external data source.

Creating functions

The following table describes the parameters of a `FunctionRequest`.

Parameter	Description
`implementedBy`	A DataConnector object that specifies the Lambda function that connects to the external data source.
`requiredProperties`	A list of properties that the function needs in order to read from and write to an external data source.
`scope`	The scope of the function. Use `Workspace` for functions with a scope that spans an entire workspace. Use `Entity` for functions with a scope that is limited to the entity that contains the component.

For examples that show how to create and extend component types, see ???.

Example component types

This topic contains examples that show how to implement key concepts of component types.

Alarm (abstract)

The following example is the abstract alarm component type that appears in the AWS IoT TwinMaker console. It contains a `functions` list that consists of a `dataReader` that has no `implementedBy` value.

```
{
  "componentTypeId": "com.example.alarm.basic:1",
  "workspaceId": "MyWorkspace",
  "description": "Abstract alarm component type",
  "functions": {
    "dataReader": {
         "isInherited": false
    }
  },
```

```
  "isSingleton": false,
  "propertyDefinitions": {
    "alarm_key": {
      "dataType": { "type": "STRING" },
      "isExternalId": true,
      "isRequiredInEntity": true,
      "isStoredExternally": false,
      "isTimeSeries": false
    },
    "alarm_status": {
      "dataType": {
        "allowedValues": [
          {
            "stringValue": "ACTIVE"
          },
          {
            "stringValue": "SNOOZE_DISABLED"
          },
          {
            "stringValue": "ACKNOWLEDGED"
          },
          {
            "stringValue": "NORMAL"
          }
        ],
        "type": "STRING"
      },
      "isRequiredInEntity": false,
      "isStoredExternally": true,
      "isTimeSeries": true
    }
  }
}
```

Notes:

Values for `componentTypeId` and `workspaceID` are required. The value of `componentTypeId` must be unique to your workspace. The value of `alarm_key` is a unique identifier that a function can use to retrieve alarm data from an external source. The value of the key is required and stored in AWS IoT TwinMaker. The `alarm_status` time series values are stored in the external source.

More examples are available in AWS IoT TwinMaker Samples.

Timestream telemetry

The following example is a simple component type that retrieves telemetry data about a specific type of component (such as an alarm or a cookie mixer) from an external source. It specifies a Lambda function that component types inherit.

```
{
    "componentTypeId": "com.example.timestream-telemetry",
    "workspaceId": "MyWorkspace",
    "functions": {
        "dataReader": {
            "implementedBy": {
                "lambda": {
                    "arn": "lambdaArn"
                }
            }
        }
    },
    "propertyDefinitions": {
        "telemetryType": {
            "dataType": { "type": "STRING" },
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "telemetryId": {
            "dataType": { "type": "STRING" },
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        }
    }
}
```

Alarm (inherits from abstract alarm)

The following example inherits from both the abstract alarm and the timestream telemetry component types. It specifies its own Lambda function that retrieves alarm data.

```
{
    "componentTypeId": "com.example.cookiefactory.alarm",
    "workspaceId": "MyWorkspace",
    "extendsFrom": [
        "com.example.timestream-telemetry",
        "com.amazon.iottwinmaker.alarm.basic"
    ],
    "propertyDefinitions": {
        "telemetryType": {
            "defaultValue": {
                "stringValue": "Alarm"
            }
        }
    },
    "functions": {
        "dataReader": {
            "implementedBy": {
                "lambda": {
                    "arn": "lambdaArn"
                }
            }
        }
    }
}
```

Note

Because the alarm connector inherits from the abstract alarm component type, the Lambda function must return the `alarm_key` value. If you don't return this value, Grafana won't recognize it as an alarm. This is required for all components that return alarms.

Equipment examples

The examples in this section show how to model potential pieces of equipment. You can use these examples to get some ideas about how to model equipment in your own processes.

Cookie mixer

The following example inherits from the timestream telemetry component type. It specifies additional time-series properties for a cookie mixer's rotation rate and temperature.

```
{
    "componentTypeId": "com.example.cookiefactory.mixer",
    "workspaceId": "MyWorkspace",
    "extendsFrom": [
        "com.example.timestream-telemetry"
    ],
    "propertyDefinitions": {
        "telemetryType": {
            "defaultValue" : { "stringValue": "Mixer" }
        },
        "RPM": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        },
        "Temperature": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        }
    }
}
```

Water tank

The following example inherits from the timestream telemetry component type. It specifies additional time-series properties for a water tank's volume and flow rate.

```
{
    "componentTypeId": "com.example.cookiefactory.watertank",
    "workspaceId": "MyWorkspace",
    "extendsFrom": [
        "com.example.timestream-telemetry"
    ],
```

```
    "propertyDefinitions": {
        "telemetryType": {
            "defaultValue" : { "stringValue": "WaterTank" }
        },
        "tankVolume1": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        },
        "tankVolume2": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        },
        "flowRate1": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        },
        "flowrate2": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isStoredExternally": true
        }
    }
}
```

Space location

The following example contains properties, the values of which are stored in AWS IoT TwinMaker. Because the values are specified by users and stored internally, no function is required to retrieve them. The example also uses the `RELATIONSHIP` data type to specify a relationship with another component type.

This component provides a lightweight mechanism for adding context to a digital twin. You can use it to add metadata indicating where something is located. You can also use this information in logic used for determining which cameras can see a piece of equipment or space, or for knowing how to dispatch someone to a location.

```
{
```

```
    "componentTypeId": "com.example.cookiefactory.space",
    "workspaceId": "MyWorkspace",
    "propertyDefinitions": {
        "position":  {"dataType": {"nestedType": {"type": "DOUBLE"},"type": "LIST"}},
        "rotation":  {"dataType": {"nestedType": {"type": "DOUBLE"},"type": "LIST"}},
        "bounds":  {"dataType": {"nestedType": {"type": "DOUBLE"},"type": "LIST"}},
        "parent_space" : { "dataType": {"type": "RELATIONSHIP"}}
    }
}
```

AWS IoT TwinMaker bulk operations

Use a metadataTransferJob to transfer and manage your AWS IoT TwinMaker resources at scale. A metadataTransferJob allows you to perform bulk operations and transfer resources between AWS IoT TwinMaker and AWS IoT SiteWise and Amazon S3.

You can use bulk operations in the following scenarios:

- Mass migration of assets and data between accounts, for example migrating from a development account to a production account.
- Large scale asset management, such as uploading, and editing AWS IoT assets at scale.
- Mass import of your assets into AWS IoT TwinMaker and AWS IoT SiteWise.
- Bulk import of AWS IoT TwinMaker entities from existing ontology files such as `revit` or `BIM` files.

Topics

Key concepts and terminology

AWS IoT TwinMaker bulk operations use the following concepts and terminology:

- **Import**: The action of moving resources into an AWS IoT TwinMaker workspace. For example, from a local file, a file in an Amazon S3 bucket, or from AWS IoT SiteWise to an AWS IoT TwinMaker workspace.
- **Export**: The action of moving resources from an AWS IoT TwinMaker workspace to a local machine or an Amazon S3 bucket.
- **Source**: The starting location from where you want to move resources.

  For example, an Amazon S3 bucket is an import source, and an AWS IoT TwinMaker workspace is an export source.
- **Destination**: The desired location where you want to move your resources to.

For example, an Amazon S3 bucket is an export destination, and an AWS IoT TwinMaker workspace is an import destination.

- **AWS IoT SiteWise Schema**: A schema used to import and export resources to and from AWS IoT SiteWise.
- **AWS IoT TwinMaker Schema**: A schema used to import and export resources to and from AWS IoT TwinMaker.
- **AWS IoT TwinMaker top-level resources**: Resources used in existing APIs. Specifically, an **Entity** or a **ComponentType**.
- **AWS IoT TwinMaker sub-level resources**: Nested resource types used in metadata definitions. Specifically, a **Component**.
- **Metadata**: Key information required to successfully import or export AWS IoT SiteWise and AWS IoT TwinMaker resources.
- **metadataTransferJob**: The object created when you run `CreateMetadataTransferJob`.

AWS IoT TwinMaker metadataTransferJob functionality

This topic explains the behavior AWS IoT TwinMaker follows when you run a bulk operation– how a metadataTransferJob is processed. It also explains how to define a schema with the metadata required to transfer your resources. AWS IoT TwinMaker bulk operations support the following functionality:

- **Top-level resource create or replace:** AWS IoT TwinMaker will create new resources or replace all existing resources that are uniquely identified by a resource ID.

  For example, if an entity exists in the system, the entity definition will be replaced by the new one defined in the template under the `Entity` key.

- **Sub-resource create or replace:**

  From the EntityComponent level, you can only create or replace a component. The entity must already exist, otherwise, the action will produce a ValidationException.

  From the property or relationship level, you can only create or replace a property or relationship, and the containing EntityComponent must already exist.

- **Sub-resource delete:**

AWS IoT TwinMaker also supports sub-resource deletion. A sub-resource can be a component, property, or relationship.

If you want to delete a component, you must do it from the entity level.

If you want to delete a property or relationship, you must do it from the Entity or EntityComponent level.

To delete a sub-resource, you update the higher level resource and omit the definition of the sub-resource.

- **No top-level resource deletion:** AWS IoT TwinMaker will never delete top-level resources. A top-level resource refers to an entity or ComponentType.
- **No sub-resource Definitions for the same top-level resource in one template:**

  You can't provide the full entity definition and sub-resource (like property) definition of the same entity in the same template.

  If an entityId is used in Entity, you cannot use the same ID in Entity, EntityComponent, property, or relationship.

  If an entityId or componentName combination is used in EntityComponent, you cannot use the same combination in EntityComponent, property, or relationship.

  If an entityId, componentName, propertyName combination is used in property or relationship, you cannot use the same combination in the property or relationship.
- **ExternalId is optional for AWS IoT TwinMaker:** The ExternalId can be used to help you identify your resources.

Performing bulk import and export operations

This topic covers how to perform bulk import and export operations and how to handle errors in your transfer jobs. It provides examples of transfer jobs using CLI commands.

The AWS IoT TwinMaker API Reference contains information on the CreateMetadataTransferJob and other API actions.

Topics

- metadataTransferJob prerequisites

- IAM permissions
- Run a bulk operation
- Error handling
- Import metadata templates
- AWS IoT TwinMaker metadataTransferJob examples

metadataTransferJob prerequisites

Please complete the following prerequisites before you run a metadataTransferJob:

- Create an AWS IoT TwinMaker workspace. The workspace can be the import destination or export source for a metadataTransferJob. For information on creating a workspace see, Create a workspace.
- Create an Amazon S3 bucket to store resources. For more information on using Amazon S3 see, What is Amazon S3?

IAM permissions

When you perform bulk operations you need to create an IAM policy with permissions to allow for the exchange of AWS resources between Amazon S3, AWS IoT TwinMaker, AWS IoT SiteWise, and your local machine. For more information on creating IAM policies, see Creating IAM policies.

The policy statements for AWS IoT TwinMaker, AWS IoT SiteWise and Amazon S3 are listed here:

- **AWS IoT TwinMaker policy**:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetBucketLocation",
            "s3:ListBucket",
            "s3:AbortMultipartUpload",
```

```
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts"
        ],
        "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:GetWorkspace",
                "iottwinmaker:CreateEntity",
                "iottwinmaker:GetEntity",
                "iottwinmaker:UpdateEntity",
                "iottwinmaker:GetComponentType",
                "iottwinmaker:CreateComponentType",
                "iottwinmaker:UpdateComponentType",
                "iottwinmaker:ListEntities",
                "iottwinmaker:ListComponentTypes",
                "iottwinmaker:ListTagsForResource",
                "iottwinmaker:TagResource",
                "iottwinmaker:UntagResource"
            ],
            "Resource": "*"
        }
    ]
}
```

- **AWS IoT SiteWise policy**:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetBucketLocation",
            "s3:ListBucket",
            "s3:AbortMultipartUpload",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts"
        ],
        "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:CreateAsset",
                "iotsitewise:CreateAssetModel",
                "iotsitewise:UpdateAsset",
                "iotsitewise:UpdateAssetModel",
                "iotsitewise:UpdateAssetProperty",
                "iotsitewise:ListAssets",
                "iotsitewise:ListAssetModels",
                "iotsitewise:ListAssetProperties",
                "iotsitewise:ListAssetModelProperties",
                "iotsitewise:ListAssociatedAssets",
                "iotsitewise:DescribeAsset",
                "iotsitewise:DescribeAssetModel",
                "iotsitewise:DescribeAssetProperty",
                "iotsitewise:AssociateAssets",
                "iotsitewise:DisassociateAssets",
                "iotsitewise:AssociateTimeSeriesToAssetProperty",
                "iotsitewise:DisassociateTimeSeriesFromAssetProperty",
                "iotsitewise:BatchPutAssetPropertyValue",
                "iotsitewise:BatchGetAssetPropertyValue",
                "iotsitewise:TagResource",
                "iotsitewise:UntagResource",
                "iotsitewise:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

- **Amazon S3 policy**:

```
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:AbortMultipartUpload",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts"
```

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

Alternatively you can scope your Amazon S3 policy to only access a single Amazon S3 bucket, see the following policy.

Amazon S3 single bucket scoped policy

```
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:AbortMultipartUpload",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts"
    ],
    "Resource": [
        "arn:aws:s3:::bucket name",
        "arn:aws:s3:::bucket name/*"
    ]
}
```

Set access control for a metadataTransferJob

To control what kind of jobs a user can access, add the following IAM policy to the role used to call AWS IoT TwinMaker.

> **Note**
>
> This policy only allows access to AWS IoT TwinMaker import and export jobs that transfer resources to and from Amazon S3.

```
{
    "Effect": "Allow",
    "Action": [
```

```
        "iottwinmaker:*DataTransferJob*"
    ],
    "Resource": "*",
    "Condition": {
        "StringLikeIfExists": {
            "iottwinmaker:sourceType": [
                "s3",
                "iottwinmaker"
            ],
            "iottwinmaker:destinationType": [
                "iottwinmaker",
                "s3"
            ]
        }
    }
}
```

Run a bulk operation

This section covers how to perform bulk import and export operations.

Import data from Amazon S3 to AWS IoT TwinMaker

1. Specify the resources you want to transfer using the AWS IoT TwinMaker metadataTransferJob schema. Create and store your schema file in your Amazon S3 bucket.

   For example schemas, see Import metadata templates.

2. Create a request body and save it as a JSON file. The request body specifies the source and destination for the transfer job. Make sure to specify your Amazon S3 bucket as the source and your AWS IoT TwinMaker workspace as the destination.

   The following is an example of a request body:

   ```
   {
       "metadataTransferJobId": "your-transfer-job-Id",
       "sources": [{
           "type": "s3",
           "s3Configuration": {
               "location": "arn:aws:s3:::amzn-s3-demo-bucket/your_import_data.json"
           }
       }],
       "destination": {
   ```

```
        "type": "iottwinmaker",
        "iotTwinMakerConfiguration": {
            "workspace": "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/your-worksapce-name"
        }
    }
}
```

Record the file name you gave your request body, you will need it in the next step. In this example the request body is named `createMetadataTransferJobImport.json`.

3. Run the following CLI command to invoke `CreateMetadataTransferJob` (replace the input-json file name with the name you gave your request body):

```
aws iottwinmaker create-metadata-transfer-job --region us-east-1 \
--cli-input-json file://createMetadataTransferJobImport.json
```

This creates a metadataTransferJob and begins the process of the transferring your selected resources.

Export data from AWS IoT TwinMaker to Amazon S3

1. Create a JSON request body with the appropriate filters to choose the resources you want to export. For this example we use:

```
{
    "metadataTransferJobId": "your-transfer-job-Id",
    "sources": [{
        "type": "iottwinmaker",
        "iotTwinMakerConfiguration": {
            "workspace": "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/your-workspace-name",
            "filters": [{
                "filterByEntity": {
                    "entityId": "parent"
                }},
                {
                "filterByEntity": {
                    "entityId": "child"
                }},
                {
```

```
                "filterByComponentType": {
                    "componentTypeId": "component.type.minimal"
                }}
            ]
        }
    }],
    "destination": {
        "type": "s3",
        "s3Configuration": {
            "location": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    }
}
```

The `filters` array lets you specify which resources will be exported. In this example we filter by `entity`, and `componentType`.

Make sure to specify your AWS IoT TwinMaker workspace as the source and your Amazon S3 bucket as the destination of the metadata transfer job.

Save your request body and record the file name, you will need it in the next step. For this example, we named our request body `createMetadataTransferJobExport.json`.

2. Run the following CLI command to invoke `CreateMetadataTransferJob` (replace the input-json file name with the name you gave your request body):

```
aws iottwinmaker create-metadata-transfer-job --region us-east-1 \
--cli-input-json file://createMetadataTransferJobExport.json
```

This creates a metadataTransferJob and begins the process of the transferring your selected resources.

To check or update the status of a transfer job, use the following commands:

- To cancel a job use the CancelMetadataTransferJob API action. When you call CancelMetadataTransferJob, the API only cancels a running metadataTransferJob, and any resources already exported or imported are not affected by this API call.
- To retrieve information on a specific job use the GetMetadataTransferJob API action.

Or, you can call GetMetadataTransferJob on an existing transfer job with the following CLI command:

```
aws iottwinmaker get-metadata-transfer-job --job-id ExistingJobId
```

If you call GetMetadataTransferJob on a non-existing AWS IoT TwinMaker import or export job, you get a `ResourceNotFoundException` error in response.

- To list current jobs, use the ListMetadataTransferJobs API action.

  Here is a CLI example that calls ListMetadataTransferJobs with AWS IoT TwinMaker as the destinationType and `s3` as the sourceType:

  ```
  aws iottwinmaker list-metadata-transfer-jobs --destination-type iottwinmaker --
  source-type s3
  ```

  > **Note**
  >
  > You can change the values for the sourceType and destinationType parameters to match your import or export job's source and destination.

For more examples of CLI commands that invoke these API actions, see AWS IoT TwinMaker metadataTransferJob examples.

If you encounter any errors during the transfer job, see Error handling.

Error handling

After you create and run a transfer job, you can call GetMetadataTransferJob to diagnose any errors that occurred:

```
aws iottwinmaker get-metadata-transfer-job \
--metadata-transfer-job-id your_metadata_transfer_job_id \
--region us-east-1
```

Once you see the state of the job turn to `COMPLETED`, you can verify the results of the job. GetMetadataTransferJob returns an object called `MetadataTransferJobProgress` which contains the following fields:

- **failedCount:** Indicates the number of resources that failed during the transfer process.
- **skippedCount:** Indicates the number of resources that were skipped during the transfer process.
- **succeededCount:** Indicates the number of resources that succeeded during the transfer process.
- **totalCount:** Indicates the total count of resources involved in the transfer process.

Additionally, a reportUrl element is returned which contains a pre-signed URL. If your transfer job has errors you wish to investigate further, then you can download a full error report using this URL.

Import metadata templates

You can import many components, componentTypes, or entities with a single bulk import operation. The examples in this section show how to do this.

template: Importing entities

Use the following template format for a job that imports entities:

```
{
  "entities": [
    {
      "description": "string",
      "entityId": "string",
      "entityName": "string",
      "parentEntityId": "string",
      "tags": {
        "string": "string"
      },
      "components": {
        "string": {
          "componentTypeId": "string",
          "description": "string",
          "properties": {
            "string": {
              "definition": {
                "configuration": {
                  "string": "string"
                },
                "dataType": "DataType",
                "defaultValue": "DataValue",
                "displayName": "string",
```

```
                "isExternalId": "boolean",
                "isRequiredInEntity": "boolean",
                "isStoredExternally": "boolean",
                "isTimeSeries": "boolean"
              },
              "value": "DataValue"
            }
          },
          "propertyGroups": {
            "string": {
              "groupType": "string",
              "propertyNames": [
                "string"
              ]
            }
          }
        }
      }
    }
  ]
}
```

template: Importing componentTypes

Use the following template format for a job that imports componentTypes:

```
{
  "componentTypes": [
    {
      "componentTypeId": "string",
      "componentTypeName": "string",
      "description": "string",
      "extendsFrom": [
        "string"
      ],
      "functions": {
        "string": {
          "implementedBy": {
            "isNative": "boolean",
            "lambda": {
              "functionName": "Telemetry-tsDataReader",
              "arn": "Telemetry-tsDataReaderARN"
            }
          },
```

```
          "requiredProperties": [
            "string"
          ],
          "scope": "string"
        }
      },
      "isSingleton": "boolean",
      "propertyDefinitions": {
        "string": {
          "configuration": {
            "string": "string"
          },
          "dataType": "DataType",
          "defaultValue": "DataValue",
          "displayName": "string",
          "isExternalId": "boolean",
          "isRequiredInEntity": "boolean",
          "isStoredExternally": "boolean",
          "isTimeSeries": "boolean"
        }
      },
      "propertyGroups": {
        "string": {
          "groupType": "string",
          "propertyNames": [
            "string"
          ]
        }
      },
      "tags": {
        "string": "string"
      }
    }
  ]
}
```

template: Importing components

Use the following template format for a job that imports components:

```
{
  "entityComponents": [
    {
      "entityId": "string",
```

```
      "componentName": "string",
      "componentTypeId": "string",
      "description": "string",
      "properties": {
        "string": {
          "definition": {
            "configuration": {
              "string": "string"
            },
            "dataType": "DataType",
            "defaultValue": "DataValue",
            "displayName": "string",
            "isExternalId": "boolean",
            "isRequiredInEntity": "boolean",
            "isStoredExternally": "boolean",
            "isTimeSeries": "boolean"
          },
          "value": "DataValue"
        }
      },
      "propertyGroups": {
        "string": {
          "groupType": "string",
          "propertyNames": [
            "string"
          ]
        }
      }
    }
  ]
}
```

AWS IoT TwinMaker metadataTransferJob examples

Use the following commands to manage your metadata transfers:

- CreateMetadataTransferJob API action.

  CLI command example:

```
aws iottwinmaker create-metadata-transfer-job --region us-east-1 \
--cli-input-json file://yourTransferFileName.json
```

- To cancel a job use the CancelMetadataTransferJob API action.

  CLI command example:

  ```
  aws iottwinmaker cancel-metadata-transfer-job
  --region us-east-1 \
  --metadata-transfer-job-id job-to-cancel-id
  ```

  When you call CancelMetadataTransferJob, it only cancels a specific metadata transfer job, and any resources already exported or imported are not affected.

- To retrieve information on a specific job use the GetMetadataTransferJob API action.

  CLI command example:

  ```
  aws iottwinmaker get-metadata-transfer-job \
  --metadata-transfer-job-id your_metadata_transfer_job_id \
  --region us-east-1 \
  ```

- To list current jobs use the ListMetadataTransferJobs API action.

  You can filter the results returned by ListMetadataTransferJobs using a JSON file. See the following procedure using the CLI:

  1. Create a CLI input JSON file to specify the filters you want to use:

     ```
     {
         "sourceType": "s3",
         "destinationType": "iottwinmaker",
         "filters": [{
             "workspaceId": "workspaceforbulkimport"
         },
         {
             "state": "COMPLETED"
         }]
     }
     ```

     Save it and record the file name, you will need it when entering the CLI command.

  2. Use the JSON file as an argument to the following CLI command:

     ```
     aws iottwinmaker list-metadata-transfer-job --region us-east-1 \
     ```

```
--cli-input-json file://ListMetadataTransferJobsExample.json
```

AWS IoT TwinMaker metadata transfer job schema

metadataTransferJob import schema: Use this AWS IoT TwinMaker metadata schema to validate your data when you upload it to an Amazon S3 bucket:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "IoTTwinMaker",
  "description": "Metadata transfer job resource schema for IoTTwinMaker",
  "definitions": {
    "ExternalId": {
      "type": "string",
      "minLength": 1,
      "maxLength": 128,
      "pattern": "[a-zA-Z0-9][a-zA-Z_\\-0-9.:]*[a-zA-Z0-9]+"
    },
    "Description": {
      "type": "string",
      "minLength": 0,
      "maxLength": 512
    },
    "DescriptionWithDefault": {
      "type": "string",
      "minLength": 0,
      "maxLength": 512,
      "default": ""
    },
    "ComponentTypeName": {
      "description": "A friendly name for the component type.",
      "type": "string",
      "pattern": ".*[^\\u0000-\\u001F\\u007F]*.*",
      "minLength": 1,
      "maxLength": 256
    },
    "ComponentTypeId": {
      "description": "The ID of the component type.",
      "type": "string",
      "pattern": "[a-zA-Z_.\\-0-9:]+",
      "minLength": 1,
      "maxLength": 256
```

```
    },
    "ComponentName": {
      "description": "The name of the component.",
      "type": "string",
      "pattern": "[a-zA-Z_\\-0-9]+",
      "minLength": 1,
      "maxLength": 256
    },
    "EntityId": {
      "description": "The ID of the entity.",
      "type": "string",
      "minLength": 1,
      "maxLength": 128,
      "pattern": "[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}|^[a-zA-
Z0-9][a-zA-Z_\\-0-9.:]*[a-zA-Z0-9]+"
    },
    "EntityName": {
      "description": "The name of the entity.",
      "type": "string",
      "minLength": 1,
      "maxLength": 256,
      "pattern": "[a-zA-Z_0-9-.][a-zA-Z_0-9-. ]*[a-zA-Z0-9]+"
    },
    "ParentEntityId": {
      "description": "The ID of the parent entity.",
      "type": "string",
      "minLength": 1,
      "maxLength": 128,
      "pattern": "\\$ROOT|^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]
{12}|^[a-zA-Z0-9][a-zA-Z_\\-0-9.:]*[a-zA-Z0-9]+",
      "default": "$ROOT"
    },
    "DisplayName": {
      "description": "A friendly name for the property.",
      "type": "string",
      "pattern": ".*[^\\u0000-\\u001F\\u007F]*.*",
      "minLength": 0,
      "maxLength": 256
    },
    "Tags": {
      "description": "Metadata that you can use to manage the entity / componentType",
      "patternProperties": {
        "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$": {
          "type": "string",
```

```
          "minLength": 1,
          "maxLength": 256
        }
      },
      "existingJavaType": "java.util.Map<String,String>",
      "minProperties": 0,
      "maxProperties": 50
    },
    "Relationship": {
      "description": "The type of the relationship.",
      "type": "object",
      "properties": {
        "relationshipType": {
          "description": "The type of the relationship.",
          "type": "string",
          "pattern": ".*",
          "minLength": 1,
          "maxLength": 256
        },
        "targetComponentTypeId": {
          "description": "The ID of the target component type associated with this
relationship.",
          "$ref": "#/definitions/ComponentTypeId"
        }
      },
      "additionalProperties": false
    },
    "DataValue": {
      "description": "An object that specifies a value for a property.",
      "type": "object",
      "properties": {
        "booleanValue": {
          "description": "A Boolean value.",
          "type": "boolean"
        },
        "doubleValue": {
          "description": "A double value.",
          "type": "number"
        },
        "expression": {
          "description": "An expression that produces the value.",
          "type": "string",
          "pattern": "(^\\$\\{Parameters\\.[a-zA-z]+([a-zA-z_0-9]*)}$)",
          "minLength": 1,
```

```
          "maxLength": 316
        },
        "integerValue": {
          "description": "An integer value.",
          "type": "integer"
        },
        "listValue": {
          "description": "A list of multiple values.",
          "type": "array",
          "minItems": 0,
          "maxItems": 50,
          "uniqueItems": false,
          "insertionOrder": false,
          "items": {
            "$ref": "#/definitions/DataValue"
          },
          "default": null
        },
        "longValue": {
          "description": "A long value.",
          "type": "integer",
          "existingJavaType": "java.lang.Long"
        },
        "stringValue": {
          "description": "A string value.",
          "type": "string",
          "pattern": ".*",
          "minLength": 1,
          "maxLength": 256
        },
        "mapValue": {
          "description": "An object that maps strings to multiple DataValue objects.",
          "type": "object",
          "patternProperties": {
            "[a-zA-Z_\\-0-9]+": {
              "$ref": "#/definitions/DataValue"
            }
          },
          "additionalProperties": {
            "$ref": "#/definitions/DataValue"
          }
        },
        "relationshipValue": {
          "description": "A value that relates a component to another component.",
```

```
          "type": "object",
          "properties": {
            "TargetComponentName": {
              "type": "string",
              "pattern": "[a-zA-Z_\\-0-9]+",
              "minLength": 1,
              "maxLength": 256
            },
            "TargetEntityId": {
              "type": "string",
              "pattern": "[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}|
^[a-zA-Z0-9][a-zA-Z_\\-0-9.:]*[a-zA-Z0-9]+",
              "minLength": 1,
              "maxLength": 128
            }
          },
          "additionalProperties": false
        }
      },
      "additionalProperties": false
    },
    "DataType": {
      "description": "An object that specifies the data type of a property.",
      "type": "object",
      "properties": {
        "allowedValues": {
          "description": "The allowed values for this data type.",
          "type": "array",
          "minItems": 0,
          "maxItems": 50,
          "uniqueItems": false,
          "insertionOrder": false,
          "items": {
            "$ref": "#/definitions/DataValue"
          },
          "default": null
        },
        "nestedType": {
          "description": "The nested type in the data type.",
          "$ref": "#/definitions/DataType"
        },
        "relationship": {
          "description": "A relationship that associates a component with another
 component.",
```

```
          "$ref": "#/definitions/Relationship"
        },
        "type": {
          "description": "The underlying type of the data type.",
          "type": "string",
          "enum": [
            "RELATIONSHIP",
            "STRING",
            "LONG",
            "BOOLEAN",
            "INTEGER",
            "DOUBLE",
            "LIST",
            "MAP"
          ]
        },
        "unitOfMeasure": {
          "description": "The unit of measure used in this data type.",
          "type": "string",
          "pattern": ".*",
          "minLength": 1,
          "maxLength": 256
        }
      },
      "required": [
        "type"
      ],
      "additionalProperties": false
    },
    "PropertyDefinition": {
      "description": "An object that specifies information about a property.",
      "type": "object",
      "properties": {
        "configuration": {
          "description": "An object that specifies information about a property.",
          "patternProperties": {
            "[a-zA-Z_\\-0-9]+": {
              "type": "string",
              "pattern": "[a-zA-Z_\\-0-9]+",
              "minLength": 1,
              "maxLength": 256
            }
          },
          "existingJavaType": "java.util.Map<String,String>"
```

```
        },
        "dataType": {
          "description": "An object that contains information about the data type.",
          "$ref": "#/definitions/DataType"
        },
        "defaultValue": {
          "description": "An object that contains the default value.",
          "$ref": "#/definitions/DataValue"
        },
        "displayName": {
          "description": "An object that contains the default value.",
          "$ref": "#/definitions/DisplayName"
        },
        "isExternalId": {
          "description": "A Boolean value that specifies whether the property ID comes
from an external data store.",
          "type": "boolean",
          "default": null
        },
        "isRequiredInEntity": {
          "description": "A Boolean value that specifies whether the property is
required.",
          "type": "boolean",
          "default": null
        },
        "isStoredExternally": {
          "description": "A Boolean value that specifies whether the property is stored
externally.",
          "type": "boolean",
          "default": null
        },
        "isTimeSeries": {
          "description": "A Boolean value that specifies whether the property consists
of time series data.",
          "type": "boolean",
          "default": null
        }
      },
      "additionalProperties": false
    },
    "PropertyDefinitions": {
      "type": "object",
      "patternProperties": {
        "[a-zA-Z_\\-0-9]+": {
```

```
          "$ref": "#/definitions/PropertyDefinition"
        }
      },
      "additionalProperties": {
        "$ref": "#/definitions/PropertyDefinition"
      }
    },
    "Property": {
      "type": "object",
      "properties": {
        "definition": {
          "description": "The definition of the property",
          "$ref": "#/definitions/PropertyDefinition"
        },
        "value": {
          "description": "The value of the property.",
          "$ref": "#/definitions/DataValue"
        }
      },
      "additionalProperties": false
    },
    "Properties": {
      "type": "object",
      "patternProperties": {
        "[a-zA-Z_\\-0-9]+": {
          "$ref": "#/definitions/Property"
        }
      },
      "additionalProperties": {
        "$ref": "#/definitions/Property"
      }
    },
    "PropertyName": {
      "type": "string",
      "pattern": "[a-zA-Z_\\-0-9]+"
    },
    "PropertyGroup": {
      "description": "An object that specifies information about a property group.",
      "type": "object",
      "properties": {
        "groupType": {
          "description": "The type of property group.",
          "type": "string",
          "enum": [
```

```
            "TABULAR"
          ]
        },
        "propertyNames": {
          "description": "The list of property names in the property group.",
          "type": "array",
          "minItems": 1,
          "maxItems": 256,
          "uniqueItems": true,
          "insertionOrder": false,
          "items": {
            "$ref": "#/definitions/PropertyName"
          },
          "default": null
        }
      },
      "additionalProperties": false
    },
    "PropertyGroups": {
      "type": "object",
      "patternProperties": {
        "[a-zA-Z_\\-0-9]+": {
          "$ref": "#/definitions/PropertyGroup"
        }
      },
      "additionalProperties": {
        "$ref": "#/definitions/PropertyGroup"
      }
    },
    "Component": {
      "type": "object",
      "properties": {
        "componentTypeId": {
          "$ref": "#/definitions/ComponentTypeId"
        },
        "description": {
          "$ref": "#/definitions/Description"
        },
        "properties": {
          "description": "An object that maps strings to the properties to set in the
component type. Each string in the mapping must be unique to this object.",
          "$ref": "#/definitions/Properties"
        },
        "propertyGroups": {
```

```
          "description": "An object that maps strings to the property groups to set in
 the entity component. Each string in the mapping must be unique to this object.",
          "$ref": "#/definitions/PropertyGroups"
        }
      },
      "required": [
        "componentTypeId"
      ],
      "additionalProperties": false
    },
    "RequiredProperty": {
      "type": "string",
      "pattern": "[a-zA-Z_\\-0-9]+"
    },
    "LambdaFunction": {
      "type": "object",
      "properties": {
        "arn": {
          "type": "string",
          "pattern": "arn:((aws)|(aws-cn)|(aws-us-gov)|(\\${partition})):lambda:(([a-
z0-9-]+)|(\\${region})):([0-9]{12}|(\\${accountId})):function:[/a-zA-Z0-9_-]+",
          "minLength": 1,
          "maxLength": 128
        }
      },
      "additionalProperties": false,
      "required": [
        "arn"
      ]
    },
    "DataConnector": {
      "description": "The data connector.",
      "type": "object",
      "properties": {
        "isNative": {
          "description": "A Boolean value that specifies whether the data connector is
 native to IoT TwinMaker.",
          "type": "boolean"
        },
        "lambda": {
          "description": "The Lambda function associated with this data connector.",
          "$ref": "#/definitions/LambdaFunction"
        }
      },
```

```
      "additionalProperties": false
    },
    "Function": {
      "description": "The function of component type.",
      "type": "object",
      "properties": {
        "implementedBy": {
          "description": "The data connector.",
          "$ref": "#/definitions/DataConnector"
        },
        "requiredProperties": {
          "description": "The required properties of the function.",
          "type": "array",
          "minItems": 1,
          "maxItems": 256,
          "uniqueItems": true,
          "insertionOrder": false,
          "items": {
            "$ref": "#/definitions/RequiredProperty"
          },
          "default": null
        },
        "scope": {
          "description": "The scope of the function.",
          "type": "string",
          "enum": [
            "ENTITY",
            "WORKSPACE"
          ]
        }
      },
      "additionalProperties": false
    },
    "Entity": {
      "type": "object",
      "properties": {
        "description": {
          "description": "The description of the entity.",
          "$ref": "#/definitions/DescriptionWithDefault"
        },
        "entityId": {
          "$ref": "#/definitions/EntityId"
        },
        "entityExternalId": {
```

```
          "description": "The external ID of the entity.",
          "$ref": "#/definitions/ExternalId"
        },
        "entityName": {
          "$ref": "#/definitions/EntityName"
        },
        "parentEntityId": {
          "$ref": "#/definitions/ParentEntityId"
        },
        "tags": {
          "$ref": "#/definitions/Tags"
        },
        "components": {
          "description": "A map that sets information about a component.",
          "type": "object",
          "patternProperties": {
            "[a-zA-Z_\\-0-9]+": {
              "$ref": "#/definitions/Component"
            }
          },
          "additionalProperties": {
            "$ref": "#/definitions/Component"
          }
        }
      },
      "required": [
        "entityId",
        "entityName"
      ],
      "additionalProperties": false
    },
    "ComponentType": {
      "type": "object",
      "properties": {
        "description": {
          "description": "The description of the component type.",
          "$ref": "#/definitions/DescriptionWithDefault"
        },
        "componentTypeId": {
          "$ref": "#/definitions/ComponentTypeId"
        },
        "componentTypeExternalId": {
          "description": "The external ID of the component type.",
          "$ref": "#/definitions/ExternalId"
```

```
        },
        "componentTypeName": {
          "$ref": "#/definitions/ComponentTypeName"
        },
        "extendsFrom": {
          "description": "Specifies the parent component type to extend.",
          "type": "array",
          "minItems": 1,
          "maxItems": 256,
          "uniqueItems": true,
          "insertionOrder": false,
          "items": {
            "$ref": "#/definitions/ComponentTypeId"
          },
          "default": null
        },
        "functions": {
          "description": "a Map of functions in the component type. Each function's key
must be unique to this map.",
          "type": "object",
          "patternProperties": {
            "[a-zA-Z_\\-0-9]+": {
              "$ref": "#/definitions/Function"
            }
          },
          "additionalProperties": {
            "$ref": "#/definitions/Function"
          }
        },
        "isSingleton": {
          "description": "A Boolean value that specifies whether an entity can have
more than one component of this type.",
          "type": "boolean",
          "default": false
        },
        "propertyDefinitions": {
          "description": "An map of the property definitions in the component type.
Each property definition's key must be unique to this map.",
          "$ref": "#/definitions/PropertyDefinitions"
        },
        "propertyGroups": {
          "description": "An object that maps strings to the property groups to set in
the component type. Each string in the mapping must be unique to this object.",
          "$ref": "#/definitions/PropertyGroups"
```

```
        },
        "tags": {
          "$ref": "#/definitions/Tags"
        }
      },
      "required": [
        "componentTypeId"
      ],
      "additionalProperties": false
    },
    "EntityComponent": {
      "type": "object",
      "properties": {
        "entityId": {
          "$ref": "#/definitions/EntityId"
        },
        "componentName": {
          "$ref": "#/definitions/ComponentName"
        },
        "componentExternalId": {
          "description": "The external ID of the component.",
          "$ref": "#/definitions/ExternalId"
        },
        "componentTypeId": {
          "$ref": "#/definitions/ComponentTypeId"
        },
        "description": {
          "description": "The description of the component.",
          "$ref": "#/definitions/Description"
        },
        "properties": {
          "description": "An object that maps strings to the properties to set in the
component. Each string in the mapping must be unique to this object.",
          "$ref": "#/definitions/Properties"
        },
        "propertyGroups": {
          "description": "An object that maps strings to the property groups to set in
the component. Each string in the mapping must be unique to this object.",
          "$ref": "#/definitions/PropertyGroups"
        }
      },
      "required": [
        "entityId",
        "componentTypeId",
```

```
        "componentName"
      ],
      "additionalProperties": false
    }
  },
  "additionalProperties": false,
  "properties": {
    "entities": {
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$ref": "#/definitions/Entity"
      }
    },
    "componentTypes": {
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$ref": "#/definitions/ComponentType"
      }
    },
    "entityComponents": {
      "type": "array",
      "uniqueItems": false,
      "items": {
        "$ref": "#/definitions/EntityComponent"
      },
      "default": null
    }
  }
}
```

Here is an example that creates a new componentType called `component.type.intial` and creates an entity called `initial`:

```
{
  "componentTypes": [
    {
      "componentTypeId": "component.type.initial",
      "tags": {
        "key": "value"
      }
    }
```

```
  ],
  "entities": [
    {
      "entityName": "initial",
      "entityId": "initial"
    }
  ]
}
```

Here is an example that updates existing entities:

```
{
  "componentTypes": [
    {
      "componentTypeId": "component.type.initial",
      "description": "updated"
    }
  ],
  "entities": [
    {
      "entityName": "parent",
      "entityId": "parent"
    },
    {
      "entityName": "child",
      "entityId": "child",
      "components": {
        "testComponent": {
          "componentTypeId": "component.type.initial",
          "properties": {
            "testProperty": {
              "definition": {
                "configuration": {
                  "alias": "property"
                },
                "dataType": {
                  "relationship": {
                    "relationshipType": "parent",
                    "targetComponentTypeId": "test"
                  },
                  "type": "STRING",
                  "unitOfMeasure": "t"
                },
```

```
                "displayName": "displayName"
              }
            }
          }
        }
      },
      "parentEntityId": "parent"
    }
  ],
  "entityComponents": [
    {
      "entityId": "initial",
      "componentTypeId": "component.type.initial",
      "componentName": "entityComponent",
      "description": "additionalDescription",
      "properties": {
        "additionalProperty": {
          "definition": {
            "configuration": {
              "alias": "additionalProperty"
            },
            "dataType": {
              "type": "STRING"
            },
            "displayName": "additionalDisplayName"
          },
          "value": {
            "stringValue": "test"
          }
        }
      }
    }
  ]
}
```

AWS IoT TwinMaker data connectors

AWS IoT TwinMaker uses a connector-based architecture so that you can connect data from your own data store to AWS IoT TwinMaker. This means you don't need to migrate data prior to using AWS IoT TwinMaker. Currently, AWS IoT TwinMaker supports first-party connectors for AWS IoT SiteWise. If you store modeling and property data in AWS IoT SiteWise, then you don't need to implement your own connectors. If you store your modeling or property data in other data stores, such as Timestream, DynamoDB, or Snowflake, then you must implement AWS Lambda connectors with the AWS IoT TwinMaker data connector interface so that AWS IoT TwinMaker can invoke your connector when necessary.

Topics

AWS IoT TwinMaker data connectors

Connectors need access to your underlying data store to resolve sent queries and to return either results or an error.

To learn about the available connectors, their request interfaces, and their response interfaces, see the following topics.

For more information about the properties used in the connector interfaces, see the GetPropertyValueHistory API action.

> **Note**
>
> Some connectors have two timestamp fields in both the request and response interfaces for start time and end time properties. Both `startDateTime` and `endDateTime` use a long number to represent epoch second, which is no longer supported. To maintain backwards-compatibility, we still send a timestamp value to that field, but we recommend using the `startTime` and `endTime` fields that are consistent with our API timestamp format.

Topics

Schema initializer connector

You can use the schema initializer in the component type or entity lifecycle to fetch the component type or component properties from the underlying data source. The schema initializer automatically imports component type or component properties without explicitly calling an API action to set up `properties`.

SchemaInitializer request interface

```
{
  "workspaceId": "string",
  "entityId": "string",
  "componentName": "string",
  "properties": {
    // property name as key,
    // value is of type PropertyRequest
    "string": "PropertyRequest"
  }
}
```

> **Note**
>
> The map of properties in this request interface is a `PropertyRequest`. For more information, see PropertyRequest.

SchemaInitializer response interface

```
{
  "properties": {
    // property name as key,
    // value is of type PropertyResponse
    "string": "PropertyResponse"
  }
}
```

> **Note**
>
> The map of properties in this request interface is a `PropertyResponse`. For more information, see PropertyResponse.

DataReaderByEntity

DataReaderByEntity is a data plane connector that's used to get the time-series values of properties in a single component.

For information about the property types, syntax, and format of this connector, see the GetPropertyValueHistory API action.

DataReaderByEntity request interface

```
{
  "startDateTime": long, // In epoch sec, deprecated
  "startTime": "string", // ISO-8601 timestamp format
  "endDateTime": long, // In epoch sec, deprecated
  "endTime": "string", // ISO-8601 timestamp format
  "properties": {
    // A map of properties as in the get-entity API response
    // property name as key,
    // value is of type PropertyResponse
    "string": "PropertyResponse"
   },
  "workspaceId": "string",
  "selectedProperties": List:"string",
  "propertyFilters": List:PropertyFilter,
```

```
  "entityId": "string",
  "componentName": "string",
  "componentTypeId": "string",
  "interpolation": InterpolationParameters,
  "nextToken": "string",
  "maxResults": int,
  "orderByTime": "string"
  }
```

DataReaderByEntity response interface

```
{
  "propertyValues": [
    {
      "entityPropertyReference": EntityPropertyReference, // The same
 as EntityPropertyReference
      "values": [
        {
        "timestamp": long, // Epoch sec, deprecated
        "time": "string", // ISO-8601 timestamp format
        "value": DataValue // The same as DataValue
        }
      ]
    }
  ],
  "nextToken": "string"
}
```

DataReaderByComponentType

To get the time-series values of common properties that come from the same component type, use the data plane connector DataReaderByEntity. For example, if you define time-series properties in the component type and you have multiple components using that component type, then you can query those properties across all components in a given a time range. A common use case for this is when you want to query the alarm status of multiple components for a global view of your entities.

For information about the property types, syntax, and format of this connector, see the GetPropertyValueHistory API action.

DataReaderByComponentType request interface

```
{
  "startDateTime": long, // In epoch sec, deprecated
  "startTime": "string", // ISO-8601 timestamp format
  "endDateTime": long, // In epoch sec, deprecated
  "endTime": "string", // ISO-8601 timestamp format
  "properties": { // A map of properties as in the get-entity API response
    // property name as key,
    // value is of type PropertyResponse
    "string": "PropertyResponse"
   },
  "workspaceId": "string",
  "selectedProperties": List:"string",
  "propertyFilters": List:PropertyFilter,
  "componentTypeId": "string",
  "interpolation": InterpolationParameters,
  "nextToken": "string",
  "maxResults": int,
  "orderByTime": "string"
}
```

DataReaderByComponentType response interface

```
{
  "propertyValues": [
    {
      "entityPropertyReference": EntityPropertyReference, // The same
 as EntityPropertyReference
      "entityId": "string",
      "componentName": "string",
      "values": [
        {
        "timestamp": long, // Epoch sec, deprecated
        "time": "string", // ISO-8601 timestamp format
        "value": DataValue // The same as DataValue
        }
      ]
    }
  ],
  "nextToken": "string"
}
```

DataReader

DataReader is a data plane connector that can handle both the case of DataReaderByEntity and DataReaderByComponentType.

For information about the property types, syntax, and format of this connector, see the GetPropertyValueHistory API action.

DataReader request interface

The `EntityId` and `componentName` are optional.

```
{
  "startDateTime": long, // In epoch sec, deprecated
  "startTime": "string", // ISO-8601 timestamp format
  "endDateTime": long, // In epoch sec, deprecated
  "endTime": "string", // ISO-8601 timestamp format
  "properties": { // A map of properties as in the get-entity API response
    // property name as key,
    // value is of type PropertyRequest
    "string": "PropertyRequest"
  },

  "workspaceId": "string",
  "selectedProperties": List:"string",
  "propertyFilters": List:PropertyFilter,
  "entityId": "string",
  "componentName": "string",
  "componentTypeId": "string",
  "interpolation": InterpolationParameters,
  "nextToken": "string",
  "maxResults": int,
  "orderByTime": "string"
}
```

DataReader response interface

```
{
  "propertyValues": [
    {
      "entityPropertyReference": EntityPropertyReference, // The same
 as EntityPropertyReference
      "values": [
```

```
        {
        "timestamp": long, // Epoch sec, deprecated
        "time": "string", // ISO-8601 timestamp format
        "value": DataValue // The same as DataValue
        }
      ]
    }
  ],
  "nextToken": "string"
}
```

AttributePropertyValueReaderByEntity

AttributePropertyValueReaderByEntity is a data plane connector that you can use to fetch the value of static properties in a single entity.

For information about the property types, syntax, and format of this connector, see the GetPropertyValue API action.

AttributePropertyValueReaderByEntity request interface

```
{
  "properties": {
    // property name as key,
    // value is of type PropertyResponse
    "string": "PropertyResponse"
  }

  "workspaceId": "string",
  "entityId": "string",
  "componentName": "string",
  "selectedProperties": List:"string",
}
```

AttributePropertyValueReaderByEntity response interface

```
{
  "propertyValues": {
    "string": { // property name as key
        "propertyReference": EntityPropertyReference, // The same
 as EntityPropertyReference
        "propertyValue": DataValue // The same as DataValue
```

```
    }
}
```

DataWriter

DataWriter is a data plane connector that you can use to write time-series data points back to the underlying data store for properties in a single component.

For information about the property types, syntax, and format of this connector, see the BatchPutPropertyValues API action.

DataWriter request interface

```
{
  "workspaceId": "string",
  "properties": {
    // entity id as key
    "String": {
      // property name as key,
      // value is of type PropertyResponse
      "string": PropertyResponse
    }
  },
  "entries": [
    {
      "entryId": "string",
      "entityPropertyReference": EntityPropertyReference, // The same
 as EntityPropertyReference
      "propertyValues": [
        {
        "timestamp": long, // Epoch sec, deprecated
        "time": "string", // ISO-8601 timestamp format
        "value": DataValue // The same as DataValue
        }
      ]
    }
  ]
}
```

DataWriter response interface

```
{
```

```
  "errorEntries": [
    {
      "errors": List:BatchPutPropertyError // The value is a list of
 type BatchPutPropertyError
    }
  ]
}
```

Examples

The following JSON samples are examples of response and request syntax for multiple connectors.

- **SchemaInitializer**:

  The following examples show the schema initializer in a component type lifecycle.

  Request:

```
{
  "workspaceId": "myWorkspace",
  "properties": {
    "modelId": {
      "definition": {
          "dataType": { "type": "STRING" },
          "isExternalId": true,
          "isFinal": true,
          "isImported": false,
          "isInherited": false,
          "isRequiredInEntity": true,
          "isStoredExternally": false,
          "isTimeSeries": false,
          "defaultValue": {
              "stringValue": "myModelId"
          }
      },
      "value": {
          "stringValue": "myModelId"
      }
    },
    "tableName": {
      "definition": {
          "dataType": { "type": "STRING" },
          "isExternalId": false,
```

```
          "isFinal": false,
          "isImported": false,
          "isInherited": false,
          "isRequiredInEntity": false,
          "isStoredExternally": false,
          "isTimeSeries": false,
          "defaultValue": {
              "stringValue": "myTableName"
          }
      },
      "value": {
          "stringValue": "myTableName"
      }
    }
  }
}
```

Response:

```
{
  "properties": {
    "myProperty1": {
      "definition": {
        "dataType": {
          "type": "DOUBLE",
          "unitOfMeasure": "%"
        },
        "configuration": {
          "myProperty1Id": "idValue"
        },
        "isTimeSeries": true
      }
    },
    "myProperty2": {
      "definition": {
        "dataType": { "type": "STRING" },
        "isTimeSeries": false,
        "defaultValue": {
          "stringValue": "property2Value"
        }
      }
    }
  }
```

```
}
```

- **Schema initializer in entity lifecycle**:

  Request:

```
{
  "workspaceId": "myWorkspace",
  "entityId": "myEntity",
  "componentName": "myComponent",
  "properties": {
    "assetId": {
      "definition": {
          "dataType": { "type": "STRING" },
          "isExternalId": true,
          "isFinal": true,
          "isImported": false,
          "isInherited": false,
          "isRequiredInEntity": true,
          "isStoredExternally": false,
          "isTimeSeries": false
      },
      "value": {
          "stringValue": "myAssetId"
      }
    },
    "tableName": {
      "definition": {
          "dataType": { "type": "STRING" },
          "isExternalId": false,
          "isFinal": false,
          "isImported": false,
          "isInherited": false,
          "isRequiredInEntity": false,
          "isStoredExternally": false,
          "isTimeSeries": false
      },
      "value": {
          "stringValue": "myTableName"
      }
    }
  }
}
```

Response:

```
{
  "properties": {
    "myProperty1": {
      "definition": {
        "dataType": {
          "type": "DOUBLE",
          "unitOfMeasure": "%"
        },
        "configuration": {
          "myProperty1Id": "idValue"
        },
        "isTimeSeries": true
      }
    },
    "myProperty2": {
      "definition": {
        "dataType": { "type": "STRING" },
        "isTimeSeries": false
      },
      "value": {
        "stringValue": "property2Value"
      }
    }
  }
}
```

- **DataReaderByEntity and DataReader**:

  Request:

```
{
  "workspaceId": "myWorkspace",
  "entityId": "myEntity",
  "componentName": "myComponent",
  "selectedProperties": [
    "Temperature",
    "Pressure"
  ],
  "startTime": "2022-04-07T04:04:42Z",
  "endTime": "2022-04-07T04:04:45Z",
```

```
  "maxResults": 4,
  "orderByTime": "ASCENDING",
  "properties": {
      "assetId": {
          "definition": {
              "dataType": { "type": "STRING" },
              "isExternalId": true,
              "isFinal": true,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": true,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "myAssetId"
          }
      },
      "Temperature": {
          "definition": {
              "configuration": {
                  "temperatureId": "xyz123"
              },
              "dataType": {
                  "type": "DOUBLE",
                  "unitOfMeasure": "DEGC"
              },
              "isExternalId": false,
              "isFinal": false,
              "isImported": true,
              "isInherited": false,
              "isRequiredInEntity": false,
              "isStoredExternally": false,
              "isTimeSeries": true
          }
      },
      "Pressure": {
          "definition": {
              "configuration": {
                  "pressureId": "xyz456"
              },
              "dataType": {
                  "type": "DOUBLE",
                  "unitOfMeasure": "MPA"
```

```
            },
            "isExternalId": false,
            "isFinal": false,
            "isImported": true,
            "isInherited": false,
            "isRequiredInEntity": false,
            "isStoredExternally": false,
            "isTimeSeries": true
        }
    }
  }
}
```

Response:

```
{
  "propertyValues": [
    {
      "entityPropertyReference": {
        "entityId": "myEntity",
        "componentName": "myComponent",
        "propertyName": "Temperature"
      },
      "values": [
        {
          "time": "2022-04-07T04:04:42Z",
          "value": {
            "doubleValue": 588.168
          }
        },
        {
          "time": "2022-04-07T04:04:43Z",
          "value": {
            "doubleValue": 592.4224
          }
        }
      ]
    }
  ],
  "nextToken": "qwertyuiop"
}
```

- **AttributePropertyValueReaderByEntity**:

Request:

```
{
  "workspaceId": "myWorkspace",
  "entityId": "myEntity",
  "componentName": "myComponent",
  "selectedProperties": [
    "manufacturer",
  ],
  "properties": {
    "assetId": {
      "definition": {
        "dataType": { "type": "STRING" },
        "isExternalId": true,
        "isFinal": true,
        "isImported": false,
        "isInherited": false,
        "isRequiredInEntity": true,
        "isStoredExternally": false,
        "isTimeSeries": false
      },
      "value": {
          "stringValue": "myAssetId"
      }
    },
    "manufacturer": {
      "definition": {
        "dataType": { "type": "STRING" },
        "configuration": {
            "manufacturerPropId": "M001"
        },
        "isExternalId": false,
        "isFinal": false,
        "isImported": false,
        "isInherited": false,
        "isRequiredInEntity": false,
        "isStoredExternally": true,
        "isTimeSeries": false
      }
    }
  }
}
```

Response:

```
{
  "propertyValues": {
    "manufacturer": {
      "propertyReference": {
        "propertyName": "manufacturer",
        "entityId": "myEntity",
        "componentName": "myComponent"
      },
      "propertyValue": {
        "stringValue": "Amazon"
      }
    }
  }
}
```

- **DataWriter**:

  Request:

```
{
  "workspaceId": "myWorkspaceId",
  "properties": {
    "myEntity": {
      "Temperature": {
          "definition": {
              "configuration": {
                  "temperatureId": "xyz123"
              },
              "dataType": {
                  "type": "DOUBLE",
                  "unitOfMeasure": "DEGC"
              },
              "isExternalId": false,
              "isFinal": false,
              "isImported": true,
              "isInherited": false,
              "isRequiredInEntity": false,
              "isStoredExternally": false,
              "isTimeSeries": true
```

```
          }
        }
      }
    },
    "entries": [
      {
        "entryId": "myEntity",
        "entityPropertyReference": {
          "entityId": "myEntity",
          "componentName": "myComponent",
          "propertyName": "Temperature"
        },
        "propertyValues": [
          {
            "timestamp": 1626201120,
            "value": {
              "doubleValue": 95.6958
            }
          },
          {
            "timestamp": 1626201132,
            "value": {
              "doubleValue": 80.6959
            }
          }
        ]
      }
    ]
}
```

Response:

```
{
    "errorEntries": [
        {
            "errors": [
                {
                    "errorCode": "409",
                    "errorMessage": "Conflict value at same timestamp",
                    "entry": {
                        "entryId": "myEntity",
                        "entityPropertyReference": {
```

```
                            "entityId": "myEntity",
                            "componentName": "myComponent",
                            "propertyName": "Temperature"
                        },
                        "propertyValues": [
                            "time": "2022-04-07T04:04:42Z",
                            "value": {
                                "doubleValue": 95.6958
                            }
                        ]
                    }
                }
            ]
        }
    ]
}
```

AWS IoT TwinMaker Athena tabular data connector

With the Athena tabular data connector, you can access and use your Athena data stores in AWS IoT TwinMaker. You can use your Athena data to build digital twins without an intensive data migration effort. You can either use the prebuilt connector or create a custom Athena connector to access data from your Athena data sources.

AWS IoT TwinMaker Athena data connector prerequisites

Before you use the Athena tabular data connector, complete the following prerequisites:

- Create managed Athena tables and their associated Amazon S3 resources. For information on using Athena, see the Athena documentation.
- Create an AWS IoT TwinMaker workspace. You can create a workspace in the AWS IoT TwinMaker console.
- Update your workspace IAM role with Athena permissions. For more information, see Modify your workspace IAM role to use the Athena data connector.
- Become familiar with AWS IoT TwinMaker's entity-component system and how to create entities. For more information, see Create your first entity.
- Become familiar with AWS IoT TwinMaker's data connectors. For more information, see AWS IoT TwinMaker data connectors.

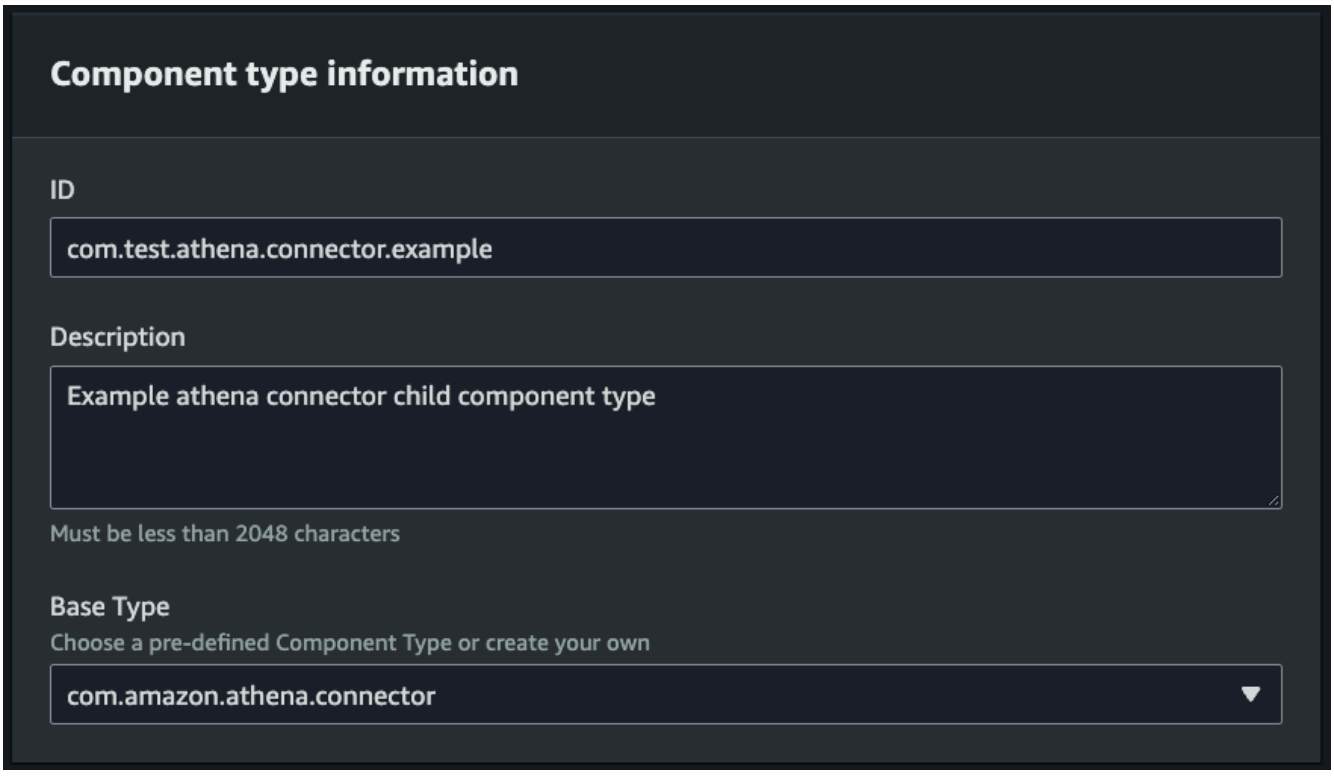
Using the Athena data connector

To use the Athena data connector, you must create a component, using the Athena connector as the component type. Then you attach the component to an entity within your scene for use in AWS IoT TwinMaker.

Create a component type with the Athena data connector

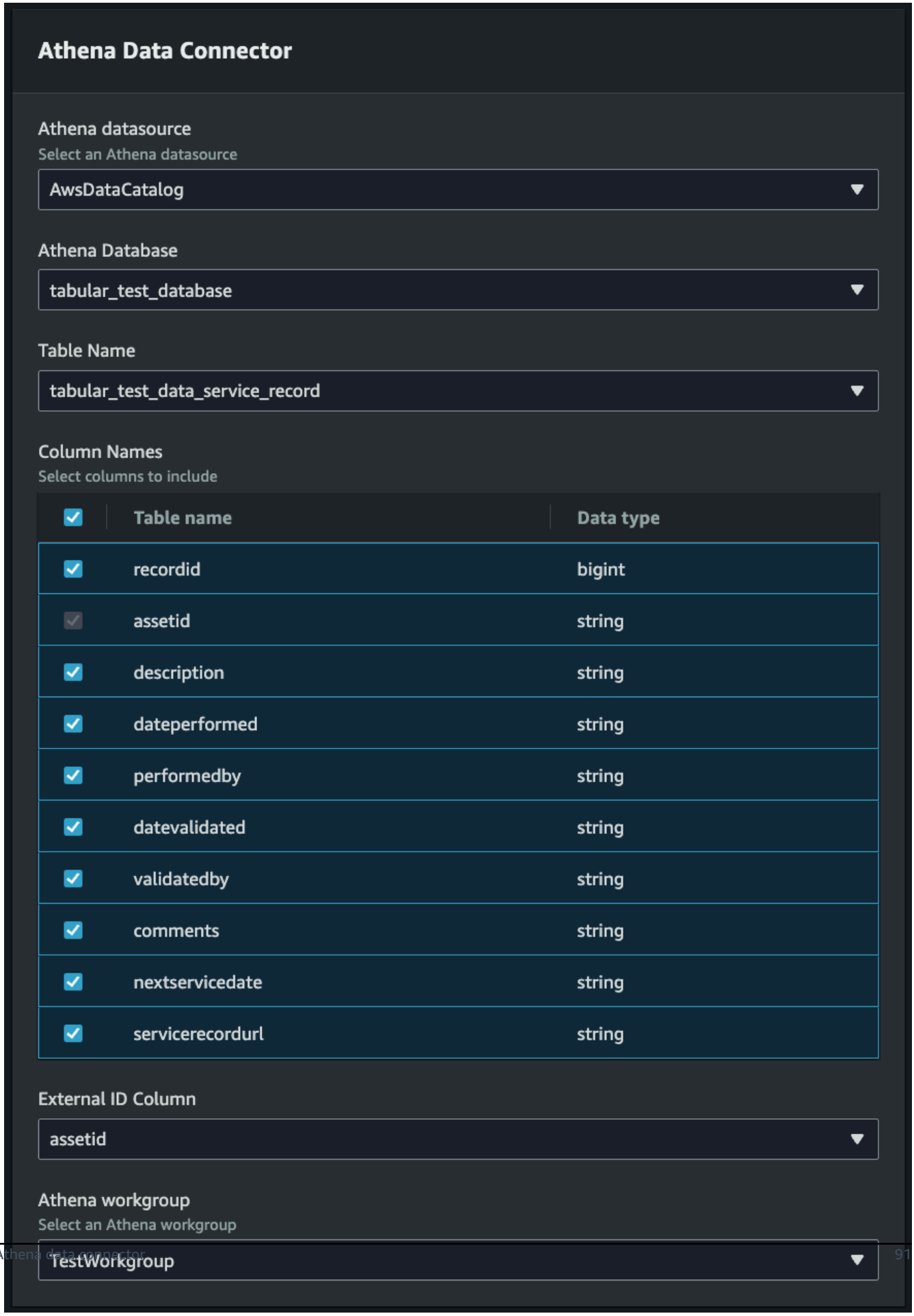
Use this procedure to create an AWS IoT TwinMaker component type with the Athena tabular data connector:

1. Navigate to the AWS IoT TwinMaker console.
2. Open an existing workspace or create a new one.
3. From the left side navigation menu, choose **Component types**, and select **Create component type** to open the component type creation page.
4. On the **Create component type** page, fill in the **ID** field with an ID that matches your use case.



5. Choose the **Base type**. From the dropdown list, select the Athena tabular data connector which is labeled as **com.amazon.athena.connector**.

6. Configure the component type's **Data source** by choosing Athena resources for the following fields:
   - Choose an **Athena datasource**.
   - Choose an **Athena database**.
   - Choose a **Table name**.
   - Choose a **Athena workGroup**.
7. Once you have selected the Athena resources you want to use as the data source, choose which columns from the table you want to include.
8. Select an **External ID column name**. Select a column from the previous step to serve as the external ID column. The external Id is the id that's used to represent an Athena asset and map it to an AWS IoT TwinMaker entity.


Athena Data Connector
Athena datasource
Select an Athena datasource
AwsDataCatalog
Athena Database
tabular_test_database
Table Name
tabular_test_data_service_record
Column Names
Select columns to include
Table name
Data type
recordid
bigint
assetid
string
description
string
dateperformed
string
performedby
string
datevalidated
string
validatedby
string
comments
string
nextservicedate
string
servicerecordurl
string
External ID Column
assetid
Athena workgroup
Select an Athena workgroup
TestWorkgroup

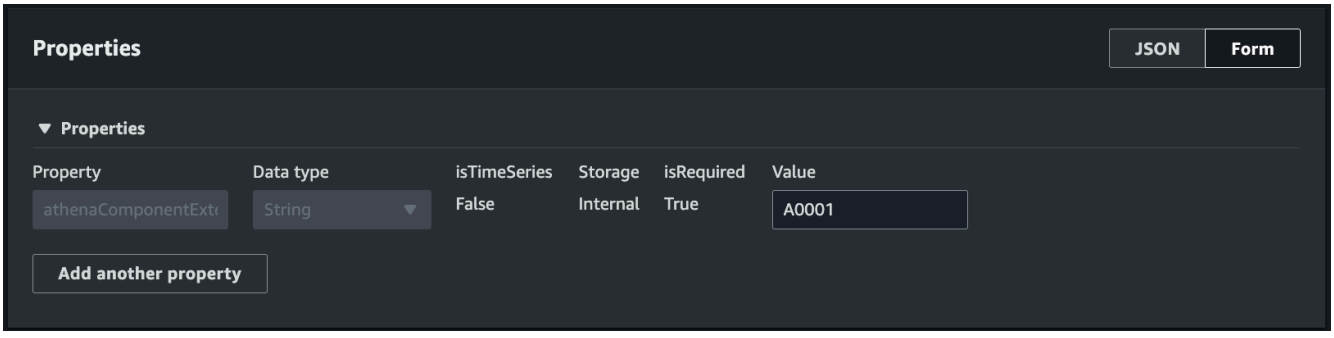
9. **(Optional)** Add AWS tags to these resources, so you can group and organize them.
10. Choose **Create component type** to finish creating the component type.

Create a component with the Athena data connector type and attach it to an entity

Use this procedure to create an AWS IoT TwinMaker component with the Athena tabular data connector and attach it to an entity:

> **Note**
>
> You must have an existing component type that uses the Athena tabular data connector as a data source in order to complete this procedure. See the previous procedure **Create a component type with the Athena data connector** before starting this walkthrough.

1. Navigate to the AWS IoT TwinMaker console.
2. Open an existing workspace or create a new one.
3. From the left side navigation menu choose **Entities**, and select the entity you want to add the component to or create a new entity.
4. Create a new entity.
5. Next select **Add component.**, fill in the **Component name** field with a name that match your use case.
6. From the **Component type** drop down menu select the **component type ID** that you created in the previous procedure.
7. Enter **Component information**, a **Component Name**, and select the child ComponentType created previously. This is the ComponentType you created with the Athena data connector.
8. In the **Properties** section, enter the **athenaComponentExternalId** for the component.



9. Choose **Add component** to finish creating the component.

You have now successfully created a component with the Athena data connector as the component type and attached it to an entity.

Using the Athena tabular data connector JSON reference

The following example is the full the JSON reference for the Athena tabular data connector. Use this as a resource to create custom data connectors and component types.

```
{
    "componentTypeId": "com.amazon.athena.connector",
    "description": "Athena connector for syncing tabular data",
    "workspaceId":"AmazonOwnedTypesWorkspace",
    "propertyGroups": {
        "tabularPropertyGroup": {
            "groupType": "TABULAR",
            "propertyNames": []
        }
    },
    "propertyDefinitions": {
        "athenaDataSource": {
            "dataType": { "type": "STRING" },
            "isRequiredInEntity": true
        },
        "athenaDatabase": {
            "dataType": { "type": "STRING" },
            "isRequiredInEntity": true
        },
        "athenaTable": {
            "dataType": { "type": "STRING" },
            "isRequiredInEntity": true
        },
        "athenaWorkgroup": {
            "dataType": { "type": "STRING" },
            "isRequiredInEntity": true
        },
        "athenaExternalIdColumnName": {
            "dataType": { "type": "STRING" },
            "isRequiredInEntity": true,
            "isExternalId": false
        },
        "athenaComponentExternalId": {
            "dataType": { "type": "STRING" },
            "isStoredExternally": false,
```

```
            "isRequiredInEntity": true,
            "isExternalId": true
        }
    },
    "functions": {
        "tabularDataReaderByEntity": {
            "implementedBy": {
                "isNative": true
            }
        }
    }
}
```

Using the Athena data connector

You can surface your entities that are using Athena tables in Grafana. For more information, see AWS IoT TwinMaker Grafana dashboard integration.

Read the Athena documentation for information on creating and using Athena tables to store data.

Troubleshooting the Athena data connector

This topic covers common issues you may encounter when configuring the Athena data connector.

Athena workgroup location:

When creating Athena connector componentType, an Athena workgroup has to have output location setup. See How workgroups work.

Missing IAM role permissions:

The AWS IoT TwinMaker; workspace role may be missing Athena API access permission when creating a componentType, adding a Ca component to an entity, or running the GetPropertyValue API. To update IAM permissions see Create and manage a service role for AWS IoT TwinMaker.

Visualize Athena tabular data in Grafana

A Grafana plugin is also available to visualize your tabular data on Grafana a dashboard panel with additional features such as sorting and filtering based on selected properties without making API

calls to AWS IoT TwinMaker, or interactions with Athena. This topic shows you how to configure Grafana to visualize Athena tabular data.

Prerequisites

Before configuring a Grafana panel for visualizing Athena tabular data, review the following prerequisites:

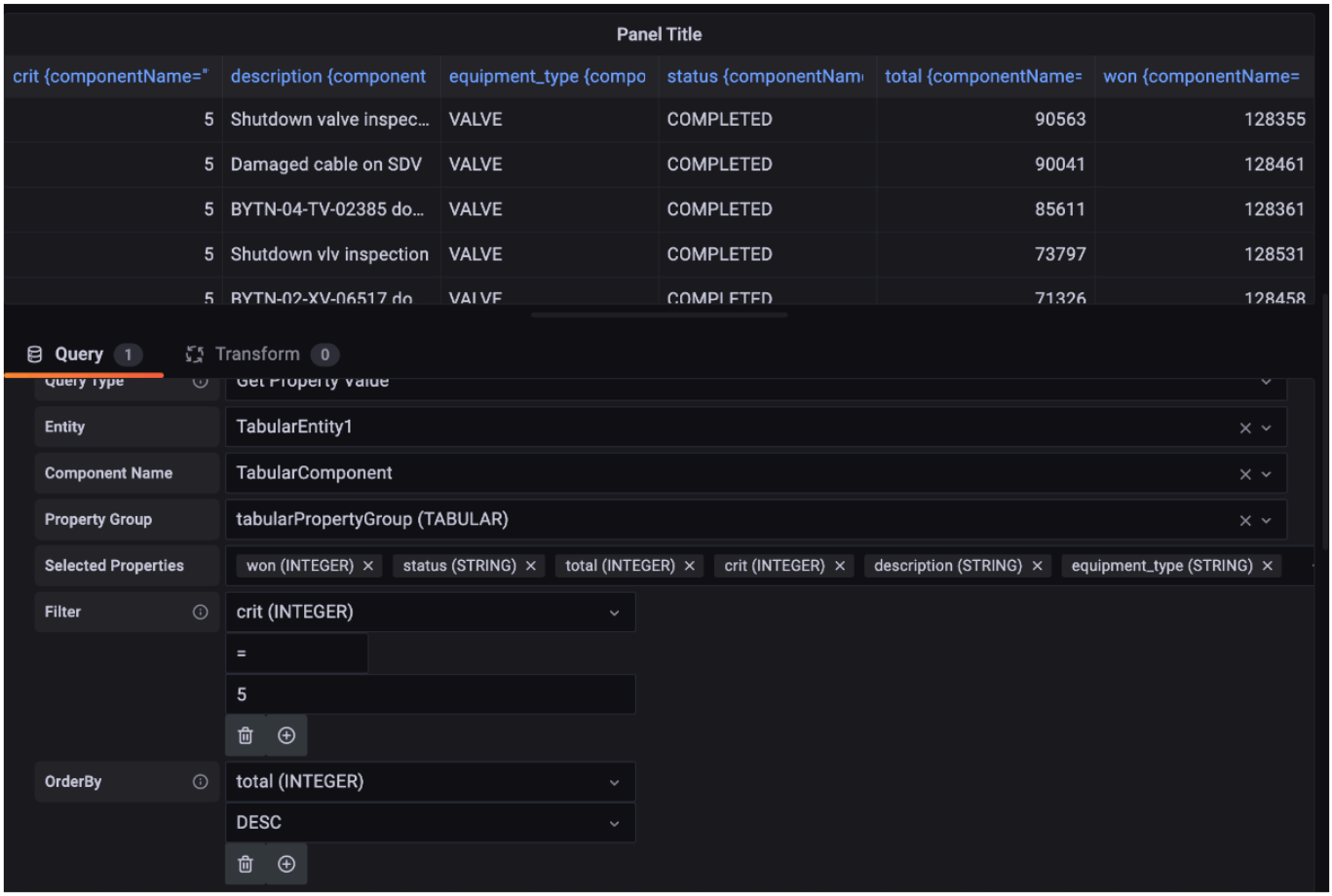
- You have set up a Grafana environment. For more information see, AWS IoT TwinMaker Grafana integration.
- You can configure a Grafana datasource. For more information see, Grafana AWS IoT TwinMaker.
- You are familiar with creating a new dashboard and add a new panel.

Visualize Athena tabular data in Grafana

This procedure shows you how to setup a Grafana panel to visualize Athena tabular data.

1. Open your AWS IoT TwinMaker Grafana dashboard.
2. Select the **Table** panel in the panel settings.
3. Select your datasource in the query configuration.
4. Select the **Get Property Value** query.
5. Select an entity.
6. Select a component that has a componentType that extends the **Athena base component type**.
7. Select the property group of your Athena table.
8. Select any number of properties from the property group.
9. Configure the tabular conditions through a list of filters and property orders. With the following options:

    - **Filter**: define an expression for a property value to filter your data.
    - **OrderBy**: specify whether data should be returned in ascending or descending order for a property.



Developing AWS IoT TwinMaker time-series data connectors

This section explains how to develop a time-series data connector in a step-by-step process. Additionally, we present an example time-series data connector based of the entire cookie factory sample, which includes 3D models, entities, components, alarms, and connectors. The cookie factory sample source is available on the AWS IoT TwinMaker samples GitHub repository .

Topics

- AWS IoT TwinMaker time-series data connector prerequisites
- Time-series data connector background
- Developing a time-series data connector
- Improving your data connector
- Testing your connector

- Security
- Creating AWS IoT TwinMaker resources
- What's next
- AWS IoT TwinMaker cookie factory example time-series connector

AWS IoT TwinMaker time-series data connector prerequisites

Before developing your time-series data connector, we recommend that you complete the following tasks:

- Create an AWS IoT TwinMaker workspace.
- Create AWS IoT TwinMaker component types.
- Create AWS IoT TwinMaker entities.
- (Optional) Read Using and creating component types.
- (Optional) Read AWS IoT TwinMaker data connector interface to get a general understanding of AWS IoT TwinMaker data connectors.

> **Note**
>
> For an example of a fully implemented connector, see our cookie factory example implementation.

Time-series data connector background

Imagine you are working with a factory that has a set of cookie mixers and a water tank. You would like to build AWS IoT TwinMaker digital twins of these physical entities so that you can monitor their operational states by checking various time-series metrics.

You have on-site sensors set up and you are already streaming measurement data into a Timestream database. You want to be able to view and organize the measurement data in AWS IoT TwinMaker with minimal overhead. You can accomplish this task by using a time-series data connector. The following image shows an example telemetry table, which is populated through the use of a time-series connector.

Rows returned (1000+)

Results are paginated. Scroll through the result pages to see more query results.

Filter

1 2 3 4 5 6 7 ... 100

TelemetryAssetId	TelemetryAssetType	measure_name	time	measure_value::varchar	measure_value::double
Mixer_22_680b5b8e-1afe-4a77-87ab-834fbe5ba01e	Mixer	Temperature	2022-04-19 00:28:00.241000000	-	99.1292877197266
Mixer_20_0568f25f-116c-429c-a974-5ceec065a6ac	Mixer	RPM	2022-04-19 00:28:00.241000000	-	59.4233207702637
Mixer_22_680b5b8e-1afe-4a77-87ab-834fbe5ba01e	Mixer	RPM	2022-04-19 00:28:00.241000000	-	59.9421195983887
Mixer_24_7ff0b75b-f0fa-43f0-bc89-b96337586d00	Mixer	Temperature	2022-04-19 00:28:00.241000000	-	99.1292877197266
Mixer_25_cf42effc-ba19-48ba-bbc3-d21d2508ce31	Mixer	RPM	2022-04-19 00:28:00.241000000	-	59.8453979492188
Mixer_20_0568f25f-116c-429c-a974-5ceec065a6ac	Mixer	Temperature	2022-04-19 00:28:00.241000000	-	99.1292877197266
Mixer_24_7ff0b75b-f0fa-43f0-bc89-b96337586d00	Mixer	RPM	2022-04-19 00:28:00.241000000	-	60.4532585144043
Mixer_15_0bb566cd-d6f3-4804-9fe1-7d2abcad82d0	Mixer	RPM	2022-04-19 00:28:00.241000000	-	58.397144317627
Mixer_2_d8e76844-e739-4845-a748-a83983279376	Mixer	RPM	2022-04-19 00:28:00.241000000	-	60.206958770752
Mixer_6_b66db3d3-c144-47b5-afb9-3a0150c53456	Mixer	RPM	2022-04-19 00:28:00.241000000	-	60.206958770752

The datasets and the Timestream table used in this screenshot are available in the AWS IoT TwinMaker samples GitHub repository. Also see the cookie factory example connector for the implementation, which produces the result shown in the preceding screenshot.

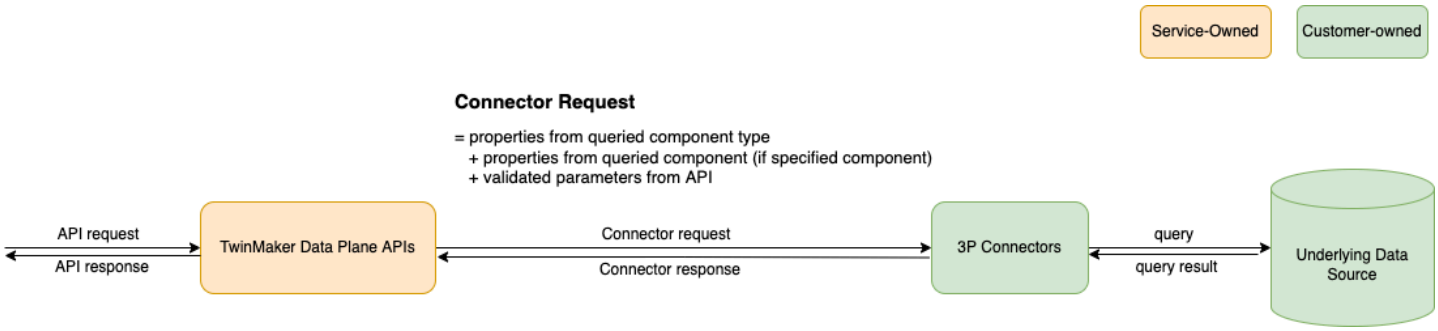
Time-series data connector data flow

For data plane queries, AWS IoT TwinMaker fetches the corresponding properties of both components and component types from components and component types definitions. AWS IoT TwinMaker forwards properties to AWS Lambda functions along with any API query parameters in the query.

AWS IoT TwinMaker uses Lambda functions to access and resolve queries from data sources and return the results of those queries. The Lambda functions use the component and component type properties from the data plane to resolve the initial request.

The results of the Lambda query are mapped to an API response and returned to you.

AWS IoT TwinMaker defines the data connector interface and uses that to interact with Lambda functions. Using data connectors, you can query your data source from AWS IoT TwinMaker API without any data migration efforts. The following image outlines the basic data flow described in the previous paragraphs.



Developing a time-series data connector

The following procedure outlines a development model that incrementally builds up to a functional time-series data connector. The basic steps are as follows:

1. **Create a valid basic component type**

   In a component type, you define common properties that are shared across your components. To learn more about defining component types, see Using and creating component types.

   AWS IoT TwinMaker uses an entity-component modeling pattern so each component is attached to an entity. We recommend that you model each physical item as an entity and model different data sources with their own component types.

   The following example shows a Timestream template component type with one property:

```
{"componentTypeId": "com.example.timestream-telemetry",
    "workspaceId": "MyWorkspace",
    "functions": {
        "dataReader": {
            "implementedBy": {
                "lambda": {
                    "arn": "lambdaArn"
                }
            }
        }
    },
    "propertyDefinitions": {
        "telemetryType": {
            "dataType": { "type": "STRING" },
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "telemetryId": {
            "dataType": { "type": "STRING" },
            "isExternalId": true,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
```

```
        "Temperature": {
            "dataType": { "type": "DOUBLE" },
            "isExternalId": false,
            "isTimeSeries": true,
            "isStoredExternally": true,
            "isRequiredInEntity": false
        }
    }
}
```

The key elements of the component type are the following:

- The `telemetryId` property identifies the unique key of the physical item in the corresponding data source. The data connector uses this property as a filter condition to only query values associated with the given item. Additionally, if you include the `telemetryId` property value in the data plane API response, then the client side takes the ID and can perform a reverse lookup if necessary.
- The `lambdaArn` field identifies the Lambda function with which the component type engages.
- The `isRequiredInEntity` flag enforces the ID creation. This flag is required so that when the component is created, the item's ID is also instantiated.
- The `TelemetryId` is added to the component type as an external id so that the item can be identified in the Timestream table.

2. **Create a component with the component type**

   To use the component type you created, you must create a component and attach it to the entity from which you wish to retrieve data. The following steps detail the process of creating that component:

   a. Navigate to the AWS IoT TwinMaker console.

   b. Select and open the same workspace in which you created the component types.

   c. Navigate to the entity page.

   d. Create a new entity or select an existing entity from the table.

   e. Once you have selected the entity you wish to use, choose **Add component** to open the **Add component** page.

   f. Give the component a name and for the **Type**, choose the component type you created with the template in **1. Create a valid basic component type**.

3. **Make your component type call a Lambda connector**

   The Lambda connector needs to access the data source and generate the query statement based on the input and forward it to the data source. The following example shows a JSON request template that does this.

```
{
  "workspaceId": "MyWorkspace",
  "entityId": "MyEntity",
  "componentName": "TelemetryData",
  "selectedProperties": ["Temperature"],
  "startTime": "2022-08-25T00:00:00Z",
  "endTime": "2022-08-25T00:00:05Z",
  "maxResults": 3,
  "orderByTime": "ASCENDING",
  "properties": {
      "telemetryType": {
          "definition": {
              "dataType": { "type": "STRING" },
              "isExternalId": false,
              "isFinal": false,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": false,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "Mixer"
          }
      },
      "telemetryId": {
          "definition": {
              "dataType": { "type": "STRING" },
              "isExternalId": true,
              "isFinal": true,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": true,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
```

```
                "stringValue": "item_A001"
            }
        },
        "Temperature": {
            "definition": {
                "dataType": { "type": "DOUBLE", },
                "isExternalId": false,
                "isFinal": false,
                "isImported": true,
                "isInherited": false,
                "isRequiredInEntity": false,
                "isStoredExternally": false,
                "isTimeSeries": true
            }
        }
    }
}
```

The key elements of the request:

- The `selectedProperties` is a list you populate with the properties for which you want Timestream measurements.
- The `startDateTime`, `startTime`, `EndDateTime`, and `endTime` fields specify a time range for the request. This determines the sample range for the measurements returned.
- The `entityId` is the name of the entity from which you are querying data.
- The `componentName` is the name of the component from which you are querying data.
- Use the `orderByTime` field to organize the order in which the results are displayed.

In the preceding example request, we would expect to get a series of samples for the selected properties during the given time window for the given item, with the selected time order. The response statement can be summarized as the following:

```
{
  "propertyValues": [
    {
      "entityPropertyReference": {
        "entityId": "MyEntity",
        "componentName": "TelemetryData",
        "propertyName": "Temperature"
      },
```

```
      "values": [
        {
          "time": "2022-08-25T00:00:00Z",
          "value": {
            "doubleValue": 588.168
          }
        },
        {
          "time": "2022-08-25T00:00:01Z",
          "value": {
            "doubleValue": 592.4224
          }
        },
        {
          "time": "2022-08-25T00:00:02Z",
          "value": {
            "doubleValue": 594.9383
          }
        }
      ]
    }
  ],
  "nextToken": "..."
}
```

4. **Update your component type to have two properties**

   The following JSON template shows a valid component type with two properties:

```
{
    "componentTypeId": "com.example.timestream-telemetry",
    "workspaceId": "MyWorkspace",
    "functions": {
        "dataReader": {
            "implementedBy": {
                "lambda": {
                    "arn": "lambdaArn"
                }
            }
        }
    },
    "propertyDefinitions": {
        "telemetryType": {
            "dataType": { "type": "STRING" },
```

```
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "telemetryId": {
            "dataType": { "type": "STRING" },
            "isExternalId": true,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "Temperature": {
            "dataType": { "type": "DOUBLE" },
            "isExternalId": false,
            "isTimeSeries": true,
            "isStoredExternally": true,
            "isRequiredInEntity": false
        },
        "RPM": {
            "dataType": { "type": "DOUBLE" },
            "isExternalId": false,
            "isTimeSeries": true,
            "isStoredExternally": true,
            "isRequiredInEntity": false
        }
    }
}
```

5. **Update the Lambda connector to handle the second property**

   The AWS IoT TwinMaker data plane API supports querying multiple properties in a single request, and AWS IoT TwinMaker follows a single request to a connector by providing a list of `selectedProperties`.

   The following JSON request shows a modified template that now supports a request for two properties.

```
{
  "workspaceId": "MyWorkspace",
  "entityId": "MyEntity",
  "componentName": "TelemetryData",
  "selectedProperties": ["Temperature", "RPM"],
```

```
"startTime": "2022-08-25T00:00:00Z",
"endTime": "2022-08-25T00:00:05Z",
"maxResults": 3,
"orderByTime": "ASCENDING",
"properties": {
    "telemetryType": {
        "definition": {
            "dataType": { "type": "STRING" },
            "isExternalId": false,
            "isFinal": false,
            "isImported": false,
            "isInherited": false,
            "isRequiredInEntity": false,
            "isStoredExternally": false,
            "isTimeSeries": false
        },
        "value": {
            "stringValue": "Mixer"
        }
    },
    "telemetryId": {
        "definition": {
            "dataType": { "type": "STRING" },
            "isExternalId": true,
            "isFinal": true,
            "isImported": false,
            "isInherited": false,
            "isRequiredInEntity": true,
            "isStoredExternally": false,
            "isTimeSeries": false
        },
        "value": {
            "stringValue": "item_A001"
        }
    },
    "Temperature": {
        "definition": {
            "dataType": { "type": "DOUBLE" },
            "isExternalId": false,
            "isFinal": false,
            "isImported": true,
            "isInherited": false,
            "isRequiredInEntity": false,
            "isStoredExternally": false,
```

```
                "isTimeSeries": true
            }
        },
        "RPM": {
            "definition": {
                "dataType": { "type": "DOUBLE" },
                "isExternalId": false,
                "isFinal": false,
                "isImported": true,
                "isInherited": false,
                "isRequiredInEntity": false,
                "isStoredExternally": false,
                "isTimeSeries": true
            }
        }
    }
}
```

Similarly, the corresponding response is also updated, as shown in the following example:

```
{
  "propertyValues": [
    {
      "entityPropertyReference": {
        "entityId": "MyEntity",
        "componentName": "TelemetryData",
        "propertyName": "Temperature"
      },
      "values": [
        {
          "time": "2022-08-25T00:00:00Z",
          "value": {
            "doubleValue": 588.168
          }
        },
        {
          "time": "2022-08-25T00:00:01Z",
          "value": {
            "doubleValue": 592.4224
          }
        },
        {
          "time": "2022-08-25T00:00:02Z",
```

```
          "value": {
            "doubleValue": 594.9383
          }
        }
      ]
    },
    {
      "entityPropertyReference": {
        "entityId": "MyEntity",
        "componentName": "TelemetryData",
        "propertyName": "RPM"
      },
      "values": [
        {
          "time": "2022-08-25T00:00:00Z",
          "value": {
            "doubleValue": 59
          }
        },
        {
          "time": "2022-08-25T00:00:01Z",
          "value": {
            "doubleValue": 60
          }
        },
        {
          "time": "2022-08-25T00:00:02Z",
          "value": {
            "doubleValue": 60
          }
        }
      ]
    }
  ],
  "nextToken": "..."
}
```

Note

In terms of the pagination for this case, the page size in the request applies to all properties. This means that with five properties in the query and a page size of 100, if

there are enough data points in the source, you should expect to see 100 data points per property, with 500 data points in total.

For an example implementation, see Snowflake connector sample on GitHub.

Improving your data connector

Handling exceptions

It is safe for the Lambda connector to throw exceptions. In the data plane API call, the AWS IoT TwinMaker service waits for the Lambda function to return a response. If the connector implementation throws an exception, AWS IoT TwinMaker translates the exception type to be `ConnectorFailure`, making the API client aware that an issue happened inside the connector.

Handling pagination

In the example, Timestream provides a utility function which can help support pagination natively. However, for some other query interfaces, such as SQL, it might need extra effort to implement an efficient pagination algorithm. There is a Snowflake connector example that handles pagination in an SQL interface.

When the new token is returned to AWS IoT TwinMaker through the connector response interface, the token is encrypted before being returned to the API client. When the token is included in another request, AWS IoT TwinMaker decrypts it before forwarding it to the Lambda connector. We recommend that you avoid adding sensitive information to the token.

Testing your connector

Though you can still update the implementation after you link the connector to the component type, we strongly recommend you verify the Lambda connector before integrating with AWS IoT TwinMaker.

There are multiple ways to test your Lambda connector: you can test the Lambda connector in the Lambda console or locally in the AWS CDK.

For more information on testing your Lambda functions, see Testing Lambda functions and Locally testing AWS CDK applications.

Security

For documentation on security best practices with Timestream, see Security in Timestream.

For an example of SQL injection prevention, see the following Python script in AWS IoT TwinMaker Samples GitHub Repository.

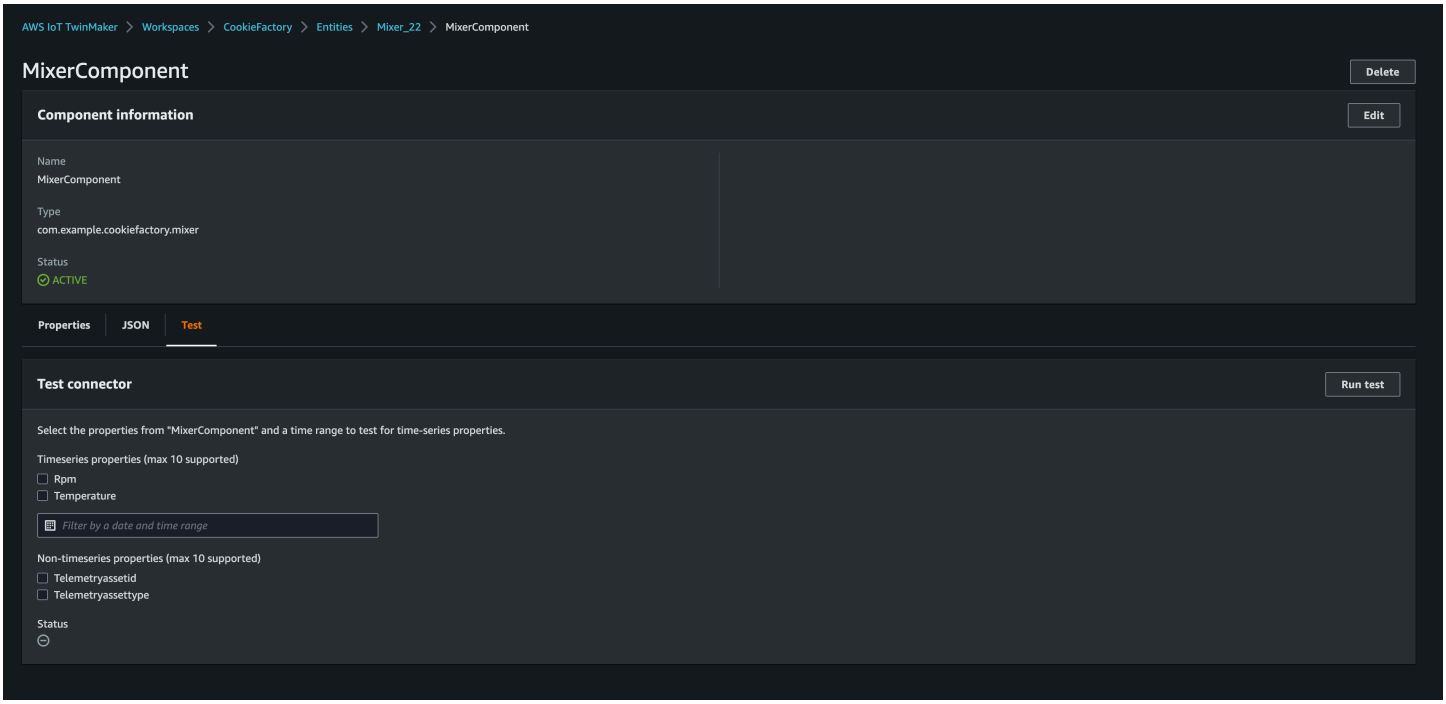
Creating AWS IoT TwinMaker resources

Once you have implemented the Lambda function, you can create AWS IoT TwinMaker resources such as component types, entities, and components through the AWS IoT TwinMaker console or API.

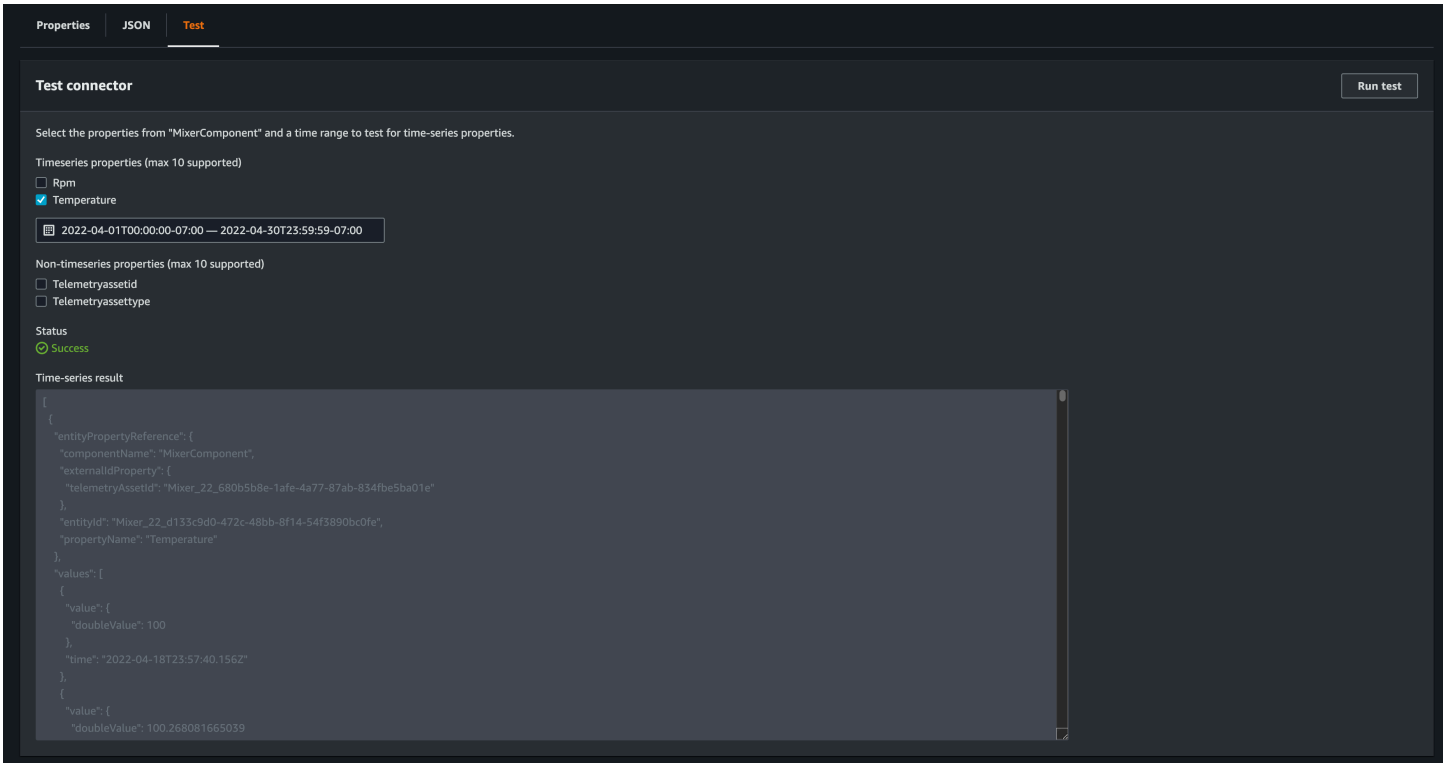
> **Note**
>
> If you follow the setup instructions in the GitHub sample, all AWS IoT TwinMaker resources are available automatically. You can check the component type definitions in the AWS IoT TwinMaker GitHub sample. Once the component type is used by any components, the property definitions and functions of the component type cannot be updated.

Integration testing

We recommend having an integrated test with AWS IoT TwinMaker to verify the data plane query works end-to-end. You can perform that through GetPropertyValueHistory API or easily in AWS IoT TwinMaker console.



In the AWS IoT TwinMaker console, go to **component details** and then under the **Test**, you'll see all the properties in the component are listed there. The **Test** area of the console allows you to test time-series properties as well as non-time-series properties. For time-series properties you can also use the GetPropertyValueHistory API and for non-time-series properties use GetPropertyValue API. If your Lambda connector supports multiple property query, you can choose more than one property.



What's next

You can now set up an AWS IoT TwinMaker Grafana dashboard to visualize metrics. You can also explore other data connector samples in the AWS IoT TwinMaker samples GitHub repository to see if they fit your use case.

AWS IoT TwinMaker cookie factory example time-series connector

The complete code of the cookie factory Lambda function is available on GitHub. Though you can still update the implementation after you link the connector to the component type, we strongly recommend you verify the Lambda connector before integrating with AWS IoT TwinMaker. You can test your Lambda function in the Lambda console or locally in the AWS CDK. For more information on testing your Lambda functions, see Testing Lambda functions, and Locally testing AWS CDK applications.

Example cookie factory component types

In a component type, we define common properties that are shared across components. For the cookie factory example, physical components of the same type share the same measurements, so we can define the measurements schema in the component type. As an example, the mixer type is defined in the following example.

```
{
    "componentTypeId": "com.example.cookiefactory.mixer"
    "propertyDefinitions": {
        "RPM": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isRequiredInEntity": false,
            "isExternalId": false,
            "isStoredExternally": true
        },
        "Temperature": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isRequiredInEntity": false,
            "isExternalId": false,
            "isStoredExternally": true
        }
    }
}
```

For example, a physical component might have measurements in a Timestream database, maintenance records in an SQL database, or alarm data in alarm systems. Creating multiple components and associating them with an entity links different data sources to the entity and populates the entity-component graph. In this context, each component needs an `telemetryId` property to identify the unique key of the component in the corresponding data source. Specifying the `telemetryId` property has two benefits: the property can be used in the data connector as a filter condition to only query values of the given component and, if you include the `telemetryId` property value in the data plane API response, then the client side takes the ID and can perform a reverse lookup if necessary.

If you add the `TelemetryId` to the component type as an external id, it identifies the component in the `TimeStream` table.

```
{
    "componentTypeId": "com.example.cookiefactory.mixer"
    "propertyDefinitions": {
        "telemetryId": {
            "dataType": { "type": "STRING" },
            "isTimeSeries": false,
            "isRequiredInEntity": true,
            "isExternalId": true,
```

```
            "isStoredExternally": false
        },
        "RPM": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isRequiredInEntity": false,
            "isExternalId": false,
            "isStoredExternally": true
        },
        "Temperature": {
            "dataType": { "type": "DOUBLE" },
            "isTimeSeries": true,
            "isRequiredInEntity": false,
            "isExternalId": false,
            "isStoredExternally": true
        }
    }
}
```

Similarly we have the component type for the `WaterTank`, as shown in the following JSON example.

```
{
  "componentTypeId": "com.example.cookiefactory.watertank",
  "propertyDefinitions": {
    "flowRate1": {
      "dataType": { "type": "DOUBLE" },
      "isTimeSeries": true,
      "isRequiredInEntity": false,
      "isExternalId": false,
      "isStoredExternally": true
    },
    "flowrate2": {
      "dataType": { "type": "DOUBLE" },
      "isTimeSeries": true,
      "isRequiredInEntity": false,
      "isExternalId": false,
      "isStoredExternally": true
    },
    "tankVolume1": {
      "dataType": { "type": "DOUBLE" },
      "isTimeSeries": true,
      "isRequiredInEntity": false,
```

```
      "isExternalId": false,
      "isStoredExternally": true
    },
    "tankVolume2": {
      "dataType": { "type": "DOUBLE" },
      "isTimeSeries": true,
      "isRequiredInEntity": false,
      "isExternalId": false,
      "isStoredExternally": true
    },
    "telemetryId": {
      "dataType": { "type": "STRING" },
      "isTimeSeries": false,
      "isRequiredInEntity": true,
      "isExternalId": true,
      "isStoredExternally": false
    }
  }
}
```

The `TelemetryType` is an optional property in the component type if it's aimed at querying property values in the entity scope. For an example, see the defined component types in the AWS IoT TwinMaker samples GitHub repository. There are alarm types also embedded into the same table, so the `TelemetryType` is defined and you extract common properties like the `TelemetryId` and `TelemetryType` to a parent component type for other child types to share.

Example Lambda

The Lambda connector needs to access the data source and generate the query statement based on the input and forward it to the data source. An example request sent to the Lambda is shown in the following JSON example.

```
{
    'workspaceId': 'CookieFactory',
    'selectedProperties': ['Temperature'],
    'startDateTime': 1648796400,
    'startTime': '2022-04-01T07:00:00.000Z',
    'endDateTime': 1650610799,
    'endTime': '2022-04-22T06:59:59.000Z',
    'properties': {
        'telemetryId': {
            'definition': {
```

```
                'dataType': { 'type': 'STRING' },
                'isTimeSeries': False,
                'isRequiredInEntity': True,
                'isExternalId': True,
                'isStoredExternally': False,
                'isImported': False,
                'isFinal': False,
                'isInherited': True,
            },
            'value': {
                'stringValue': 'Mixer_22_680b5b8e-1afe-4a77-87ab-834fbe5ba01e'
            }
        }
        'Temperature': {
            'definition': {
                'dataType': { 'type': 'DOUBLE' },
                'isTimeSeries': True,
                'isRequiredInEntity': False,
                'isExternalId': False,
                'isStoredExternally': True,
                'isImported': False,
                'isFinal': False,
                'isInherited': False
            }
        }
        'RPM': {
            'definition': {
                'dataType': { 'type': 'DOUBLE' },
                'isTimeSeries': True,
                'isRequiredInEntity': False,
                'isExternalId': False,
                'isStoredExternally': True,
                'isImported': False,
                'isFinal':False,
                'isInherited': False
            }
        },
    'entityId': 'Mixer_22_d133c9d0-472c-48bb-8f14-54f3890bc0fe',
    'componentName': 'MixerComponent',
    'maxResults': 100,
    'orderByTime': 'ASCENDING'
}
```

The goal of the Lambda function is to query historical measurement data for a given entity. AWS IoT TwinMaker provides a component-properties map, and you should specify an instantiated value for the component ID. For example, to handle the component type-level query (which is common for alarm use cases) and return the alarm status of all components in the workspace, then the properties map has component type properties definitions.

For the most straightforward case, as in the preceding request, we want a series of temperature samples during the given time window for the given component, in ascending time order. The query statement can be summarized as the following:

```
...
SELECT measure_name, time, measure_value::double
    FROM {database_name}.{table_name}
    WHERE time < from_iso8601_timestamp('{request.start_time}')
    AND time >= from_iso8601_timestamp('{request.end_time}')
    AND TelemetryId = '{telemetry_id}'
    AND measure_name = '{selected_property}'
    ORDER BY time {request.orderByTime}
...
```

Creating and editing AWS IoT TwinMaker scenes

Scenes are three-dimensional visualizations of your digital twin. They're the primary way for you to edit your digital twin. Learn how to add alarms, time series data, color overlays, tags, and visual rules to your scene to align your digital twin visualizations with your real-world use case.

This section covers the following topics:

Before you create your first scene

Scenes rely on resources to represent your digital twin. These resources are made up of 3D models, data, or texture files. The size and complexity of your resources, elements in the scene such as lighting, and your computer hardware, impact the performance of AWS IoT TwinMaker scenes. Use the information in this topic to reduce lag, loading times, and improve the frame rate of your scenes.

Optimize your resources before importing them into AWS IoT TwinMaker

You can use AWS IoT TwinMaker to interact with your digital twin in real time. For the best experience with your scenes, we recommend optimizing your resources for use in a real-time environment.

Your 3D models can have a significant impact on performance. Complex model geometry and meshes can reduce performance. For example, industrial CAD models have a high level of detail. We recommend compressing these model's meshes and reducing their polygon count before using them in AWS IoT TwinMaker scenes. If you're creating new 3D models for AWS IoT TwinMaker,

you should establish a level of detail and maintain it across all your models. Remove details from models that don't affect the visualization or interpretation of your use case..

To compress models and reduce the file size, use open source mesh compression tools, such as DRACO 3D data compression.

Unoptimized textures can also impact performance. If you don't require any transparency in your textures, considering choosing the PEG image format over the PNG format. You can compress your texture files by using open source texture compression tools, such as Basis Universal texture compression.

Best practices for performance in AWS IoT TwinMaker

For the best performance with AWS IoT TwinMaker, note the following limitations and best practices.

- AWS IoT TwinMaker scene rendering performance is hardware dependent. Performance varies across different computer hardware configurations.
- We recommend a total polygon count of under 1 million across all your objects in your AWS IoT TwinMaker.
- We recommend a total of 200 objects per scene. Increasing the number of objects in a scene beyond 200 can decrease your scene frame rate.
- We recommend the that total size of all unique 3D assets in your scene does not exceed 100 megabytes. Otherwise, you may encounter slow loading times or degraded performance depending on your browser and hardware.
- Scenes have ambient lighting by default. You can add extra lights into a scene to bring certain objects into focus, or cast shadows on objects. We recommend using one light per scene. Use lights where needed, and avoid replicating real-world lights within a scene.

Learn more

Use these resources to learn more about optimization techniques that you can use to improve performance in your scenes.

- How to convert and compress OBJ models to GLTF for use with AWS IoT TwinMaker
- Optimize your 3D models for web content
- Optimizing scenes for better WebGL performance

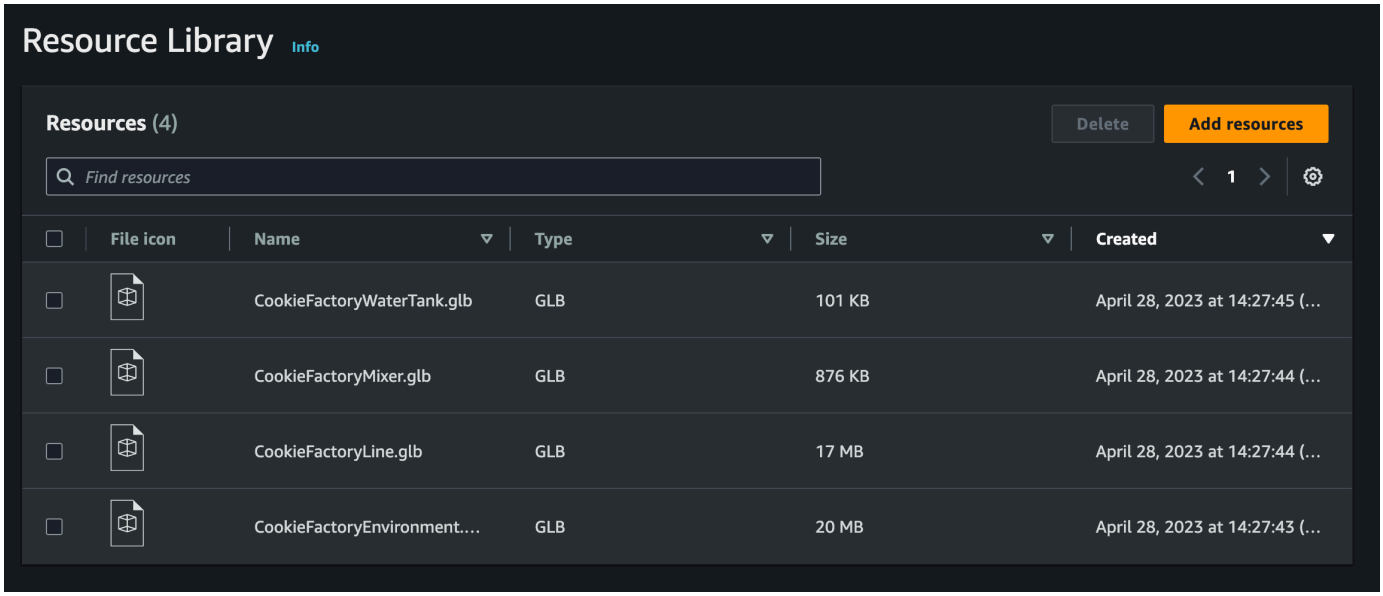
Upload resources to the AWS IoT TwinMaker Resource Library

You can use the Resource Library to control and manage any resource you want to place into scenes for your digital twin application. To make AWS IoT TwinMaker aware of the resources, upload them using the Resource Library console page.

Upload files to the Resource Library using the console

Follow these steps to add files to the Resource Library using the AWS IoT TwinMaker console.

1. In the left navigation menu, under **Workspaces**, select **Resource Library**.
2. Select **Add resources** and choose the files you want to upload.



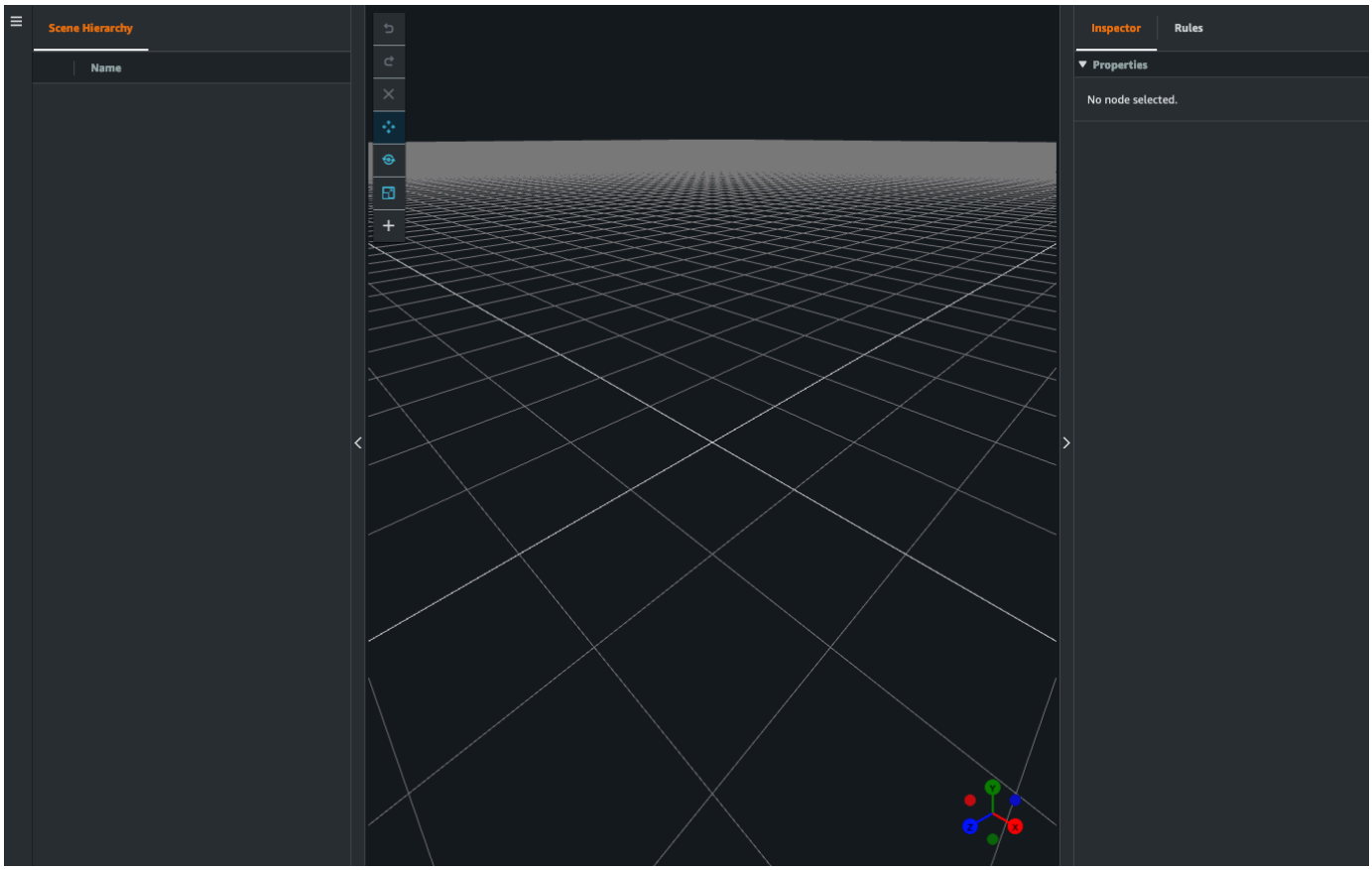
Create your scenes

In this section, you'll set up a scene so that you can edit your digital twin. You can import a 3D model that was uploaded to the resource library, then add widgets and bind property data to objects to complete your digital twin. Scene objects can include an entire building or space, or individual pieces of equipment positioned in their physical location.

> **Note**
>
> Before you create a scene, you must create a workspace.

Use the following procedure to create your scene in AWS IoT TwinMaker.

1. To open the scene pane, in the left navigation of your workspace, choose **Scenes**.
2. Choose **Create scene**. The new scene creation pane opens.
3. In the scene creation pane, enter a name and description for your new scene. If you have a standard or tiered bundle pricing plan, you can select your scene type. It is recommended to use a dynamic scene.
4. When you're ready to create the scene, choose **Create scene**. The new scene opens and is ready for you to work with it.

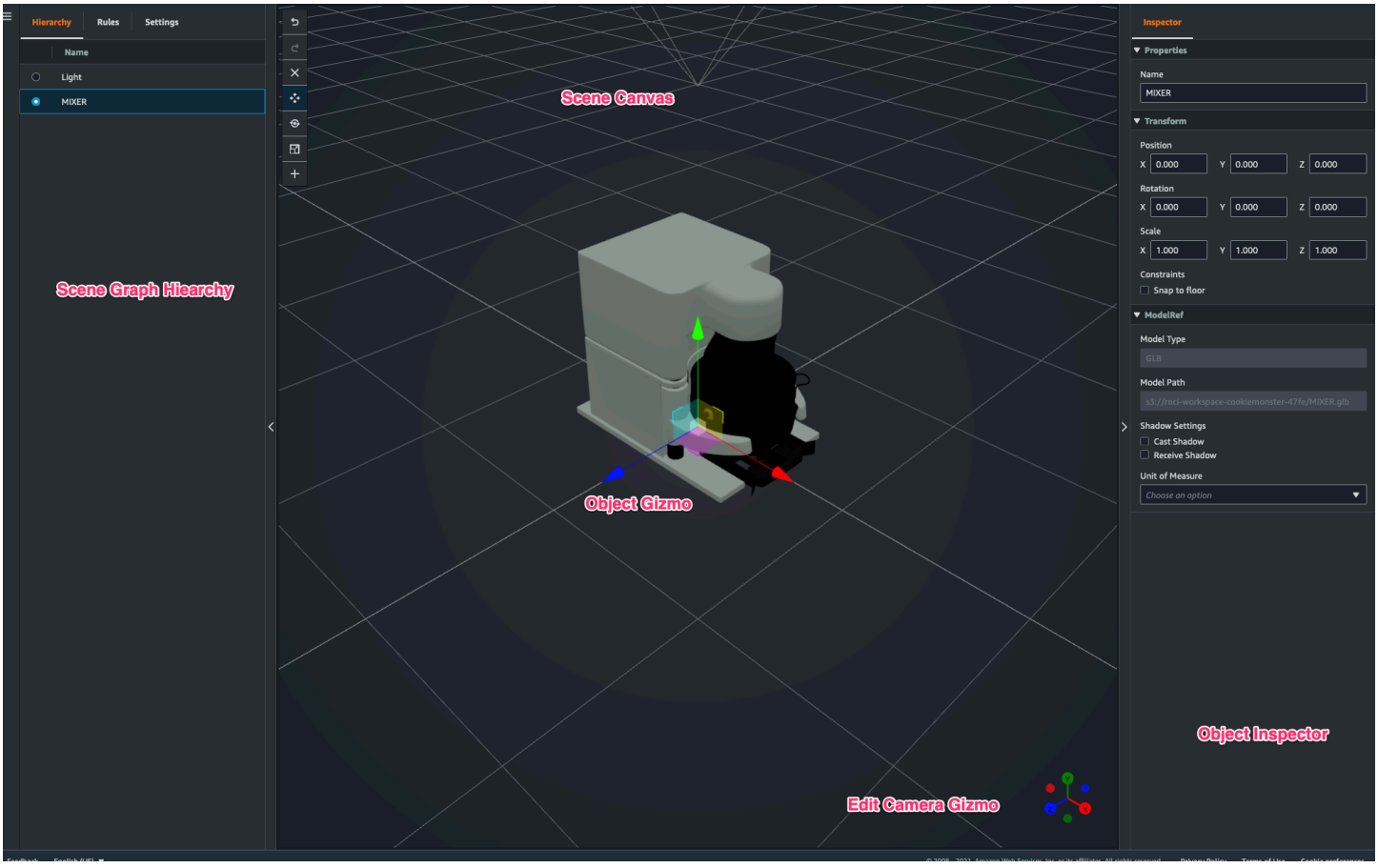


Use 3D navigation in your AWS IoT TwinMaker in scenes

The AWS IoT TwinMaker scene has a set of navigation controls that you can use to navigate efficiently through your scene's 3D space. To interact with the 3D space and objects represented by your scene, you use the following widgets and menu options.

- **Inspector**: Use the Inspector window to view and edit properties and settings of a selected entity or component in your hierarchy.

- **Scene Canvas**: The Scene Canvas is the 3D space where you can position and orient any 3D resources you want to use.
- **Scene Graph Hierarchy**: You can use this panel to see all of the entities present in your scene. It appears on the left side of the window.
- **Object gizmo**: Use this gizmo to move objects around the canvas. It appears at the center of a selected 3D object in the Scene Canvas.
- **Edit Camera gizmo**: Use the Edit Camera gizmo to quickly view the scene view camera's current orientation and modify the viewing angle. You can find this gizmo in the lower-right corner of the scene view.
- **Zoom controls**: To navigate on the Scene Canvas, use right click and drag in the direction you want to move. To rotate , left click and drag to rotate. To zoom, use the scroll wheel on your mouse, or pinch and move your fingers apart on the track pad of your laptop.



The scene buttons on the hierarchy pane have the following functions listed, in order of the buttons' layout:

- **Undo**: Undo your last change in the scene.
- **Redo**: Redo your last change in the scene.
- **Plus (+)**: Use this button to gain access to the following actions: **Add empty node**, **Add 3D model**, **Add tag**, **Add light**, and **Add model shader**.
- **Change navigation method**: Gain access to the scene camera navigation options, **Orbit** and **Pan**.
- **Trashcan (delete)**: Use this button to delete a selected object in your scene.
- **Object manipulation tools**: Use this button to translate, rotate, and scale the selected object.

Add fixed cameras to entities

You can attach fixed camera views to your entities within your AWS IoT TwinMaker scenes. These cameras provide a fixed perspective on a 3d model, allowing you to quickly and easily shift your perspective in a scene to a targeted entity.

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. In the scene hierarchy menu, select the entity you want to attach the camera to.
3. Press the **+** button, and from the drop down options select **Add camera from current view**. To apply a camera with the current perspective to the entity.
4. In the inspector, you can configure your camera and adjust the following settings:

    - A camera **Name**
    - The camera **position** and **rotation**
    - The camera **focal length**
    - The **zoom level**
    - **Near** and **Far** clipping planes
5. To access your camera after you have placed it. Select the entity you added the camera to in the hierarchy. Look for the camera name listed under the entity.
6. Once you select the placed camera from your entity, the scenes camera view will snap to the set perspective of the placed camera.

Scene enhanced editing

AWS IoT TwinMaker scenes feature a set of tools for enhanced and editing and manipulation of resources present in your scene.

The following topics teach you how to used the enhanced editing features in your AWS IoT TwinMaker scenes.

Targeted placement of scene objects

AWS IoT TwinMaker allows you to precisely place and add objects into your scene. This enhanced editing feature gives you greater control of where you're placing tags, entities lights and models in your scene.

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. Press the **+** button, and from the drop down options select one of the options. This could be a model, a light, a tag, or anything from the **+** menu.

   When you move your cursor in the 3d space of your scene you should see a target around your cursor .
3. Use the target to precisely place elements in your scene.

Submodel selection

AWS IoT TwinMaker lets you select submodels of 3d models in scenes and apply standard properties to them, such as tags, lights, or rules.

3d model file formats contain metadata that can specify sub areas of the model as submodels within the larger model. For example a model could be a filtration system, individual parts of the system like tanks, pipes, or a motor are marked as submodels of the filtration's 3d model.

Supported 3D file formats in scenes: GLB, and GLTF.

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. If you have no models in your scene, make sure to add one by selecting the option from the **+** menu.
3. Select model listed in your scene hierarchy, once selected the hierarchy should display any submodels beneath the model.

> **Note**
>
> If you do not see any submodels listed then it is likely the model was not configured to have any submodels.

4. To toggle the visibility of a submodel, press the eye icon, located to the right of the submodel's name in the hierarchy.
5. To edit submodel data, such as its name or position, the scene inspector will open when a submodel is selected. Use the inspector menu to update or change submodel data.
6. To add tags, lights, rules, or other properties to submodels, press the **+**, while the submodel is selected in the hierarchy.

Edit entities in the scene hierarchy

AWS IoT TwinMaker scenes let you directly edit properties of entities within the hierarchy table. The following procedure shows you which actions you can perform on an entity through the hierarchy menu.

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. Open the scene hierarchy, and select a sub element of an entity you wish to manipulate.
3. Once the element is selected, press the **+** button, and from the drop down select one of the options:

   - **Add empty node**
   - **Add 3D model**
   - **Add light**
   - **Add camera from current view**
   - **Add tag**
   - **Add model shader**
   - **Add motion indicator**
4. After selecting one of the options from the drop down, the selection will be applied to the scene as child of the selected element from step 2.
5. You can reorder child elements and reparent elements, by selecting a child element and dragging in the hierarchy to a new parent.

Add annotations to entities

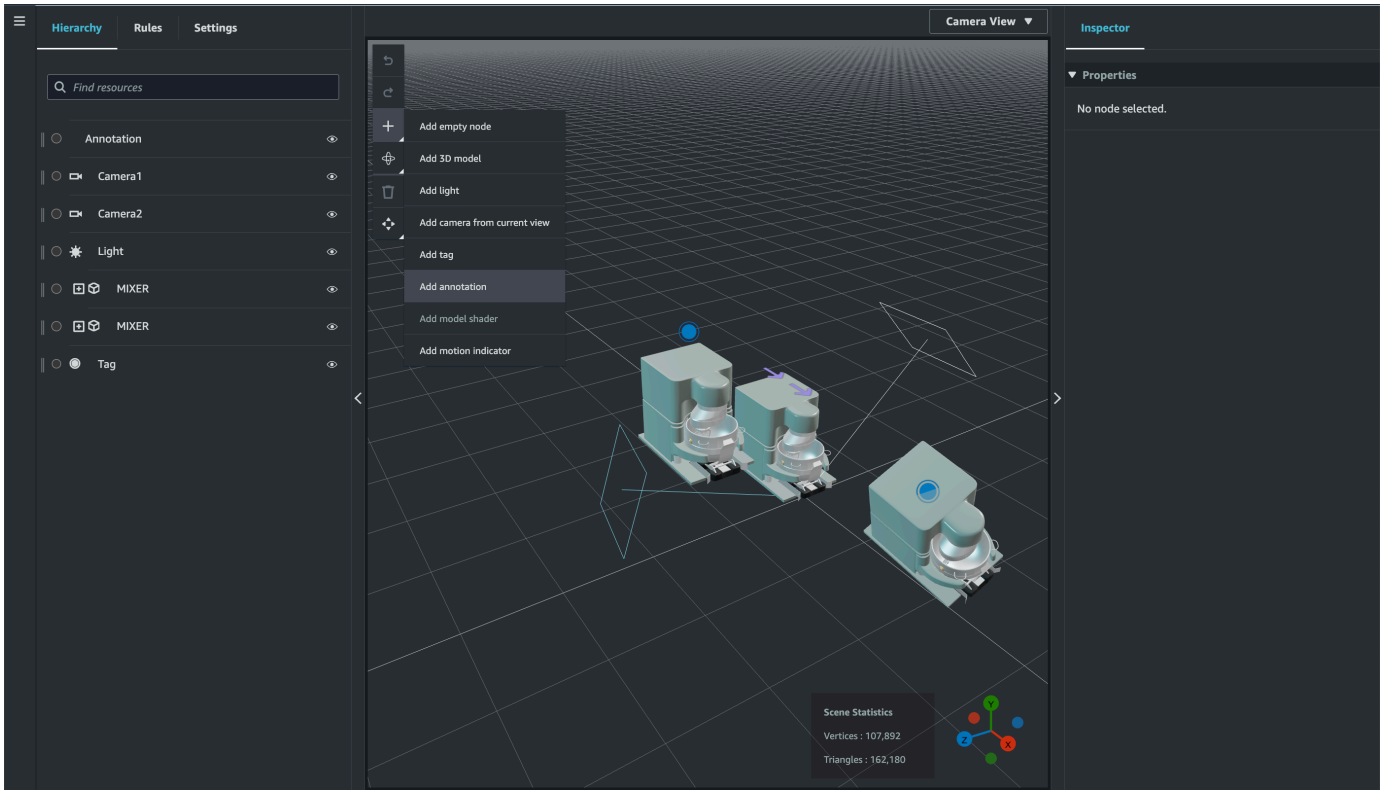
The AWS IoT TwinMaker scene composer lets you annotate any element in your scene hierarchy. The annotation is authored in markdown.

For more information on writing in Markdown, see the official documentation on markdown syntax, Basic Syntax.

> **Note**
>
> AWS IoT TwinMaker annotations and overlay Markdown syntax only and not HTML.

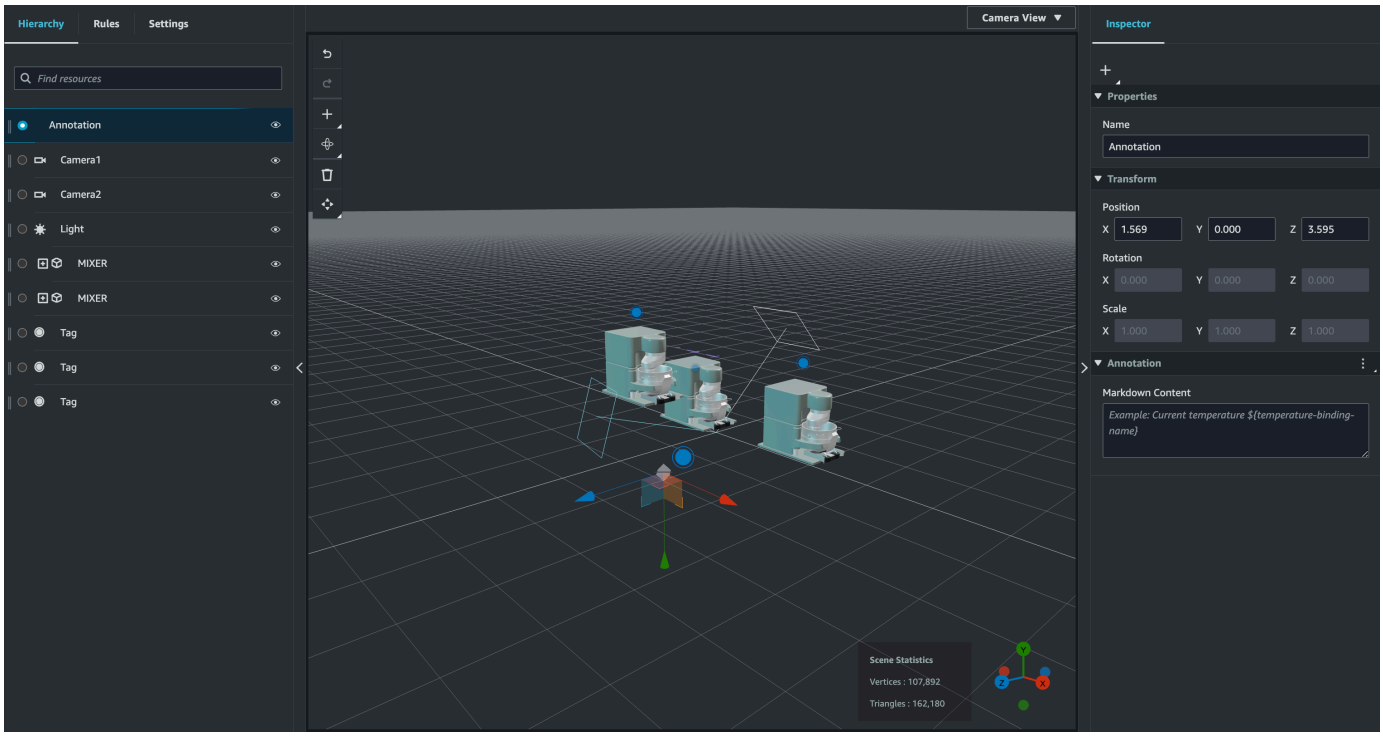
Add an annotation to an entity

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. Select an element from the scene hierarchy that you want to annotate. If no element in the hierarchy is selected, then you can add annotation to the root.
3. Press the plus **+** button and choose the **Add annotation** option.

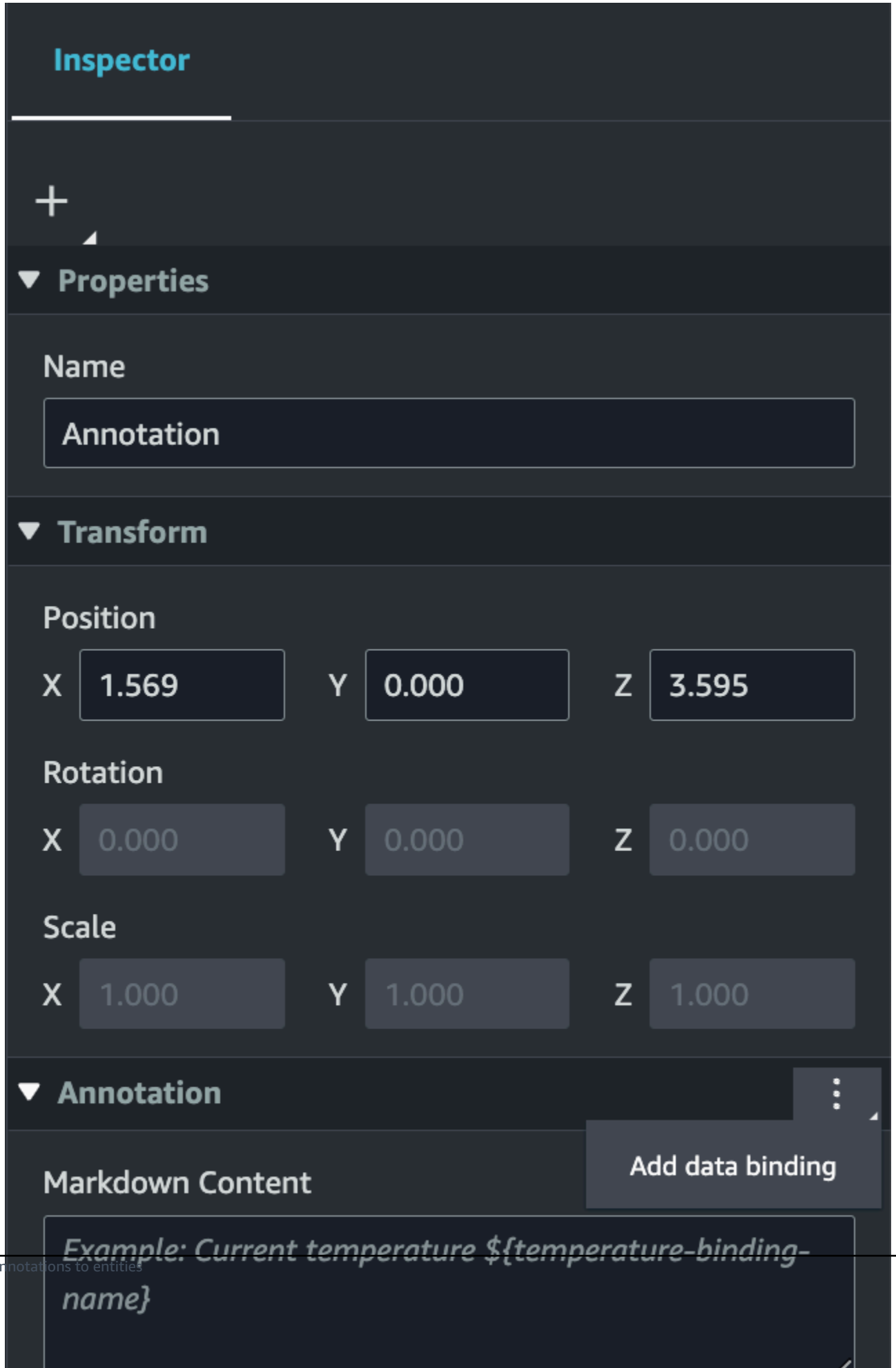


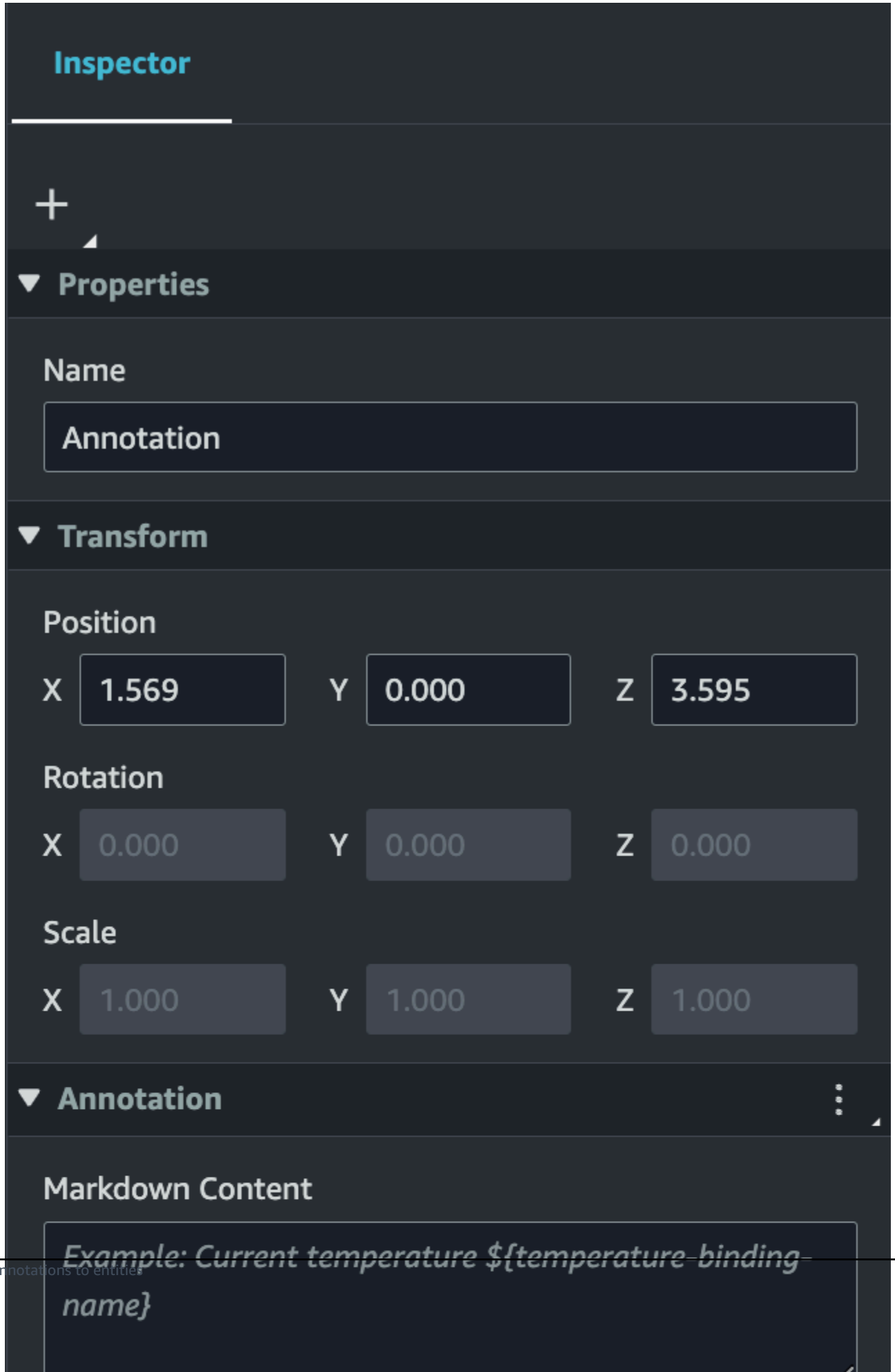
4. In the **Inspector** window on the left, scroll down to the **annotation** section. Using Markdown syntax, write the text you want your annotation to display.

For more information on writing in Markdown, see the official documentation on markdown syntax, Basic Syntax.



5. To bind your AWS IoT TwinMaker scene data to an annotation choose **Add data binding**, add the **Entity Id**, then select the **Component Name** and **Property Name** of the entity you wish to surface data from. You can update the binding name to use it as a Markdown variable, and surface the data in the annotation.

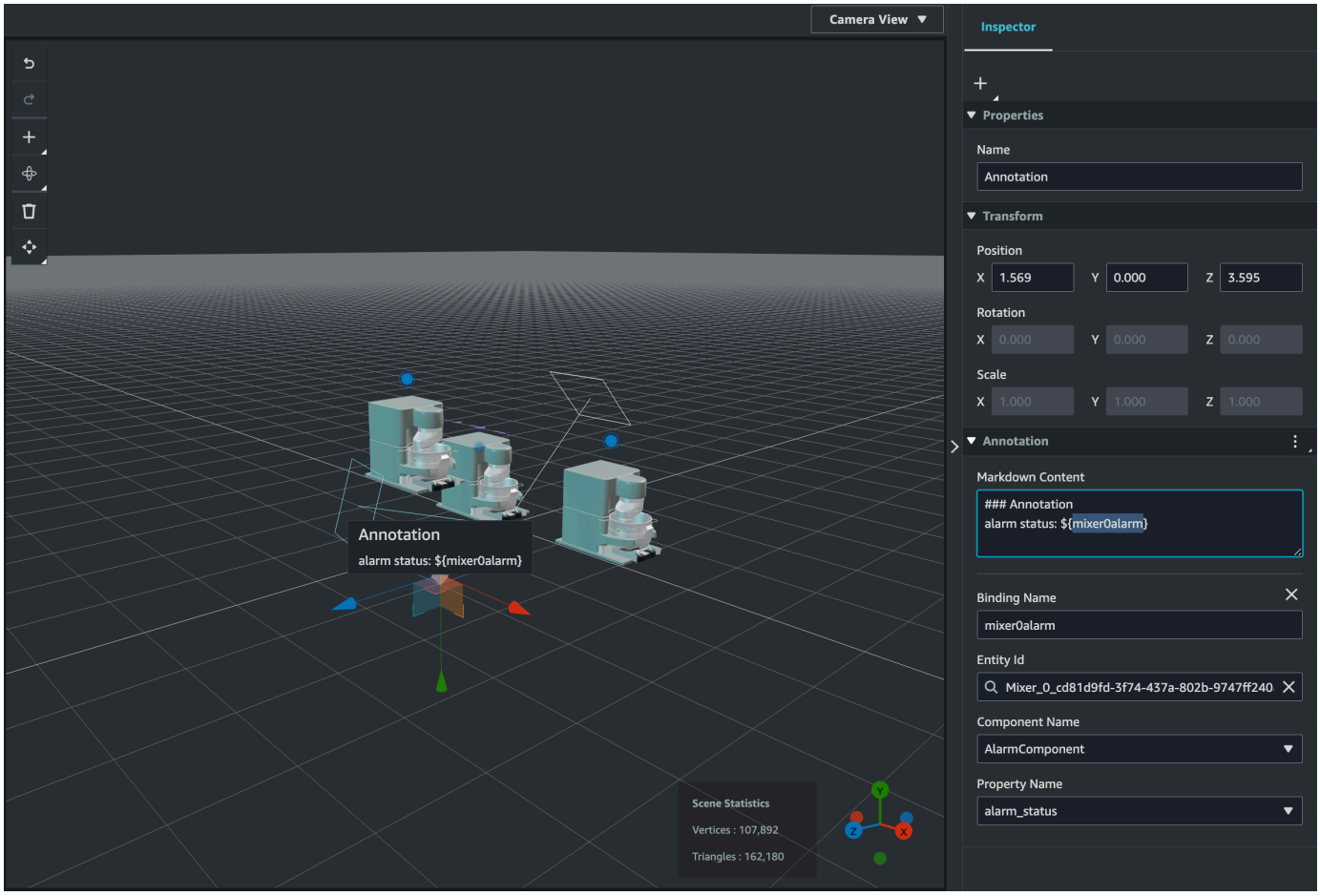

Inspector
Properties
Name
Annotation
Transform
Position
X 1.569
Y 0.000
Z 3.595
Rotation
X 0.000
Y 0.000
Z 0.000
Scale
X 1.000
Y 1.000
Z 1.000
Annotation
Add data binding
Markdown Content
Example: Current temperature ${temperature-binding-name}


Inspector
Properties
Name
Annotation
Transform
Position
X 1.569
Y 0.000
Z 3.595
Rotation
X 0.000
Y 0.000
Z 0.000
Scale
X 1.000
Y 1.000
Z 1.000
Annotation
Markdown Content
Example: Current temperature ${temperature-binding-name}

6. The **Binding Name** is used to represent the annotation's variable.

   Enter a **Binding Name** to surface the latest historical value of an entities time-series in the annotation through AWS IoT TwinMaker's variable syntax: `${variable-name}`

   As an example, this overlay displays the value of the `mixer0alarm`, in the annotation with the syntax `${mixer0alarm}`.



Add overlays to Tags

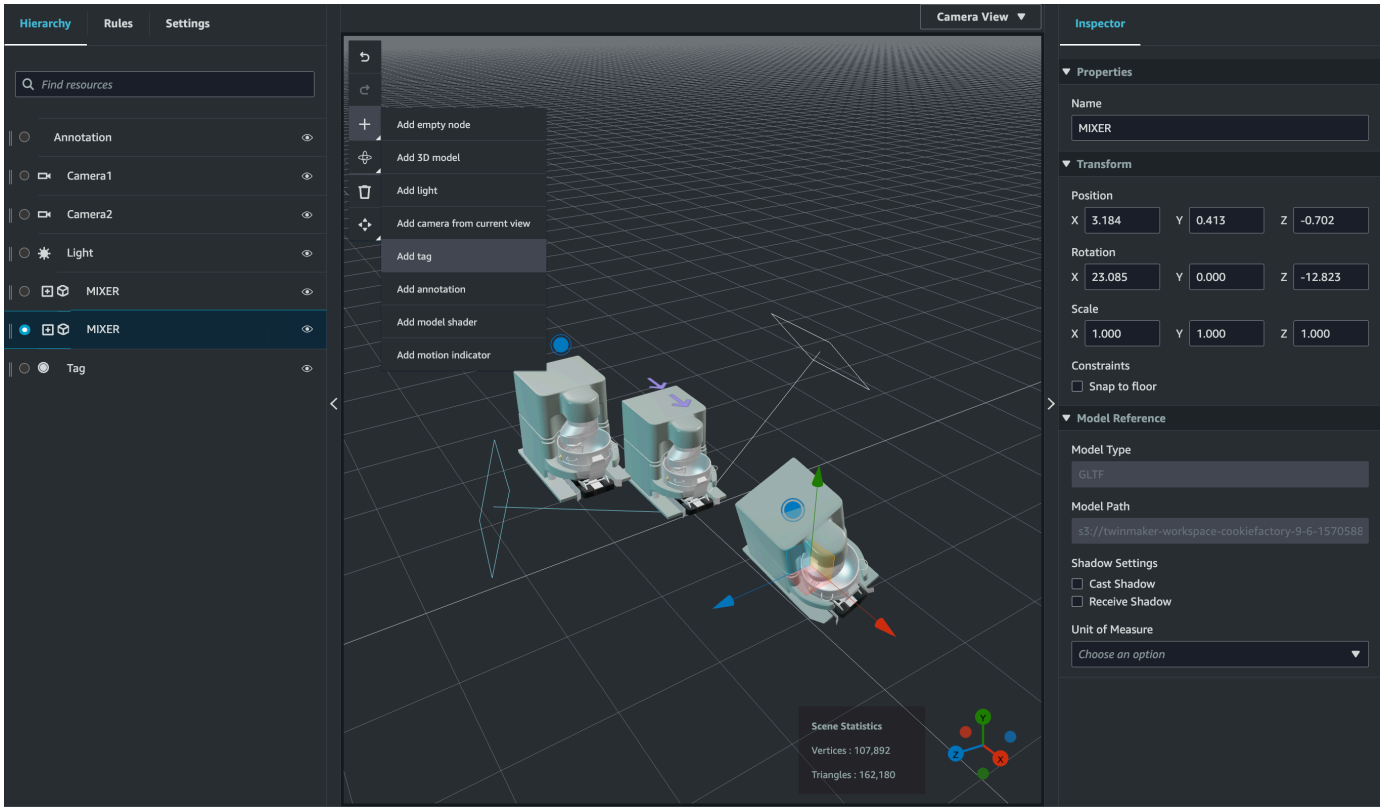
You can create overlays for your AWS IoT TwinMaker scenes. Scene overlays are associated with tags and can be used to surface critical data associated with your scene entities. The overlay is authored and rendered in Markdown.

For more information on writing in Markdown, see the official documentation on markdown syntax, Basic Syntax.

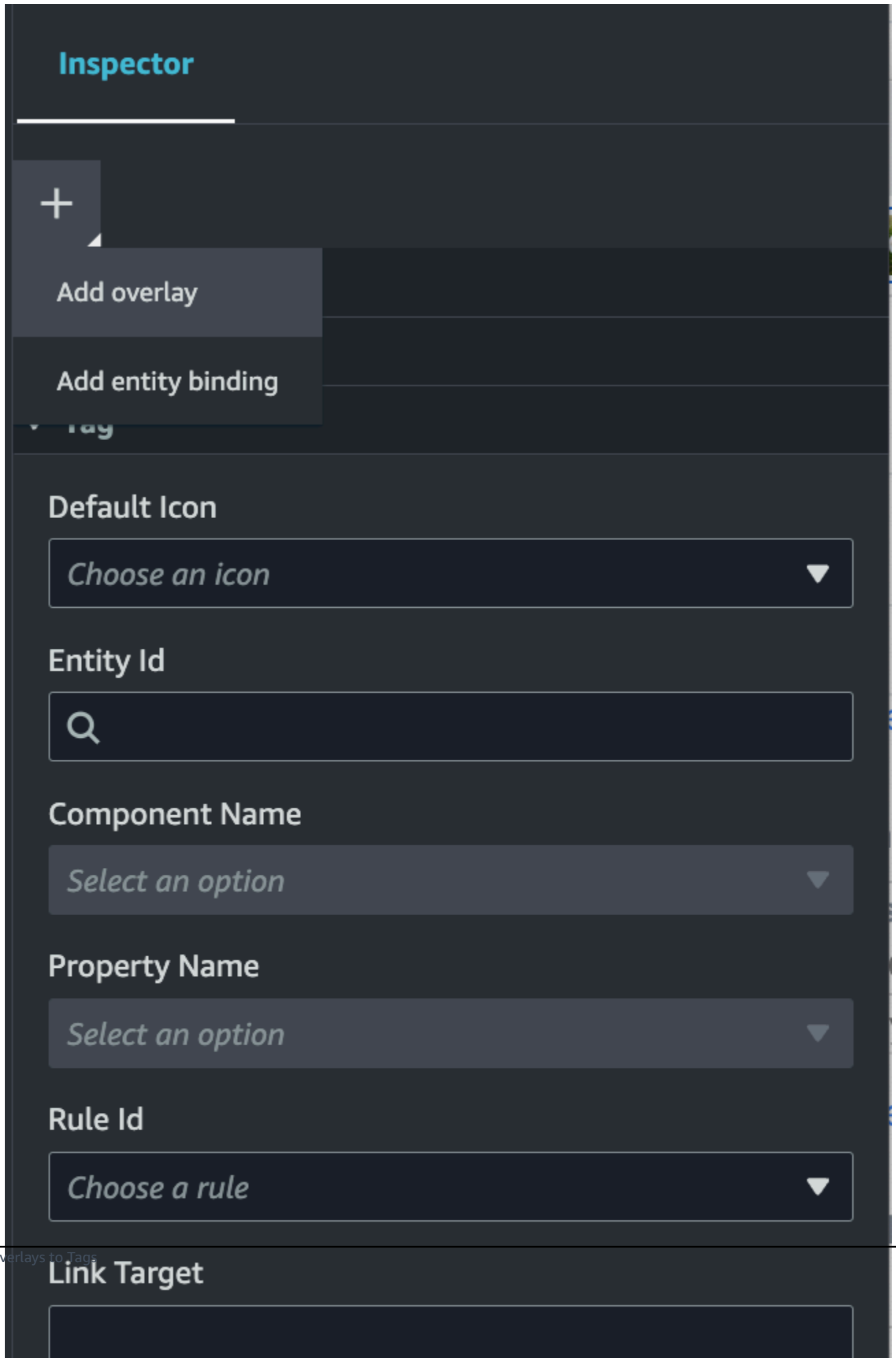
> **Note**
>
> By default, an **Overlay** is visible in a scene only when the tag associated with it is selected. You can toggle this in the scene **Settings** so that all **Overlays** are visible at once.

1. Navigate to your scene in the AWS IoT TwinMaker console.
2. The AWS IoT TwinMaker **overlay** is associated with a tag scene, you can update an existing tag or add a new one.

   Press the plus **+** button and choose the **Add tag** option.



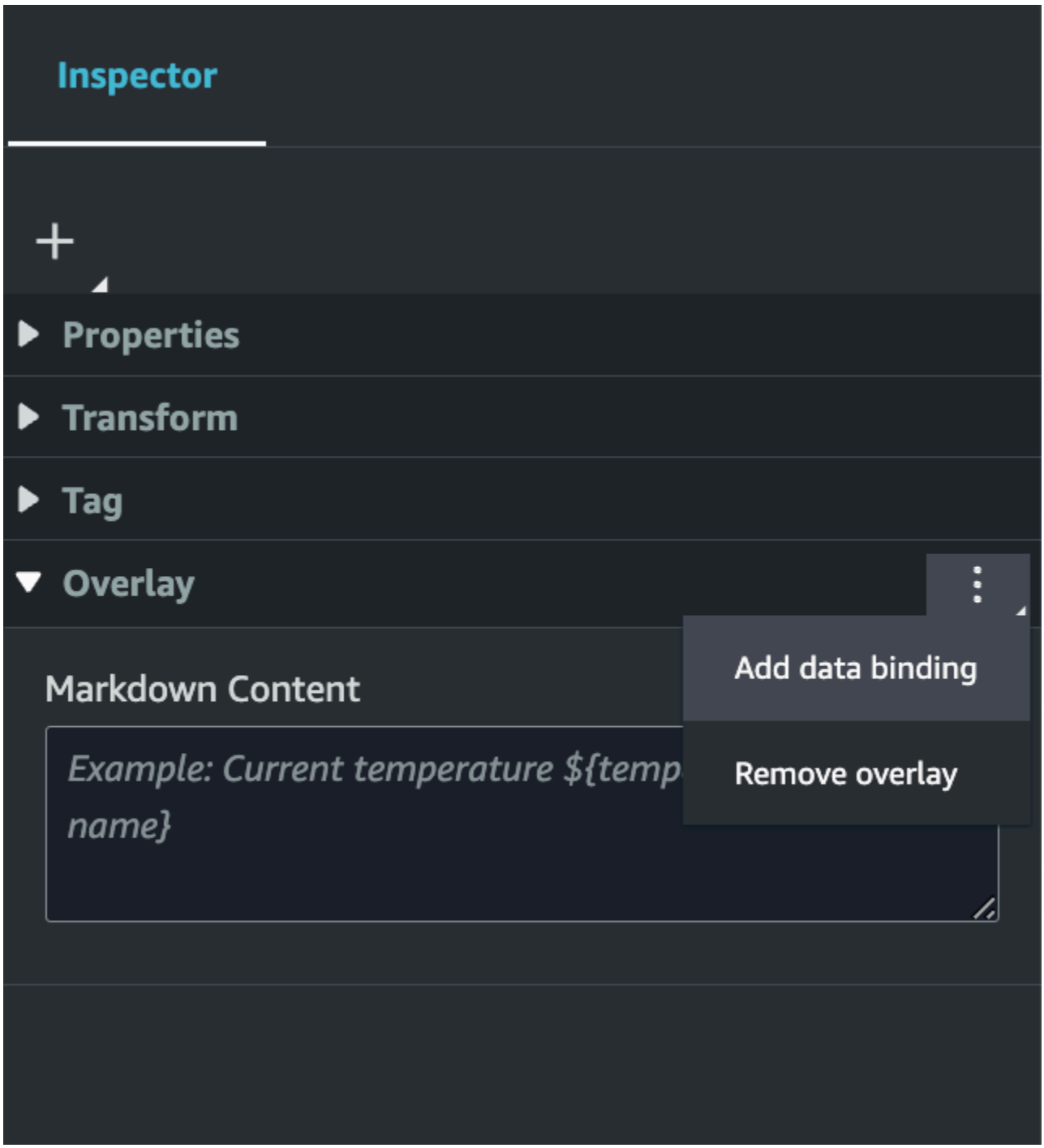
3. In the **Inspector** panel on the right, select the **+** (plus symbol) button then select **Add overlay**.


Inspector
Add overlay
Add entity binding
Default Icon
Choose an icon
Entity Id
Component Name
Select an option
Property Name
Select an option
Rule Id
Choose a rule
Link Target

4. In Markdown syntax, write the text you want your overlay to display.

   For more information on writing in Markdown, see the official documentation on markdown syntax, Basic Syntax.

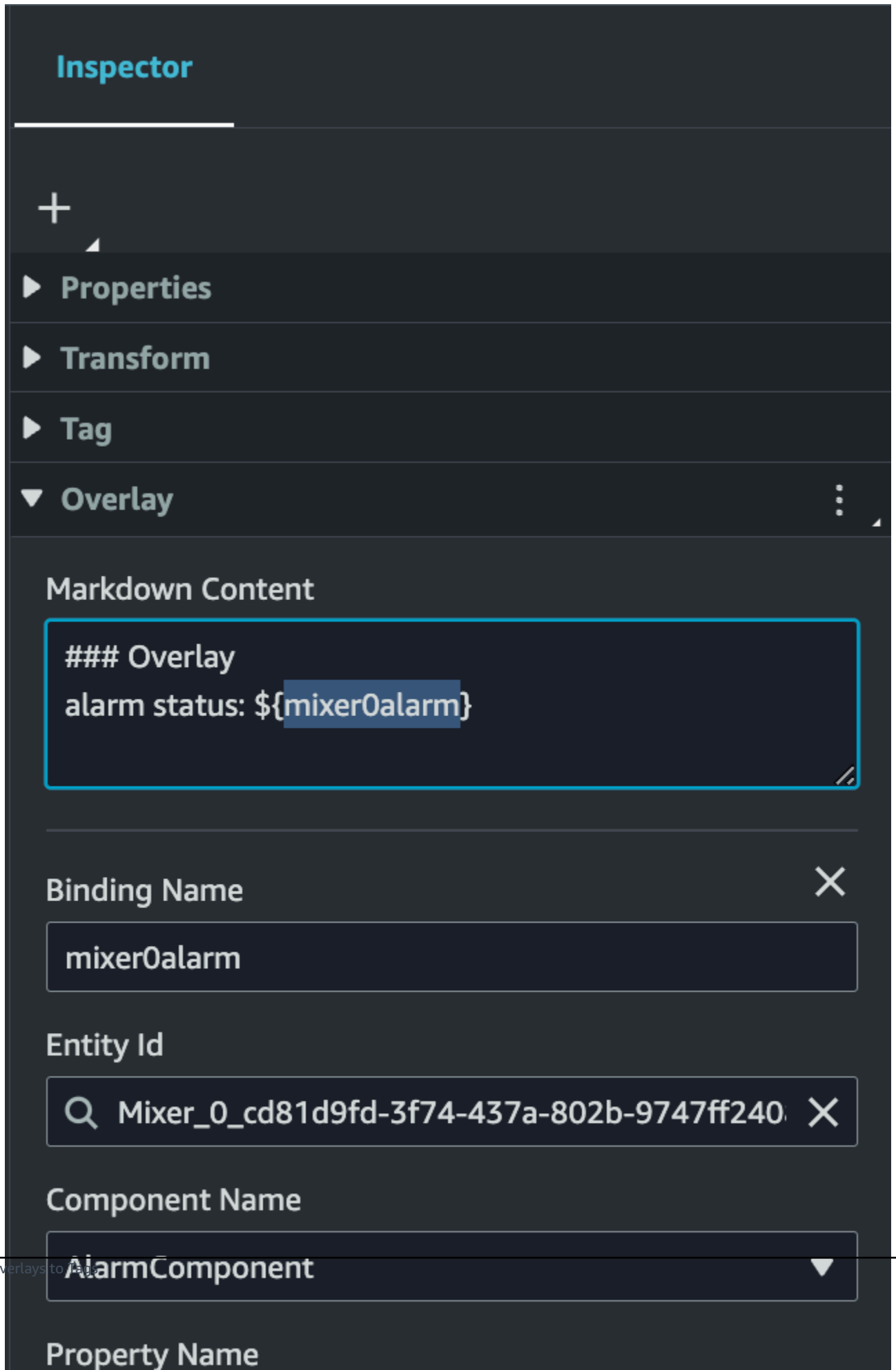
5. To bind your AWS IoT TwinMaker scene data to an overlay, select **Add data binding**.



Add the **Binding name** and **Entity Id**, then select the **Component Name** and **Property Name** of the entity you wish to surface data from.

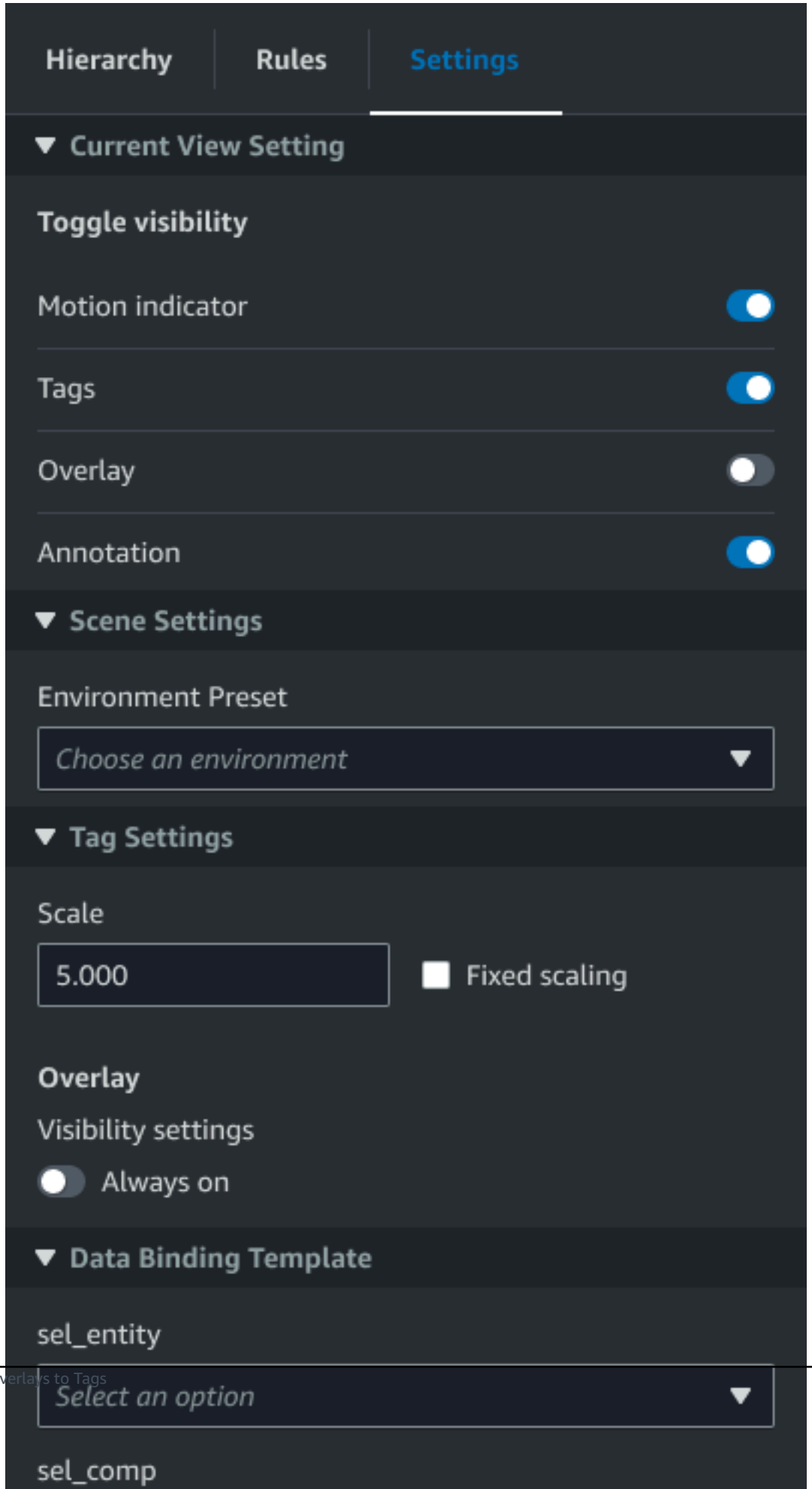
6. You can surface the latest historical value of an entities time-series data in the overlay through AWS IoT TwinMaker's variable syntax: `${variable-name}`.

   As an example, this overlay displays the value of the `mixer0alarm`, in the overlay with the syntax `${mixer0alarm}`.


Inspector
Properties
Transform
Tag
Overlay
Markdown Content
Overlay
alarm status: ${mixer0alarm}
Binding Name
mixer0alarm
Entity Id
Mixer_0_cd81d9fd-3f74-437a-802b-9747ff240
Component Name
AlarmComponent
Property Name

7. To enable **Overlay** visibility, open the **Settings** tab in the top left, and make sure the toggle for **Overlay** is switched on so that all **Overlays** are visible at once.

> **Note**
>
> By default, an **Overlay** is visible in a scene only when the tag associated with it is selected.


Hierarchy
Rules
Settings
Current View Setting
Toggle visibility
Motion indicator
Tags
Overlay
Annotation
Scene Settings
Environment Preset
Choose an environment
Tag Settings
Scale
5.000
Fixed scaling
Overlay
Visibility settings
Always on
Data Binding Template
sel_entity
Select an option
sel_comp

Edit your scenes

After you've created a scene, you can add entities, components, and configure augmented widgets into your scene. Use entity components and widgets to model your digital twin and provide functionality that matches your use case.

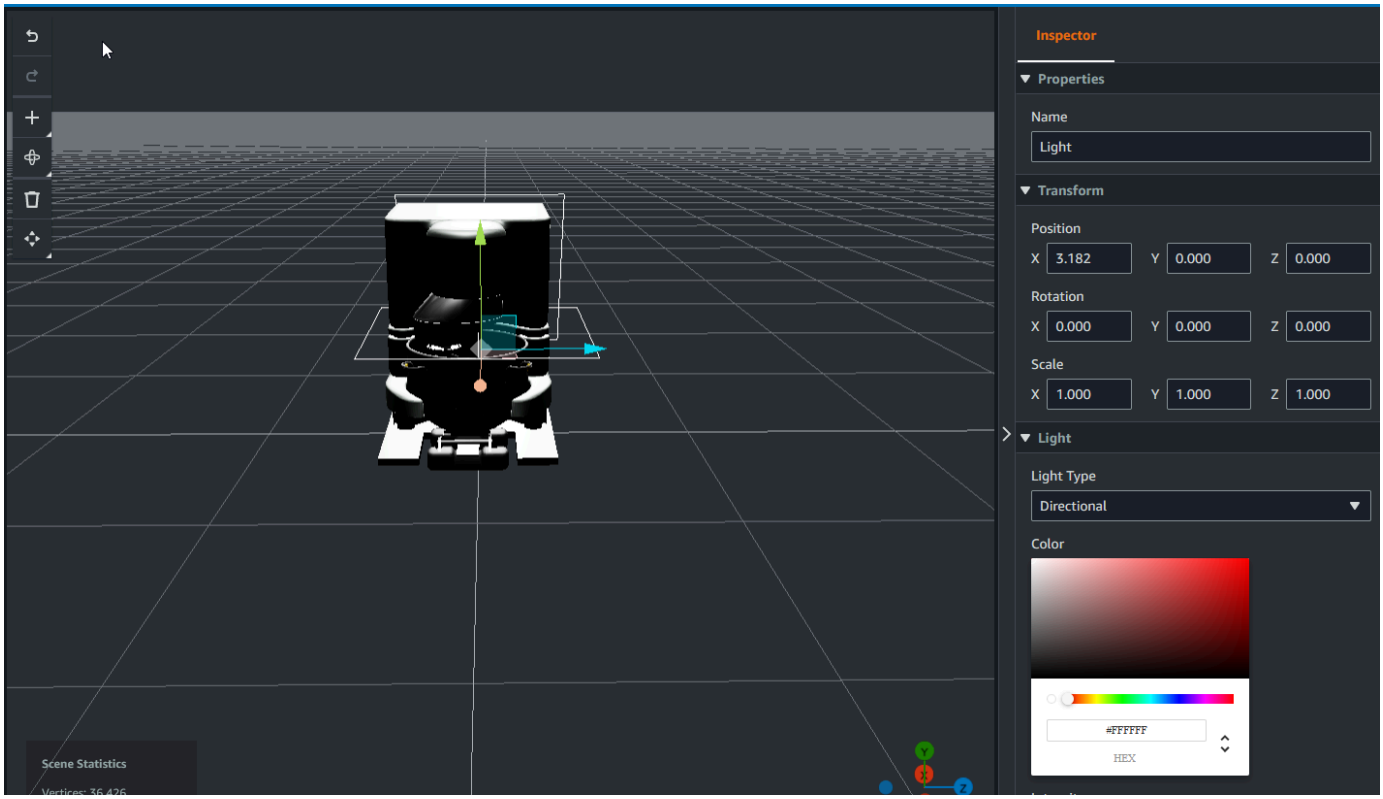
Topics

Add models to your scenes

To add models to your scene, use the following procedure.

> **Note**
>
> To add models in your scene, you must first upload the models to the AWS IoT TwinMaker Resource Library. For more information, see Upload resources to the AWS IoT TwinMaker Resource Library.

1. On the scene composer page, choose the plus (**+**) sign, and then choose **Add 3D model**.
2. On the **Add resource from resource library** window, choose the **CookieFactorMixer.glb** file, and then choose **Add**. Scene composer opens.
3. **Optional**: Choose the plus (**+**) sign, and then choose **Add light**.
4. Choose each light option to see how they affect the scene.



> **Note**
>
> Scenes have default ambient lighting. To avoid frame rate loss, consider limiting the number of additional lights placed in your scene.

Add model shader augmented UI widgets to your scene

Model shader widgets can change the color of an object under conditions that you define. For example, you can create a color widget that changes the color of a cookie mixer in your scene based on the mixer's temperature data.

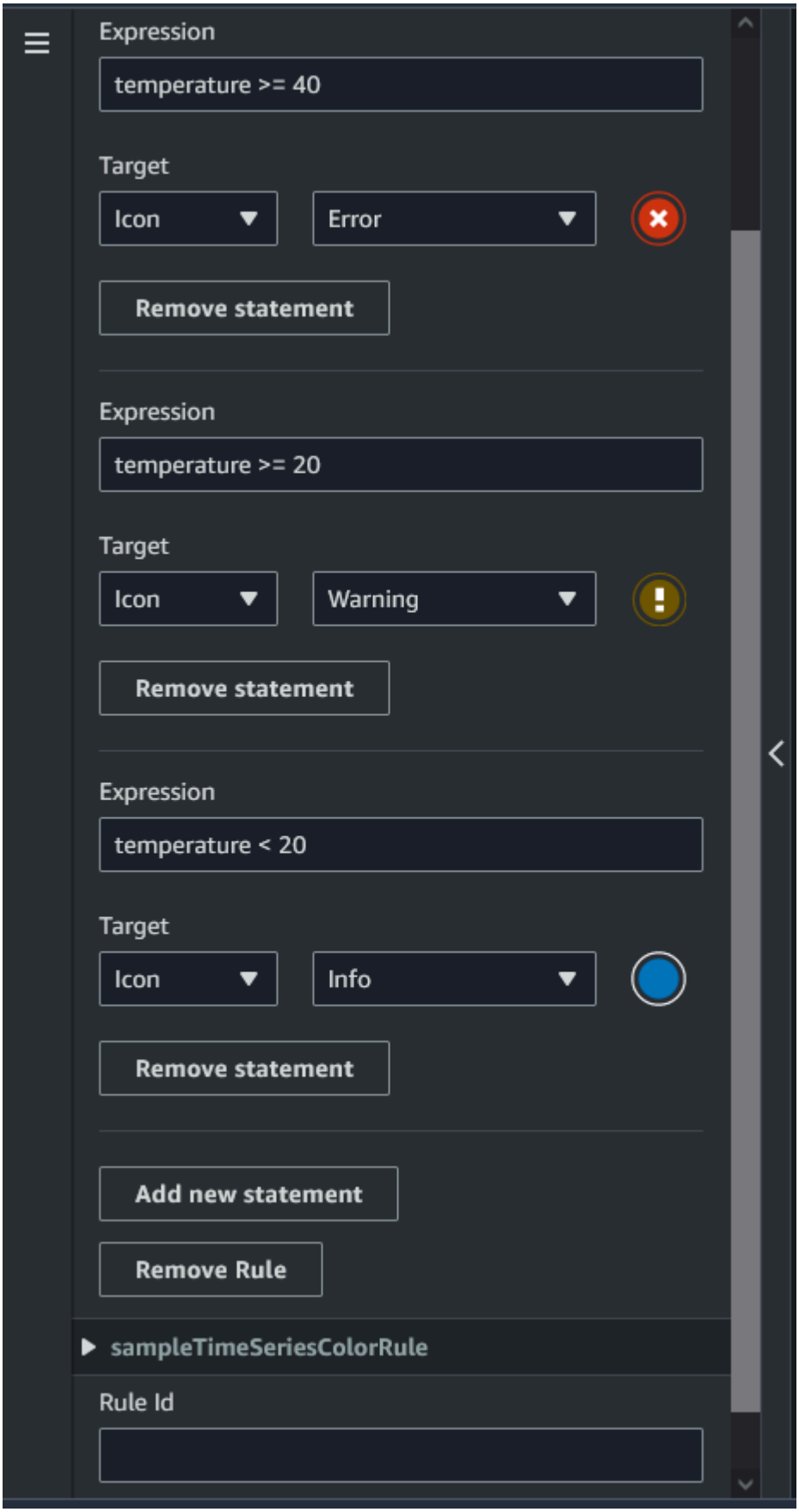
Use the following procedure to add model shader widgets to a selected object.

1. Select an object in the hierarchy that you want to add a widget to. Press the **+** button and then choose **Model Shader**.
2. To add a new visual rule group, first follow the instructions below to create the ColorRule, then in the Inspector panel for the object of the Rule ID, choose **ColorRule**.

3. Select the entityID, ComponentName, and PropertyName you want to bind the model shader to.

Create visual rules for your scenes

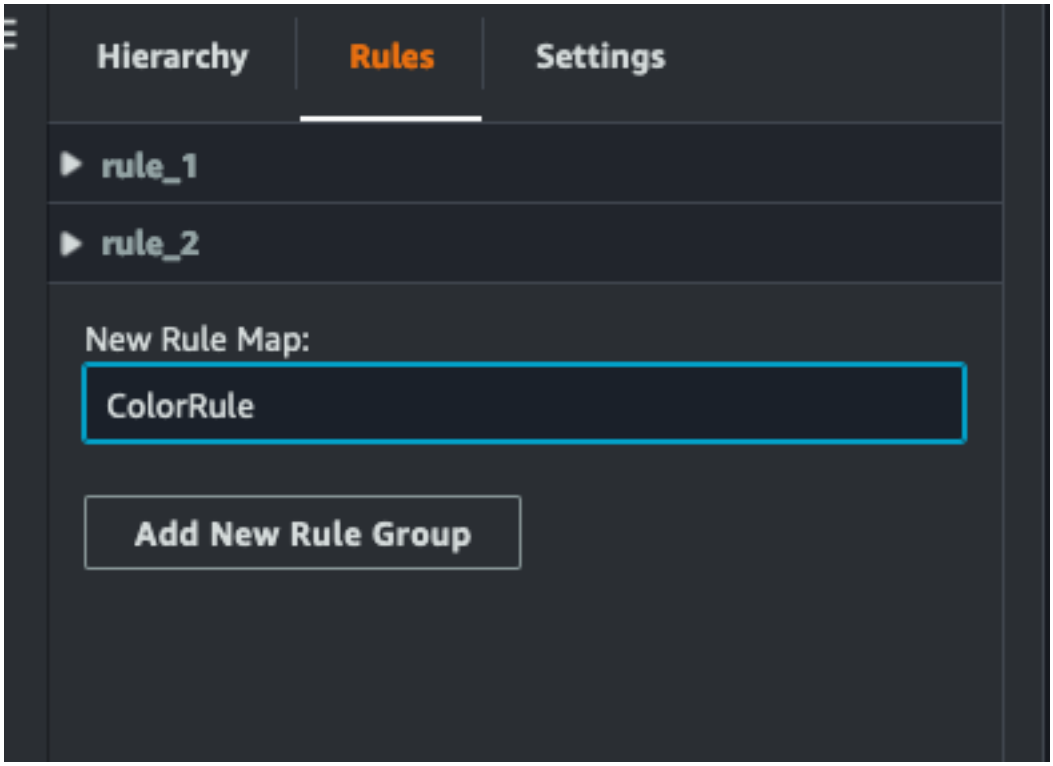
You can use visual rule maps to specify the data driven conditions that change the visual appearance of an augmented UI widget, such as a tag or a model shader. There are sample rules provided, but you can also create your own. The following example shows a visual rule.


Expression
temperature >= 40
Target
Icon
Error
Remove statement
Expression
temperature >= 20
Target
Icon
Warning
Remove statement
Expression
temperature < 20
Target
Icon
Info
Remove statement
Add new statement
Remove Rule
sampleTimeSeriesColorRule
Rule Id

The image above shows a rule for when a previously defined data property with ID 'temperature' is checked against a certain value. For example, if the 'temperature' is greater than or equal to 40, the state will change the appearance of the tag to a red circle. The **target**, when chosen in the Grafana dashboard, populates a detail panel that is configured to use the same data source.

The following procedure shows you how to add a new visual rule group for the mesh colorization augmented UI layer.

1. Under the rules tab in the console, enter a name such as ColorRule in the text field and choose **Add New Rule Group**.



2. Define a new rule for your use case. For example, you can create one based on the data property 'temperature', where the reported value is less than 20. Use the following syntax for rule expressions: Less than is **<**, greater than is **>**, less than or equal is **<=**, greater than or equal is **>=**, and equal is **==**. (For more information, see the Apache Commons JEXL syntax.)
3. Set the target to a color. To define a color, such as `#fcba03`, use hex values. (For more information about hex values, see Hexadecimal.)

Creating tags for your scenes

A tag is an annotation added to a specific `x`, `y`, `z` coordinate position of a scene. The tag uses an entity property to connect a scene part to the knowledge graph. You can use a tag to configure the behavior or visual appearance of an item in the scene, such as an alarm.

> **Note**
>
> To add functionality to tags, you apply visual rules to them.

Use the following procedure to add tags to your scene.

1. Select an object in the hierarchy, choose the **+** button, and then choose **Add Tag**.
2. Name the tag. Then, to apply a visual rule, select a visual group Id.
3. In the dropdown lists, choose the EntityID, ComponentName, and PropertyName.
4. To populate the Data Path field, choose **Create DataFrameLabel**.

3D Tiles model format

Using 3D Tiles in your scene

If you experience long wait times when you load 3D scenes in AWS IoT TwinMaker or have poor rendering performance when you navigate a complex 3D model, then you may want to convert your models to 3D tiles. This section describes the 3D tiles format and available third-party tools. Read on to decide if 3D Tiles are right for your use case and for help getting started.

Complex model use case

A 3D model in your AWS IoT TwinMaker scene may cause performance issues like slow loading times and lagging navigation if the model is:

- **Large**: its file size is larger than 100MB.
- **Dense**: it is made up of hundreds or thousands of distinct meshes.
- **Complex**: mesh geometry has millions of triangles to form complex shapes.

3D Tiles format

The 3D Tiles format is a solution for streaming model geometry and improving 3D rendering performance. It enables instantaneous loading of 3D models in an AWS IoT TwinMaker scene, and optimizes 3D interactions by loading in chunks of a model based on what is visible in the camera view.

The 3D Tiles format was created by Cesium. Cesium has a managed service to convert 3D models to 3D Tiles called Cesium Ion. This is currently the best solution for creating 3D Tiles, and we recommend this for your complex models in the supported formats. You can register Cesium and choose the appropriate subscription plan based on your business requirements on Cesium's pricing page.

To prepare a 3D Tiles model that you can add to an AWS IoT TwinMaker scene, follow the instructions documented by Cesium Ion:

- Import a model to Cesium Ion

Upload Cesium 3D tiles to AWS

Once your model has been converted to 3D Tiles, download the model files then upload them to your AWS IoT TwinMaker workspace Amazon S3 bucket:

1. Create and download your 3D Tiles model archive.
2. Unzip the archive into a folder.
3. Upload the entire 3D Tiles folder into the Amazon S3 bucket associated with your AWS IoT TwinMaker workspace. (See Uploading objects in the Amazon S3 User Guide.)
4. If your 3D Tiles model was uploaded successfully, you will see an Amazon S3 folder path in your AWS IoT TwinMaker Resource Library with type `Tiles3D`.

> **Note**
>
> The AWS IoT TwinMaker Resource Library doesn't support directly uploading 3D Tiles models.

Using 3D Tiles in AWS IoT TwinMaker

AWS IoT TwinMaker is aware of any 3D Tiles model uploaded to your workspace S3 bucket. The model must have a `tileset.json` and all dependent files (.gltf, .b3dm, .i3dm, .cmpt, .pnts) available in the same Amazon S3 directory. The Amazon S3 directory path will appear in the Resource Library with the type `Tiles3D`.

To add the 3D Tiles model to your scene, follow these steps:

1. On the scene composer page, choose the plus (**+**) sign, and then choose **Add 3D model**.
2. On the **Add resource from resource library** window, choose the path to your 3D Tiles model with the type `Tiles3D`, and then choose **Add**.
3. Click on the canvas to place the model in your scene.

3D Tiles differences

3D Tiles does not currently support geometric and semantic metadata, which means that the mesh hierarchy of the original model is not available for the sub-model selection feature. You can still add widgets to your 3D Tiles model, but you cannot use features fine-tuned to sub-models: model shader, separated 3D transformations, or entity binding for a sub-model mesh.

It is recommended to use the 3D Tiles conversion for large assets that serve as context for the background of a scene. If you want a sub-model to be further broken down and annotated then it should be extracted as a separate glTF/glb asset and added directly to the scene. This can be done with free and common 3D tools like Blender.

Example use case:

- You have a 1GB model of a factory with detailed machine rooms and floors, electrical boxes, and plumbing pipes. The electrical boxes and pipes need to glow red when associated property data cross a threshold.
- You isolate the box and pipe meshes in the model and export it into a separate glTF using Blender.
- You convert the factory without electrical and plumbing elements into a 3D Tiles model and upload it to S3.
- You add both the 3D Tiles model and glTF model to an AWS IoT TwinMaker scene at the origin (0,0,0).

- You add model shader components to the electrical box and pipe sub-models of the glTF to make the meshes red based on property rules.

Dynamic scenes

AWS IoT TwinMaker scenes unlock the power of the knowledge graph by storing scene nodes and settings in an entity component. Use the AWS IoT TwinMaker console to create **dynamic scenes** to more easily manage, build, and render 3D scenes.

Key features:

- All 3D scene node objects, settings, and data bindings are rendered "dynamically" based on knowledge graph queries.
- If you use the read-only Scene Viewer in a Grafana or custom application, you can get updates to your scenes on a 30 second interval.

Static versus dynamic scenes

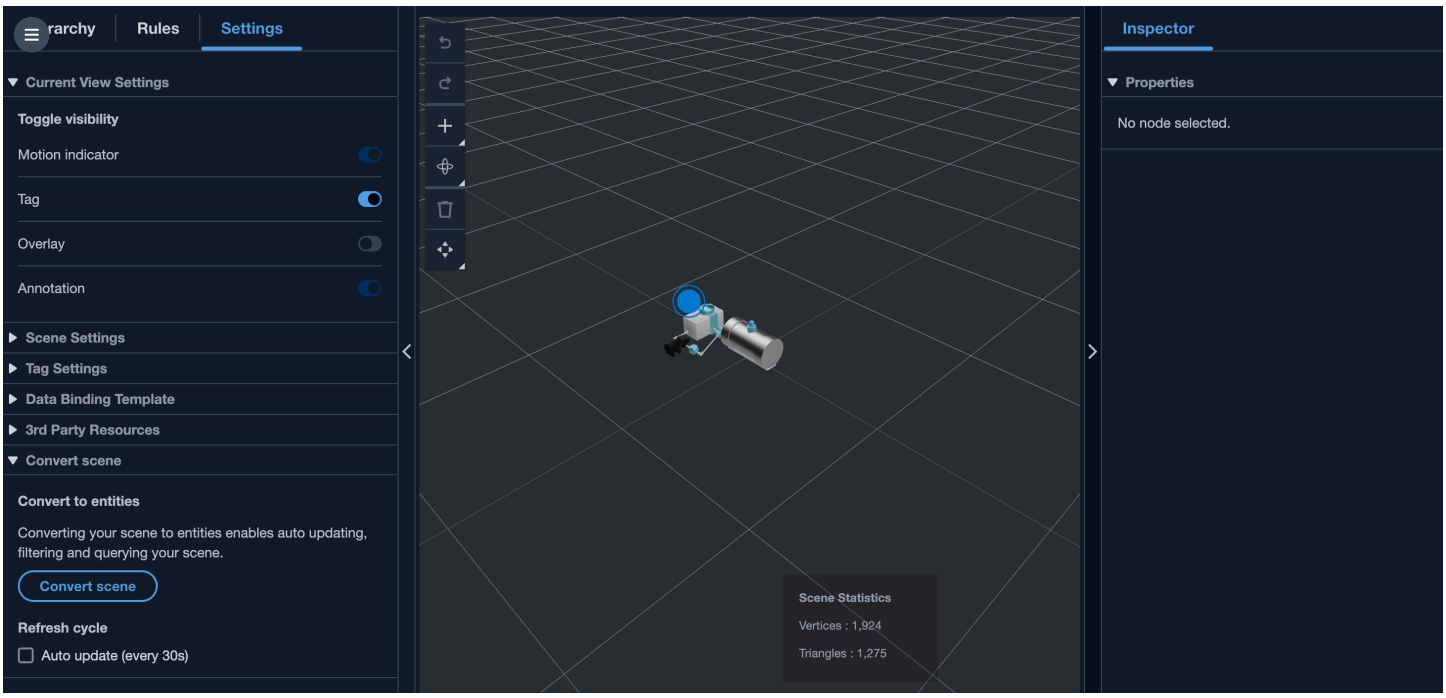
Static scenes are composed of a scene JSON file stored in S3 that has details of all scene nodes and settings. Any change to the scene must be made to the JSON document and saved to S3. A static scene is the only option if you have a basic pricing plan.

Dynamic scenes are composed of a scene JSON file that has global settings for the scene, while all other scene nodes and node settings are stored as entity components in the knowledge graph. Dynamic scenes are only supported in standard and tiered bundle pricing plans. See Switch AWS IoT TwinMaker pricing modes for information on how to upgrade your pricing plan).

You can convert an existing static scene to a dynamic scene by following these steps:

- Navigate to your scene in the AWS IoT TwinMaker console.
- On the left hand panel, click the **Settings** tab.
- Expand the **Convert scene** section at the bottom of the panel.
- Click the **Convert scene** button, then click **Confirm**.

> **Warning**
>
> The conversion from a static to dynamic scene is irreversible.



Scene component types and entities

In order to create scene-specific entity components, the following 1P component types are supported:

- **com.amazon.iottwinmaker.3d.component.camera** A component type that stores the settings of a camera widget.
- **com.amazon.iottwinmaker.3d.component.dataoverlay** A component type that stores the settings for an overlay of an annotation or tag widget.
- **com.amazon.iottwinmaker.3d.component.light** A component type that stores the settings of a light widget.
- **com.amazon.iottwinmaker.3d.component.modelref** A component type that stores the settings and S3 location of a 3D model used in a scene.
- **com.amazon.iottwinmaker.3d.component.modelshader** A component type that stores the settings of a model shader on a 3D model.

- **com.amazon.iottwinmaker.3d.component.motionindicator** A component type that stores the settings of a motion indicator widget.
- **com.amazon.iottwinmaker.3d.component.submodelref** A component type that stores the settings of a submodel of a 3D model.
- **com.amazon.iottwinmaker.3d.component.tag** A component type that stores the settings of a tag widget.
- **com.amazon.iottwinmaker.3d.node** A component type that stores the basic settings of a scene node like its 3D transform, name, and generic properties.

Dynamic scene concepts

Dynamic scene entities are stored under a global entity labelled $SCENES. Each scene is made up of a root entity and a hierarchy of children entities that match the scene node hierarchy. Each scene node under the root has a **com.amazon.iottwinmaker.3d.node** component and a component for the type of node (3D model, widget, and so on).

> **Warning**
>
> Do not manually delete any scene entities or your scene may be in a broken state. If you want to partially or fully delete a scene, use the scene composer page to add and delete scene nodes, and use the scenes page to select and delete a scene.

Create a customized web application using AWS IoT TwinMaker UI Components

AWS IoT TwinMaker provides open-source UI components for AWS IoT Application developers. Using those UI components, developers can build customized web applications with AWS IoT TwinMaker feature enabled for their digital twins.

AWS IoT TwinMaker UI components are part of the AWS IoT Application Kit, an open-source, client-side library that enables IoT application developers to simplify the development of complex IoT applications

AWS IoT TwinMaker UI components include:

- **AWS IoT TwinMaker source**:

  A data connector component that enables you to retrieve data and interact with your AWS IoT TwinMaker data and digital twins.

  For more information, see AWS IoT TwinMaker source documentation.
- **Scene viewer**:

  A 3D rendering component built over `@react-three/fiber` that renders your digital twin and enables you to interact with it.

  For more information, see Scene Viewer documentation.
- **Video player**:

  A video player component that allows you to stream a video from the Kinesis Video Streams through AWS IoT TwinMaker.

  For more information, see Video Player documentation.

To learn more about using AWS IoT Application Kit, please visit AWS IoT Application Kit Github page.

For instructions on how to start a new web application using AWS IoT Application Kit, please visit the official IoT App Kit documentation page.

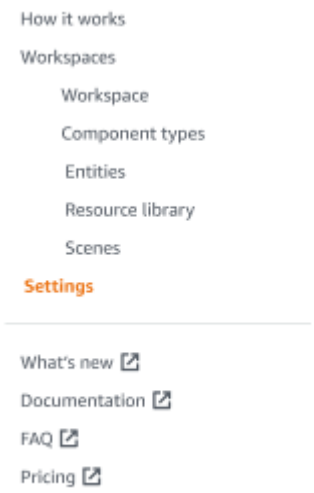
Switch AWS IoT TwinMaker pricing modes

AWS IoT TwinMaker currently has three pricing modes, basic, standard or tiered bundle. Standard pricing mode is set as the default pricing mode.

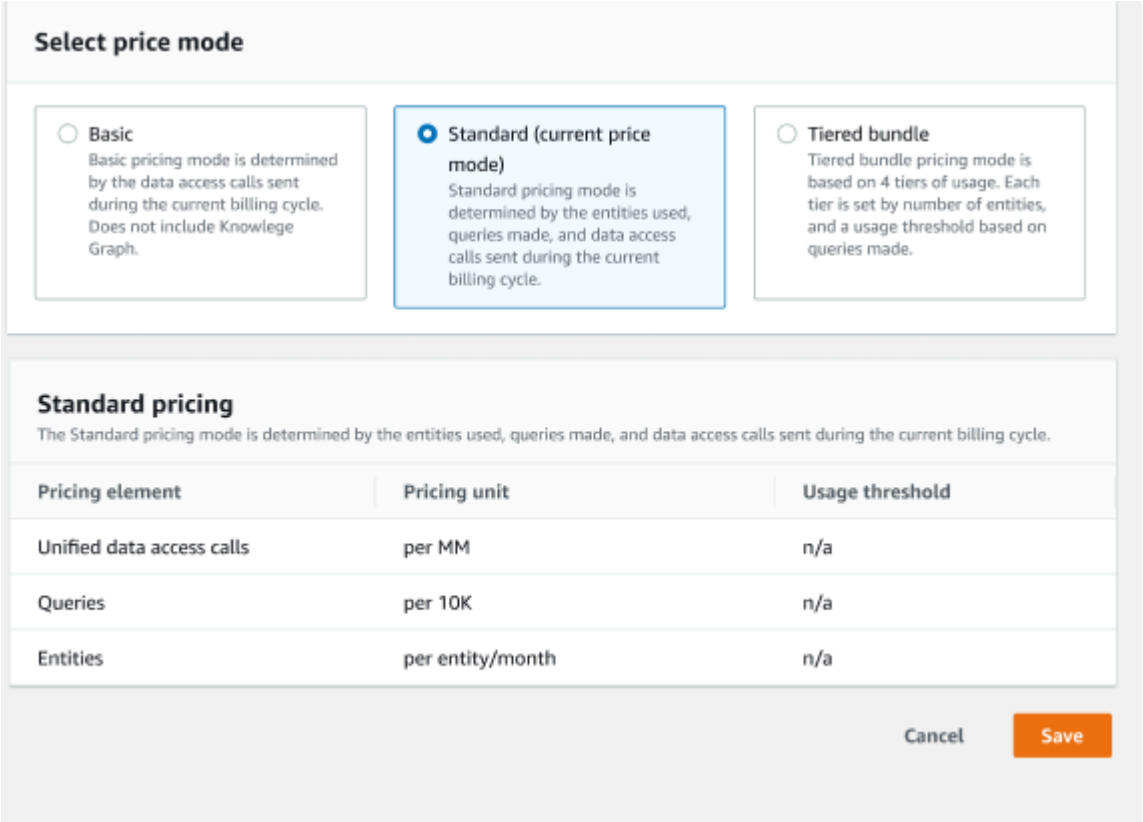
You can switch from the usage-based to the tiered-based pricing mode at any time, but the change takes effect at the beginning of your next billing cycle. Once you have switched from usage-based to the tiered-based pricing mode, you cannot switch back to the usage-based pricing mode for the next three usage cycles. If you switch from basic to standard, the change is effective immediately. For details and cost information, see AWS IoT TwinMaker Pricing

This procedure shows you how to switch your pricing mode in the AWS IoT TwinMaker console:

1. Open the AWS IoT TwinMaker console.
2. In the left navigation pane, select **Settings**. The **Pricing** page opens.



3. Choose **Change price mode**.
4. Select either the **Standard** or **Tiered bundle** modes, as shown in the following screenshot.



5. Choose **Save** to confirm your new pricing mode.
6. You have now changed your pricing mode.

> **Note**
>
> You can switch from the usage-based to the tiered-based pricing mode at any time, but the change takes effect at the beginning of your next billing cycle. Once you have switched from usage-based to the tiered-based pricing mode, you cannot switch back to the usage-based pricing mode for the next three usage cycles. If you switch from basic to standard, the change is effective immediately.

AWS IoT TwinMaker knowledge graph

The AWS IoT TwinMaker knowledge graph organizes all the information contained within your AWS IoT TwinMaker workspaces and presents it in a visual graph format. You can run queries against your entities, components, and component types to generate visual graphs that show you the relationships between your AWS IoT TwinMaker resources.

The following topics show you how to use and integrate the knowledge graph.

Topics

AWS IoT TwinMaker knowledge graph core concepts

This topic covers the key concepts and vocabulary of the knowledge graph feature.

How knowledge graph works:

Knowledge graph creates relationships between entities and their components with the existing CreateEntity or UpdateEntity APIs. A relationship is just a property of a special data type RELATIONSHIP that is defined on a component of an entity. AWS IoT TwinMaker knowledge graph calls the ExecuteQuery API to make a query based on any data in the entities or the relationships between them. Knowledge graph uses the flexible PartiQL query language (used by many AWS services) that has newly added graph match syntax support to help you write your queries. After the calls are made, you can view the results as a table or visualize them as a graph of connected nodes and edges.

Knowledge graph key terms:

- **Entity graph**: A collection of nodes and edges within a workspace.
- **Node**: Every entity in your workspace becomes a node in the entity graph.
- **Edge**: Every relationship property defined on a component of an entity becomes an edge in the entity graph. In addition, a hierarchical parent-child relationship defined using the

parentEntityId field of an entity also becomes an edge in the entity graph with an "isChildOf" relationship name. All edges are directional edges.

- **Relationship**: An AWS IoT TwinMaker Relationship is a special type of property of an Entity's component. You can use the AWS IoT TwinMaker CreateEntity or UpdateEntity API to define and edit a relationship. In AWS IoT TwinMaker, a relationship must be defined in a component of an entity. A relationship cannot be defined as an isolated resource. A relationship must be directional from one entity to another.

How to Run AWS IoT TwinMaker knowledge graph queries

Before you use the AWS IoT TwinMaker knowledge graph, make sure you have completed the following prerequisites:

- Create an AWS IoT TwinMaker workspace. You can create a workspace in the AWS IoT TwinMaker console.
- Become familiar with AWS IoT TwinMaker's entity-component system and how to create entities. For more information, see Create your first entity.
- Become familiar with AWS IoT TwinMaker's data connectors. For more information, see AWS IoT TwinMaker data connectors.

> **Note**
>
> In order to use the AWS IoT TwinMaker knowledge graph, you need to be in either the **standard** or **tiered bundle** pricing modes. For more information, see Switch AWS IoT TwinMaker pricing modes.

The following procedures show you how to write, run, save, and edit queries.

Open the query editor

To navigate to the knowledge graph query editor

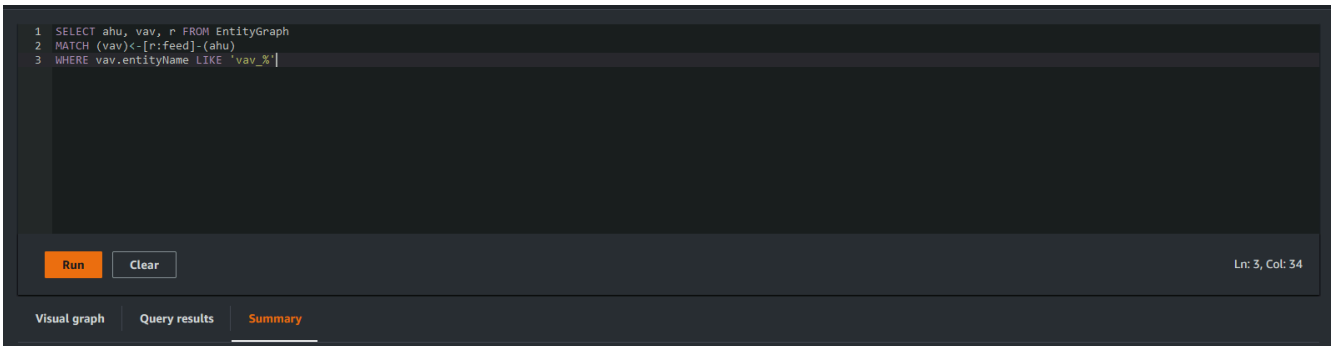
1. Open the AWS IoT TwinMaker console.
2. Open the workspace in which you wish to use knowledge graph.
3. In the left navigation menu, choose **Query editor**.

4. The query editor opens. You are now ready to run queries on your workspace's resources.

Run a query

To run a query and generate a graph

1. In the query editor, choose the **Editor** tab to open the syntax editor.
2. In the editor space, write the query you wish to run against your workspace's resources.



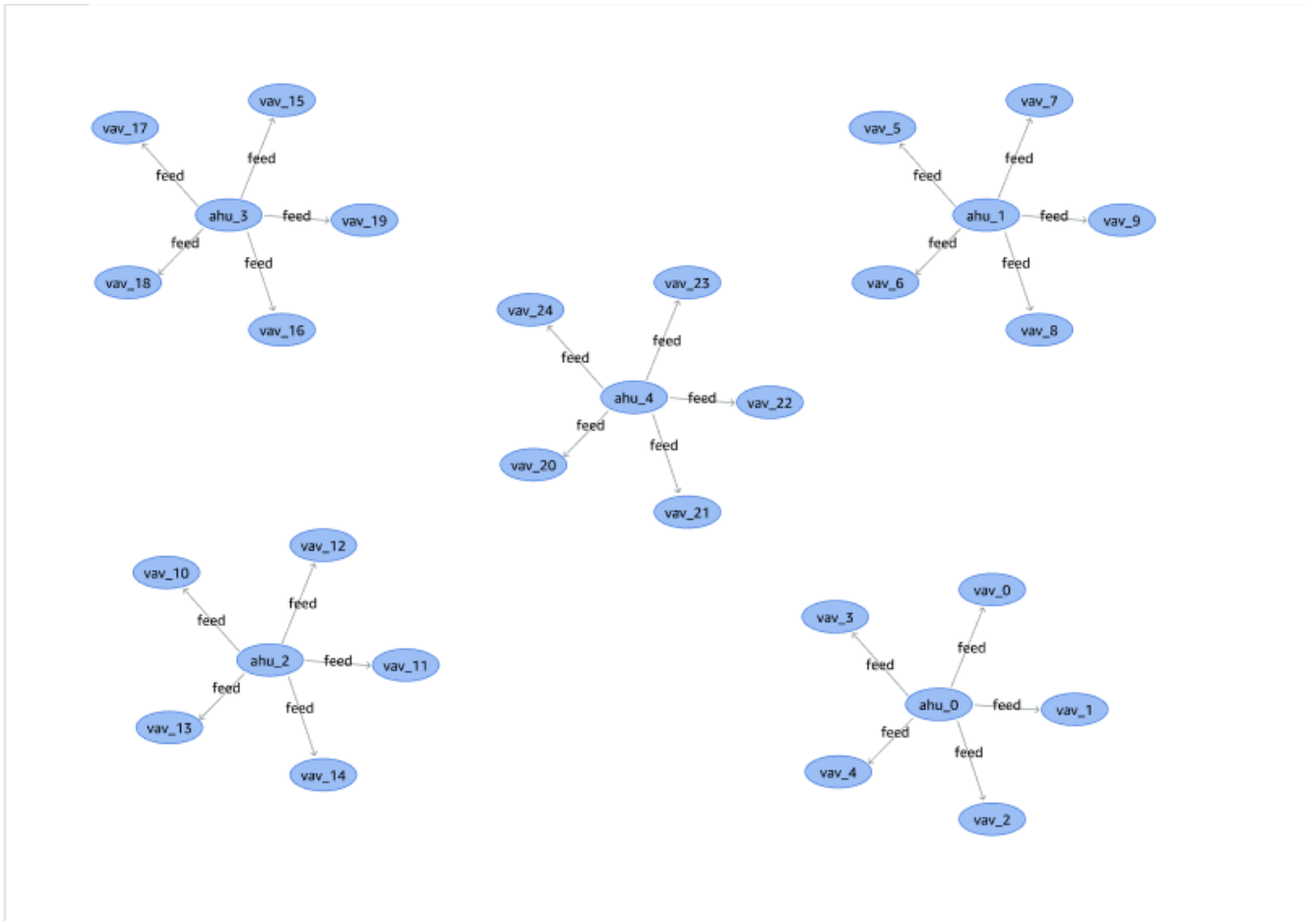
In the example shown, the request searches for entities that contain `vav_%` in their name, then organizes these entities by the `feed` relationship between them, using the following code.

```
SELECT ahu, vav, r FROM EntityGraph
MATCH (vav)<-[r:feed]-(ahu)
WHERE vav.entityName LIKE 'vav_%'
```

> **Note**
>
> The knowledge graph syntax uses PartiQL. For information on this syntax, see AWS IoT TwinMaker knowledge graph additional resources.

3. Choose **Run query** to run the request you created.

   A graph is generated based on your request.



The example graph shown above is based on the query example in step 2.

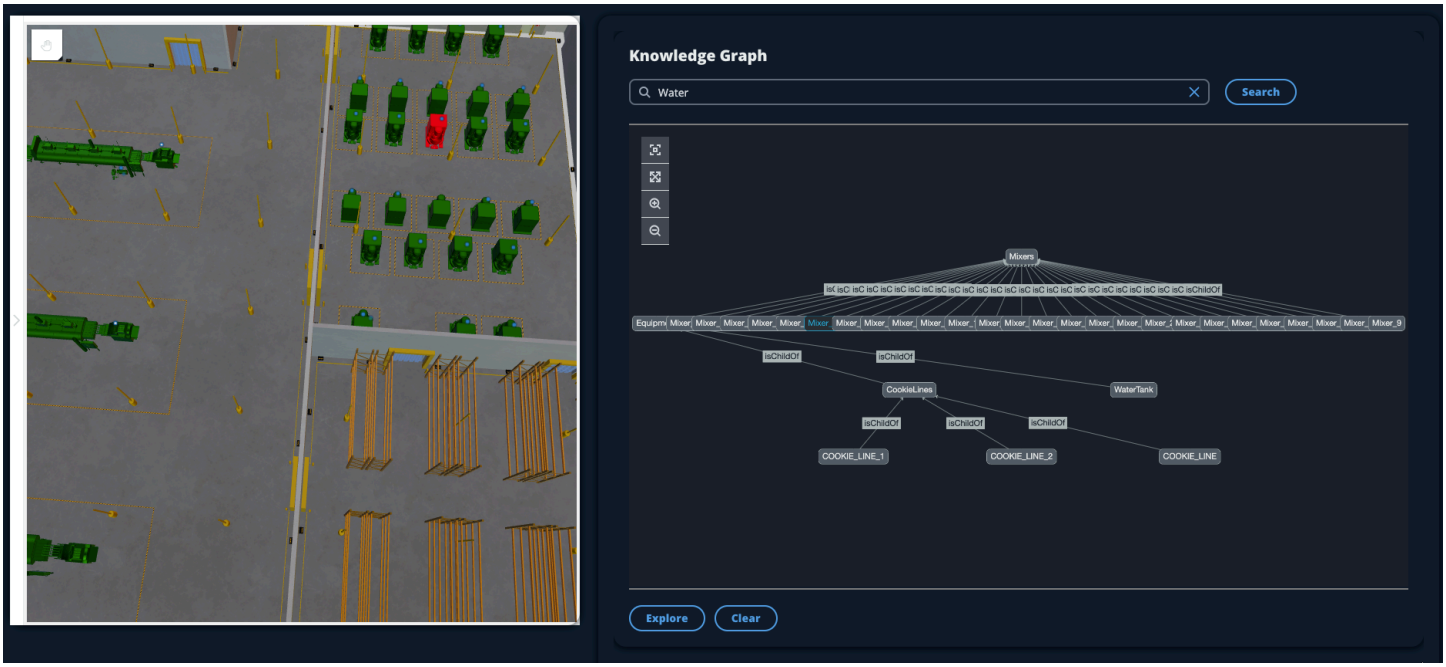
4. The results of the query are also presented in a list. Choose **results** to view the query results in a list.
5. Optionally, choose **Export as** to export the query results in JSON or CSV format.

This covers the basic use of knowledge graph in the console. For more information and examples demonstrating the knowledge graph syntax, see AWS IoT TwinMaker knowledge graph additional resources.

Knowledge graph scene integration

You can use AWS IoT app kit components to build a web application that integrates knowledge graph into your AWS IoT TwinMaker scenes. This allows you to generate graphs based on the 3D nodes (the 3D models which represent your equipment or systems) that are present within your scene. To create an application that graphs 3D nodes from your scene, first bind the 3D nodes

to entities in your workspace. With this mapping, AWS IoT TwinMaker graphs the relationships between the 3D models present in your scene and the entities in your workspace. Then you can create a web application, select 3D models with your scene, and explore their relationships to other entities in a graph format.



For an example of a working web application that utilizes the AWS IoT app kit components to generate graphs in an AWS IoT TwinMaker scene, see the AWS IoT TwinMaker sample react app on github.

AWS IoT TwinMaker scene graph prerequisites

Before you create a web app that uses AWS IoT TwinMaker knowledge graph in your scenes, complete the following prerequisites:

- Create an AWS IoT TwinMaker workspace. You can create a workspace in the AWS IoT TwinMaker console.
- Become familiar with AWS IoT TwinMaker's entity-component system and how to create entities. For more information, see Create your first entity.
- Create an AWS IoT TwinMaker scene populated with 3D models.
- Become familiar with AWS IoT TwinMaker's AWS IoT app kit components. For more information on the AWS IoT TwinMaker components, see Create a customized web application using AWS IoT TwinMaker UI Components.

- Become fimalliar with knowledge graph concepts and key terminology. See AWS IoT TwinMaker knowledge graph core concepts.

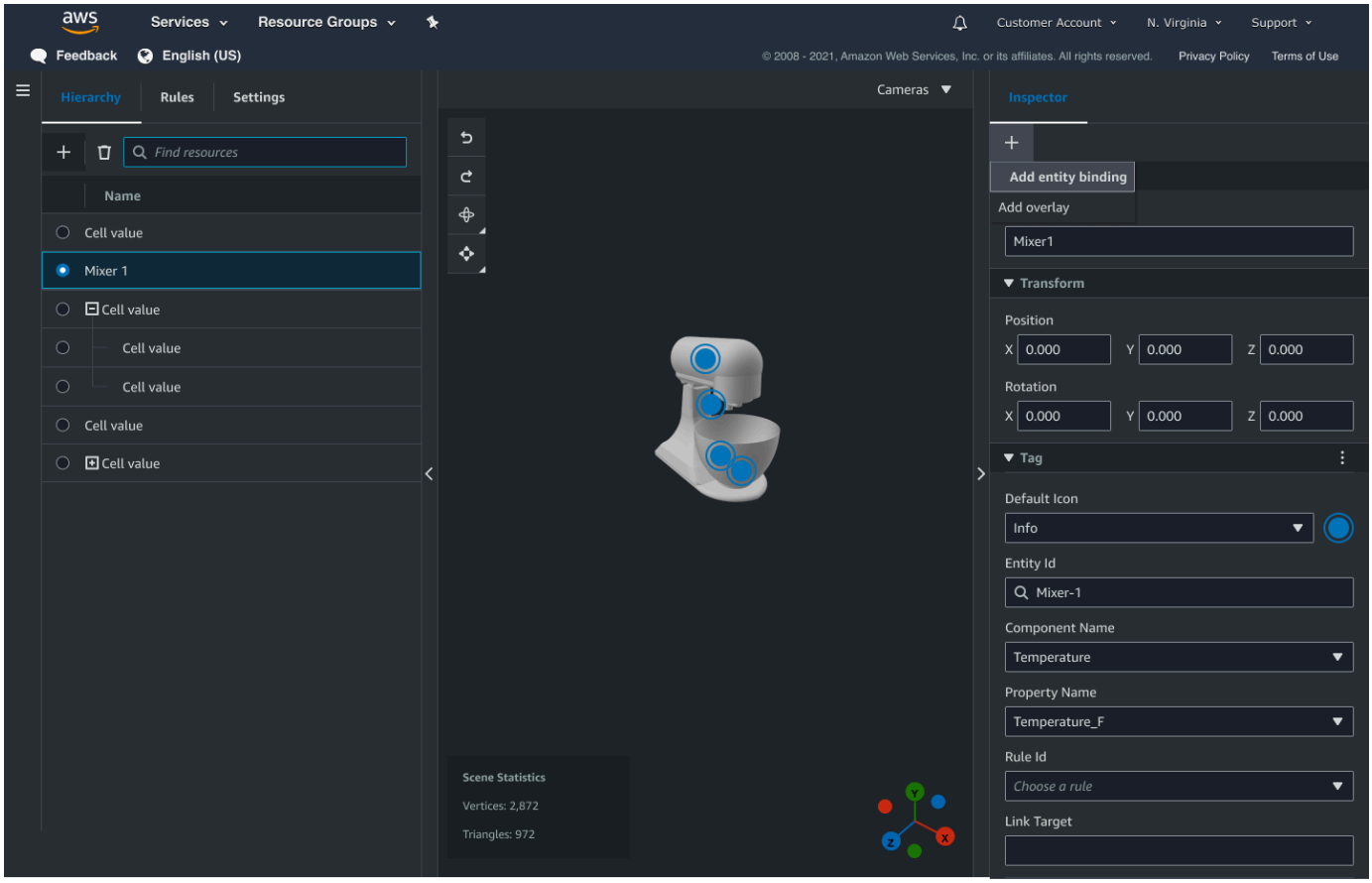
> **Note**
>
> To use the AWS IoT TwinMaker knowledge graph and any related features, you need to be in either the **standard** or **tiered bundle** pricing modes. For more information on AWS IoT TwinMaker pricing, see Switch AWS IoT TwinMaker pricing modes.

Bind 3D nodes in your scene

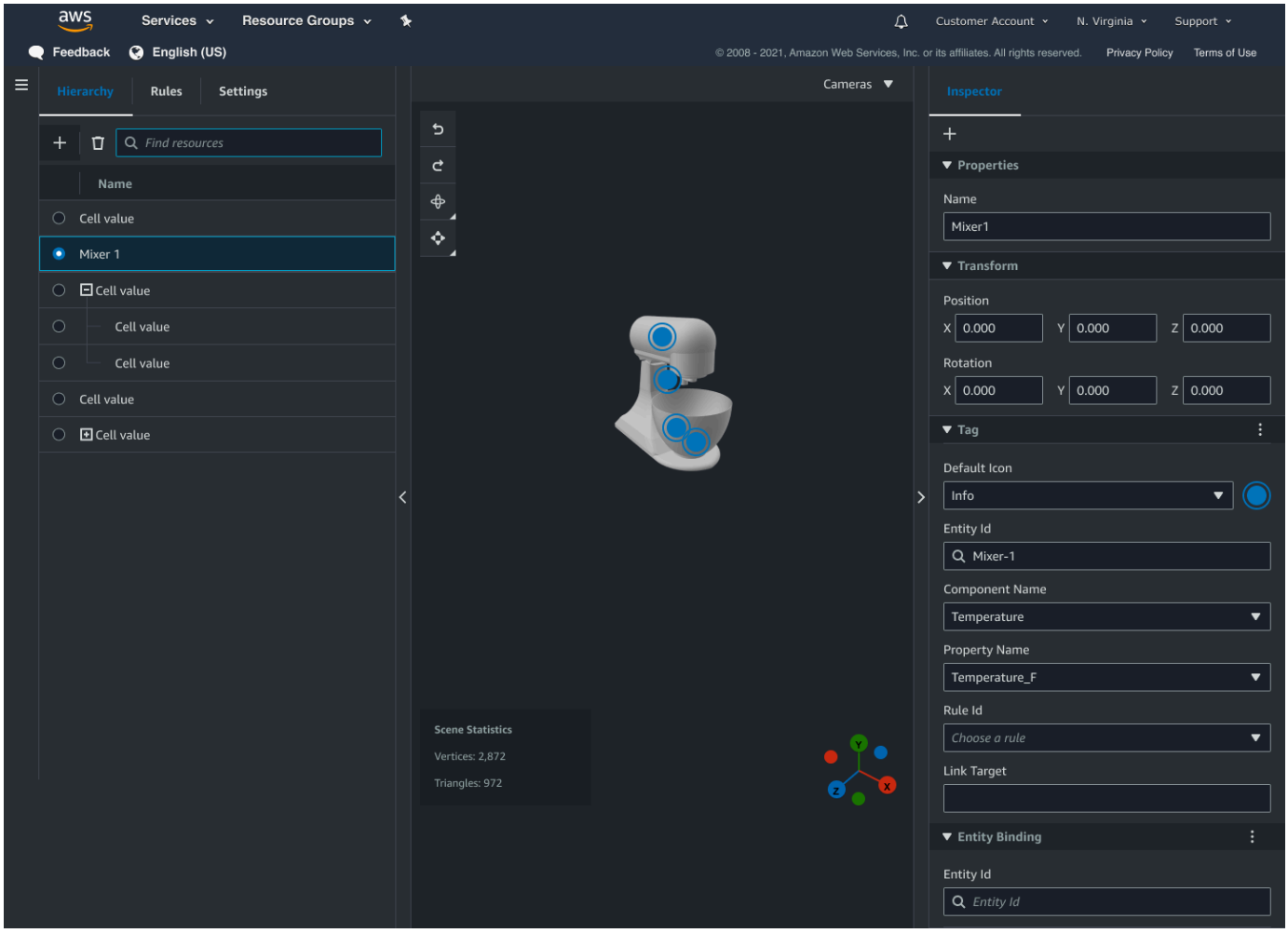
Before you create a web app that integrates knowledge graph with your scene, bind the 3D models, referred to as 3D nodes, that are present in your scene to the associated workspace entity. For example, if you have a model of mixer equipment in a scene, and a corresponding entity called `mixer_0`, create a **data binding** between the model of the mixer and the entity representing the mixer– so that the model and entity can be graphed.

To perform a data binding action

1. Log in to the AWS IoT TwinMaker console.
2. Open your workspace and select a scene with the 3D nodes you wish to bind.
3. Select a node (3D model) in the scene composer. When you select a node, it will open an inspector panel on the right side of the screen.
4. In the inspector panel, navigate to the top of the panel and select the **+** button. Then choose the **Add entity binding** option. This will open a drop-down where you can select an entity to bind to your currently selected node.



5. From the data binding drop-down menu, select the entity id you want to map to the 3D model. For the **Component name** and **Property name** fields, select the components and properties you want to bind.



Once you have made selections for the **Entity Id**, **Component Name** and **Property Name** fields, the binding is complete.

6. Repeat this process for all models and entities you want to graph.

> **Note**
>
> The same data binding operation can be performed on your scene tags, simply select a tag instead of an entity and follow the same process to bind the tag to a node.

Create a web application

After you bind your entities, use the AWS IoT app kit library to build a web app with a knowledge graph widget that lets you view your scene and explore the relationships between your scene nodes and entities.

Use the following resources to create your own app:

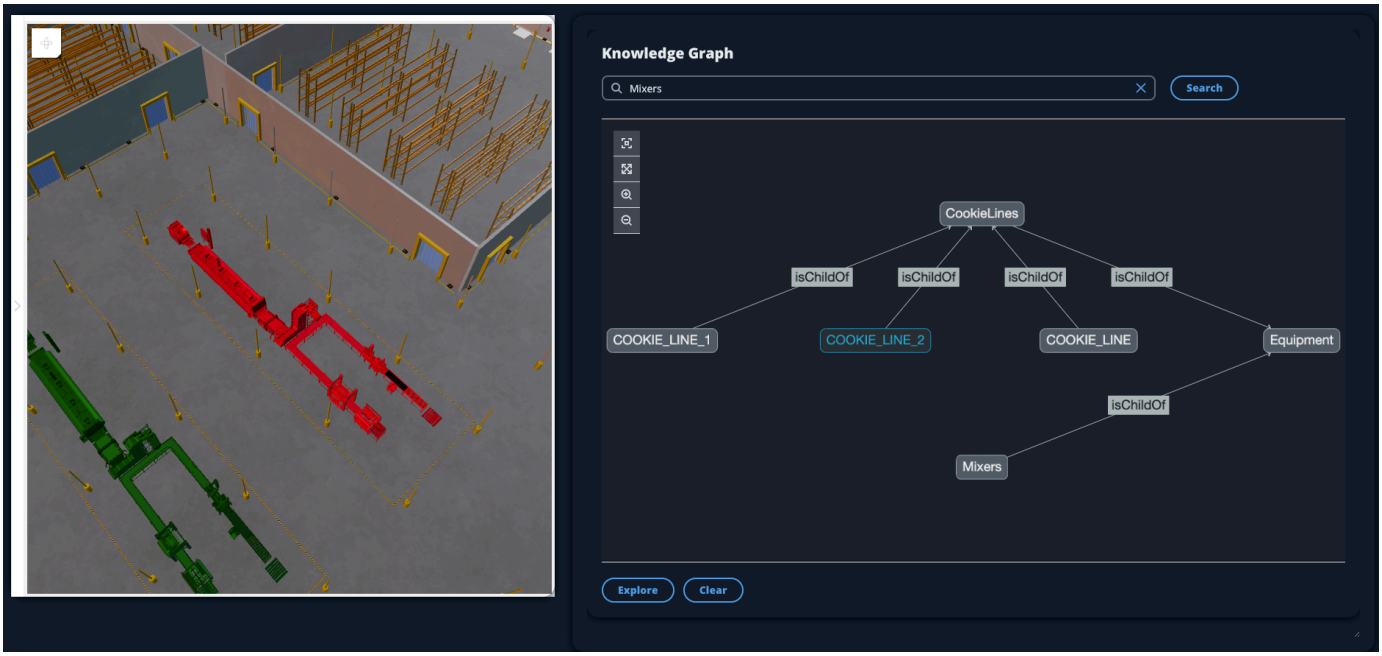
- The AWS IoT TwinMaker sample react app github Readme documentation.
- The AWS IoT TwinMaker sample react app source on github.
- The AWS IoT app kit Getting started documentation.
- The AWS IoT app kit Video Player component documentation.
- The AWS IoT app kit Scene Viewer component documentation.

The following procedure demonstrates the functionality of the scene viewer component in a web app.

> **Note**
>
> This procedure is based on the implementation of the AWS IoT app kit scene viewer component in the AWS IoT TwinMaker sample react app.

1. Open the scene viewer component of the AWS IoT TwinMaker sample react app. In the search field type an entity name or partial entity name (case sensitive search) then select the **Search** button. If a model is bound to the entity id, then the model in the scene will be highlighted and a node of the entity will be shown in the scene viewer panel.

2. To generate a graph of all relationships, select a node in the scene viewer widget and select the **Explore** button.



3. Press the **Clear** button to clear your current graph selection and start over.

How to use AWS IoT TwinMaker knowledge graph with Grafana

This section shows you how to add a query editor panel to your AWS IoT TwinMaker Grafana dashboard to run and display queries.

AWS IoT TwinMaker query editor prerequisites

Before you use the AWS IoT TwinMaker knowledge graph in Grafana, complete the following prerequisites:

- Create an AWS IoT TwinMaker workspace. You can create a workspace in the AWS IoT TwinMaker console.
- Configure AWS IoT TwinMaker for use with Grafana. For instructions, see AWS IoT TwinMaker Grafana dashboard integration.

> **Note**
>
> To use the AWS IoT TwinMaker knowledge graph, you need to be in either the **standard** or **tiered bundle** pricing modes. For more information, see Switch AWS IoT TwinMaker pricing modes.

AWS IoT TwinMaker query editor permissions

To use the AWS IoT TwinMaker query editor in Grafana, you must have an IAM role with permission for the action `iottwinmaker:ExecuteQuery`. Add that permission to your workspace dashboard role, as shown in this example:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:GetEntity",
                "iottwinmaker:ListEntities",
                "iottwinmaker:ExecuteQuery"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-2:111122223333:workspace/workspaceId",
                "arn:aws:iottwinmaker:us-
east-2:111122223333:workspace/workspaceId/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        }
    ]
}
```

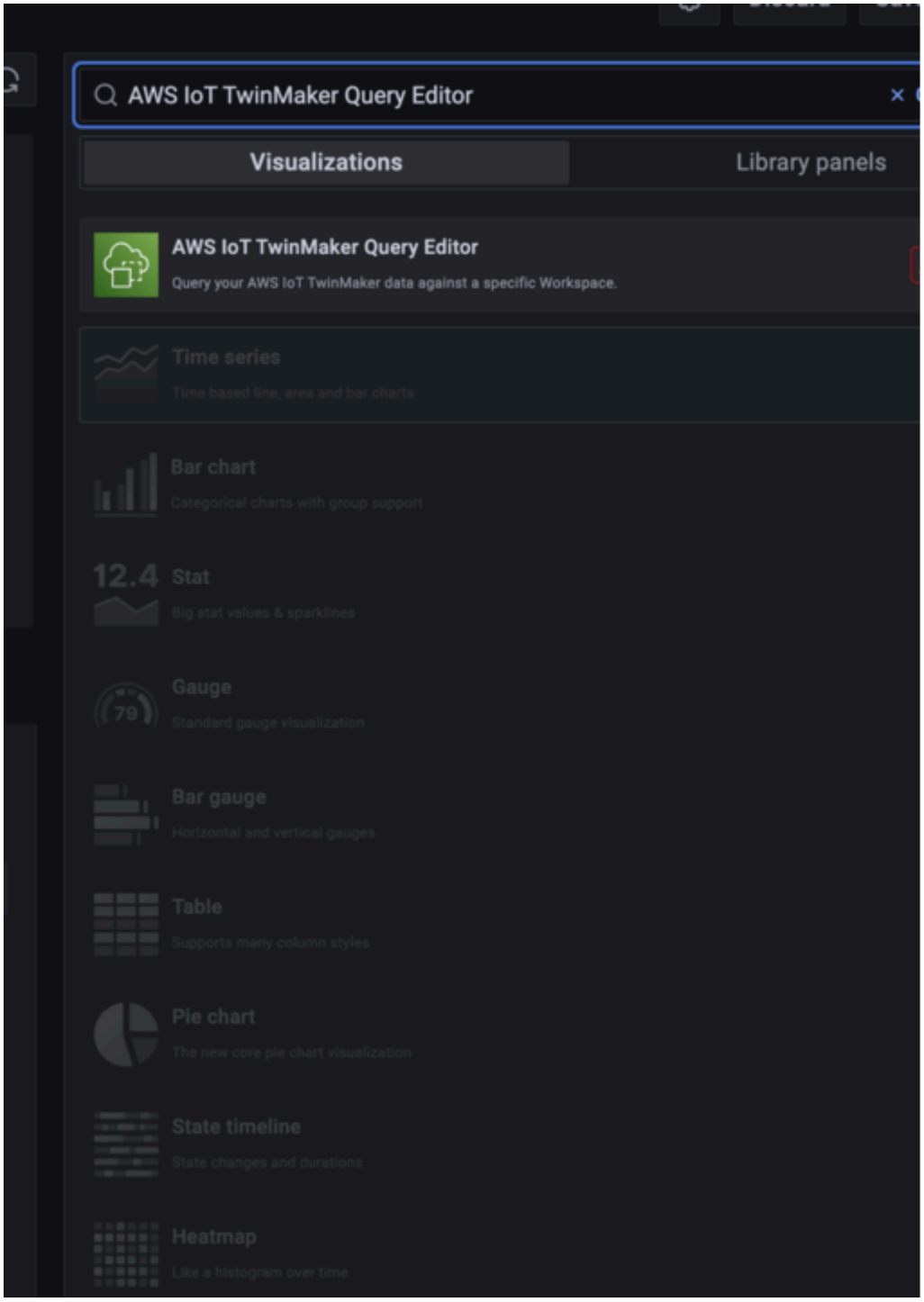
> **Note**
>
> When you configure your AWS IoT TwinMaker Grafana data source, make sure to use the role with this permission for the **Assume role ARN** field. After you add it, you can select your workspace from the dropdown next to **Workspace**.

For more information, see Creating a dashboard IAM role.

Set up the AWS IoT TwinMaker query editor panel

To set up a new Grafana dashboard panel for knowledge graph

1. Open your AWS IoT TwinMaker Grafana dashboard.
2. Create a new **dashboard panel**. For detailed steps on how to create a panel, see Create a dashboard in the Grafana documentation.
3. From the list of visualizations, select **AWS IoT TwinMaker Query Editor**.



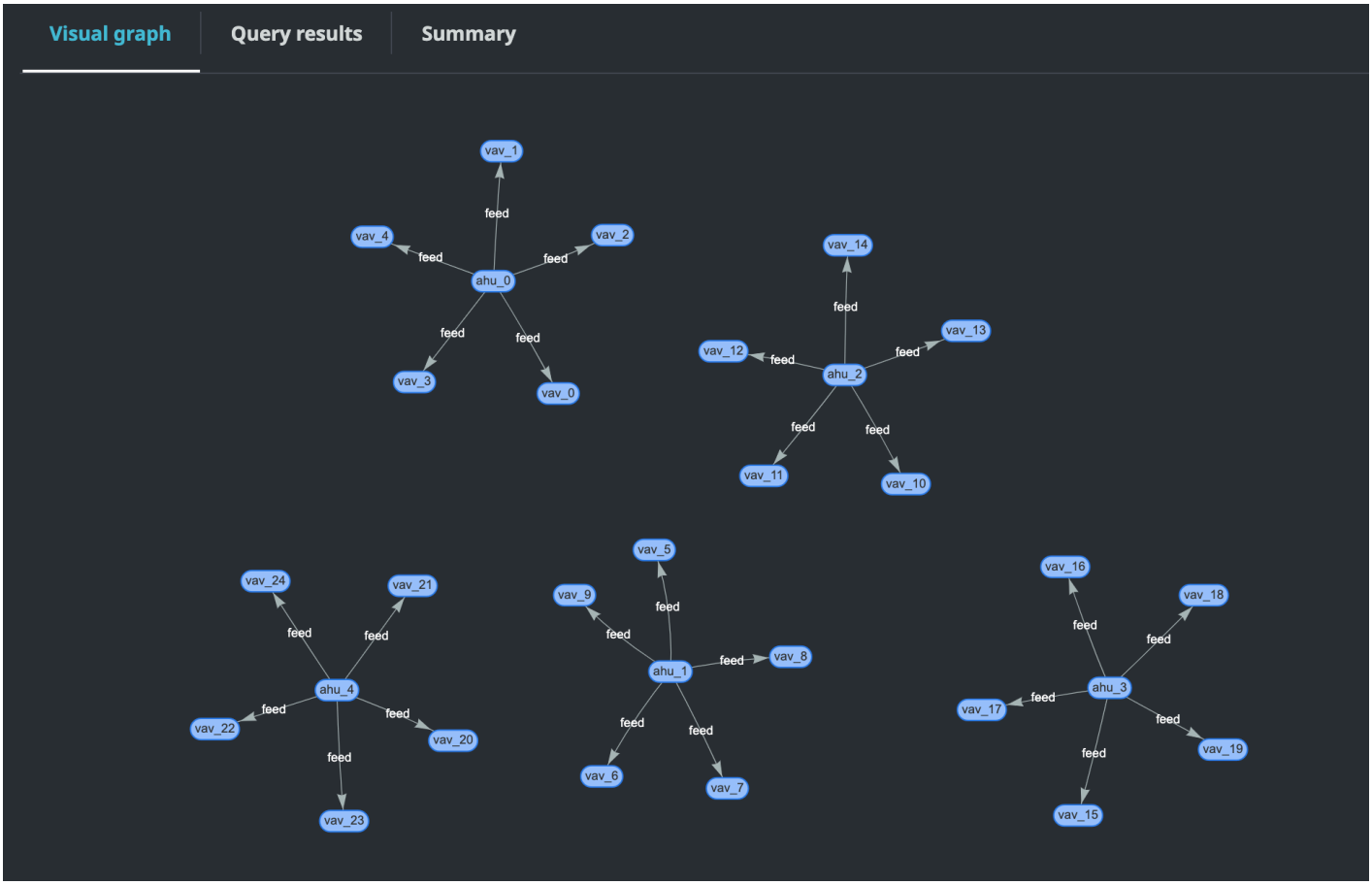
4. Select the data source to run queries against.
5. **(Optional)** Add a name for the new panel in the provided field.
6. Select **Apply** to save and confirm your new panel.

The knowledge graph panel works in a similar way as the query editor provided in the AWS IoT TwinMaker console. You can run, write, and clear queries you make in the panel. For more information on how to write queries, see AWS IoT TwinMaker knowledge graph additional resources.

How to use the AWS IoT TwinMaker query editor

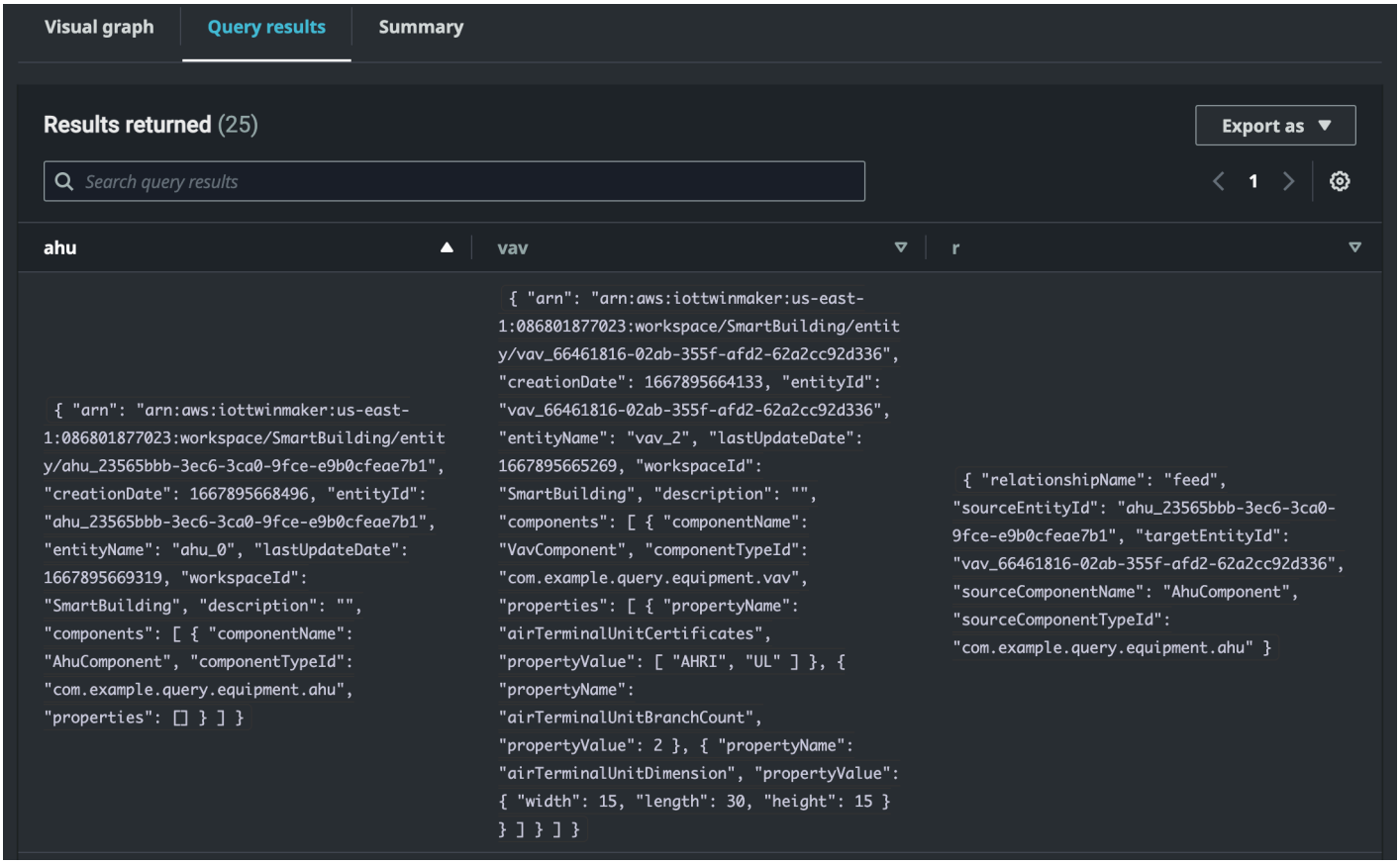
The results of your queries are displayed in three ways, as shown in the following images: visualized in a graph, listed in a table, or presented as a run summary.

- **Graph visualization:**



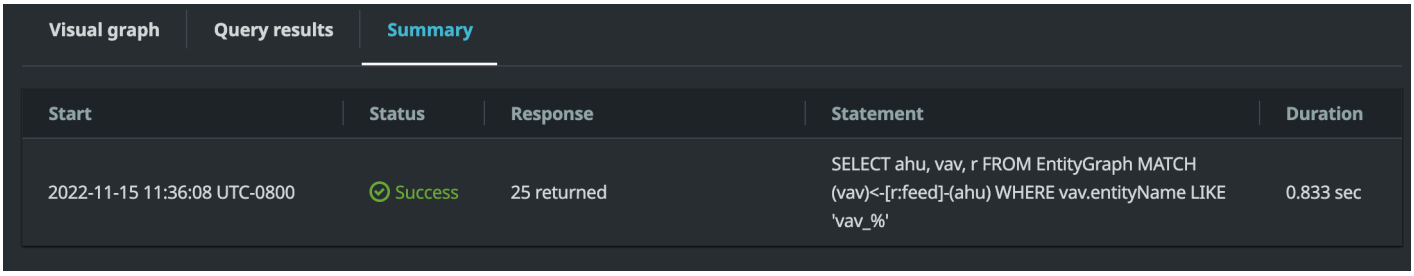
The visual graph only displays data for queries that have at least one relation in the result. The graph displays entities as nodes and relationships as directed edges in the graph.

- **Tabular data:**



The tabular data format displays the data for all queries. You can search the table for specific results or subsets of the results. The data can be exported in JSON or CSV format.

- **Run summary**



The run summary displays the query and metadata about the status of the query.

AWS IoT TwinMaker knowledge graph additional resources

This section provides basic examples of the PartiQL syntax used to write queries in the knowledge graph, as well as links to PartiQL documentation that provide information on the knowledge graph data model.

- PartiQL graph data model documentation
- PartiQL graph query documentation

This set of examples shows basic queries with their responses. Use this as a reference to write your own queries.

Basic queries

- **Get all entities with a filter**

```
SELECT entity
FROM EntityGraph MATCH (entity)
WHERE entity.entityName = 'room_0'
```

This query returns all the entities in a workspace with the name `room_0`.

`FROM` clause: `EntityGraph` is the graph collection that contains all the entities and their relationships in a workspace. This collection is automatically created and managed by AWS IoT TwinMaker based on the entities in your workspace.

`MATCH` clause: specifies a pattern that matches a portion of the graph. In this case, the pattern `(entity)` matches every node in the graph and is bound to the entity variable. The `FROM` clause must be followed by the `MATCH` clause.

`WHERE` clause: specifies a filter on the `entityName` field of the node, where the value must match `room_0`.

`SELECT` clause: specifies the `entity` variable so the whole entity node is returned.

Response:

```
{
  "columnDescriptions": [
    {
      "name": "entity",
      "type": "NODE"
    }
  ],
  "rows": [
    {
      "rowData": [
```

```
        {
          "arn": "arn:aws:iottwinmaker:us-east-1: 577476956029: workspace /
 SmartBuilding8292022 / entity / room_18f3ef90 - 7197 - 53 d1 - abab -
 db9c9ad02781 ",
          "creationDate": 1661811123914,
          "entityId": "room_18f3ef90-7197-53d1-abab-db9c9ad02781",
          "entityName": "room_0",
          "lastUpdateDate": 1661811125072,
          "workspaceId": "SmartBuilding8292022",
          "description": "",
          "components": [
            {
              "componentName": "RoomComponent",
              "componentTypeId": "com.example.query.construction.room",
              "properties": [
                {
                  "propertyName": "roomFunction",
                  "propertyValue": "meeting"
                },
                {
                  "propertyName": "roomNumber",
                  "propertyValue": 0
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```


The `columnDescriptions` returns metadata about the column, such as the name and type. The type returned is `NODE`. This indicates that the whole node has been returned. Other values for the type can be `EDGE` which would indicate a relationship or `VALUE` which would indicate a scalar value such as an integer or string.

The `rows` returns a list of rows. As only one entity was matched, one `rowData` is returned which contains all the fields in an entity.

> **Note**
>
> Unlike SQL where you can only return scalar values, you can return an object (as JSON) using PartiQL.

Each node contains all the entity-level fields such as `entityId`, `arn` and `components`, component-level fields such as `componentName`, `componentTypeId` and `properties` as well as property-level fields such as `propertyName` and `propertyValue`, all as a nested JSON.

- **Get all relationships with a filter**:

```
SELECT relationship
FROM EntityGraph MATCH (e1)-[relationship]->(e2)
WHERE relationship.relationshipName = 'isLocationOf'
```

This query returns all the relationships in a workspace with relationship name `isLocationOf`.

The `MATCH` clause: specifies a pattern that matches two nodes (indicated by `( )`) that are connected by a directed edge (indicated by `-[]->`) and bound to a variable called `relationship`.

The `WHERE` clause: specifies a filter on the `relationshipName` field of the edge, where the value is `isLocationOf`.

The `SELECT` clause: specifies the relationship variable so the whole edge node is returned.

Response

```
{
    "columnDescriptions": [{
        "name": "relationship",
        "type": "EDGE"
    }],
    "rows": [{
        "rowData": [{
            "relationshipName": "isLocationOf",
            "sourceEntityId": "floor_83faea7a-ea3b-56b7-8e22-562f0cf90c5a",
```

```
            "targetEntityId": "building_4ec7f9e9-e67e-543f-9d1b- 235df7e3f6a8",
            "sourceComponentName": "FloorComponent",
            "sourceComponentTypeId": "com.example.query.construction.floor"
        }]
    },
        ... //rest of the rows are omitted
    ]
}
```

The type of the column in `columnDescriptions` is an `EDGE`.

Each `rowData` represents an edge with fields like `relationshipName`. This is the same as the relationship property name defined on the entity. The `sourceEntityId`, `sourceComponentName` and `sourceComponentTypeId` give information about which entity and component the relationship property was defined on. The `targetEntityId` specifies which entity this relationship is pointing towards.

- **Get all entities with a specific relationship to a specific entity**

```
SELECT e2.entityName
      FROM EntityGraph MATCH (e1)-[r]->(e2)
      WHERE relationship.relationshipName = 'isLocationOf'
      AND e1.entityName = 'room_0'
```

This query returns all the entity names of all entities that have an `isLocationOf` relationship with the `room_0` entity.

The `MATCH` clause: specifies a pattern that matches any two nodes (`e1`, `e2`) that have a directed edge (`r`).

The `WHERE` clause: specifies a filter on the relationship name and source entity's name.

The `SELECT` clause: returns the `entityName` field in the `e2` node.

Response

```
{
  "columnDescriptions": [
    {
       "name": "entityName",
       "type": "VALUE"
    }
```

```
  ],
   "rows": [
    {
       "rowData": [
         "floor_0"
      ]
    }
  ]
}
```

In the columnDescriptions, the type of the column is `VALUE` since `entityName` is a string.

One entity, `floor_0`, is returned.

MATCH

The following patterns are supported in a `MATCH` clause:

- Match node 'b' pointing to node 'a':

```
FROM EntityGraph MATCH (a)-[rel]-(b)
```

- Match node 'a' pointing to node 'b':

```
FROM EntityGraph MATCH (a)-[]->(b)
```

  There is no variable bound to a relationship assuming a filter doesn't need to be specified on the relationship.

- Match node 'a' pointing to node 'b' and node 'b' pointing to node 'a':

```
FROM EntityGraph MATCH (a)-[rel]-(b)
```

  This will return two matches: one from 'a' to 'b' and another from 'b' to 'a', so the recommendation is to use directed edges wherever possible.

- The relationship name is also a label of the property graph `EntityGraph`, so you can simply specify the relationship name following a colon (:) instead of specifying a filter on `rel.relationshipName` in the `WHERE` clause.

```
FROM EntityGraph MATCH (a)-[:isLocationOf]-(b)
```

- Chaining: patterns can be chained to match on multiple relationships.

```
FROM EntityGraph MATCH (a)-[rel1]->(b)-[rel2]-(c)
```

- Variable hop patterns can span multiple nodes and edges as well:

```
FROM EntityGraph MATCH (a)-[]->{1,5}(b)
```

This query matches any pattern with outgoing edges from node 'a' within 1 to 5 hops. The allowed quantifiers are:

`{m,n}` - between m and n repetitions

`{m,}` - m or more repetitions.

FROM:

An entity node can contain nested data, such as components which themselves contain further nested data such as properties. These can be accessed by unnesting the result of the MATCH pattern.

```
SELECT e
FROM EntityGraph MATCH (e), e.components AS c, c.properties AS p
WHERE c.componentTypeId = 'com.example.query.construction.room',
AND p.propertyName = 'roomFunction'
AND p.propertyValue = 'meeting'
```

Access nested fields by dotting `.` into a variable. A comma (,) is used to unnest (or join) entities with the components inside and then the properties inside those components. `AS` is used to bind a variable to the unnested variables so that they can be used in the `WHERE` or `SELECT` clauses. This query returns all entities that contains a property named `roomFunction` with value `meeting` in a component with component type id `com.example.query.construction.room`

To access multiple nested fields of a field such as multiple components in an entity, use the comma notation to do a join.

```
SELECT e
FROM EntityGraph MATCH (e), e.components AS c1, e.components AS c2
```

SELECT:

- Return a node:

```
SELECT e
FROM EntityGraph MATCH (e)
```

- Return an edge:

```
SELECT r
FROM EntityGraph MATCH (e1)-[r]->(e2)
```

- Return a scalar value:

```
SELECT floor.entityName, room.description, p.propertyValue AS roomfunction
FROM EntityGraph MATCH (floor)-[:isLocationOf]-(room),
room.components AS c, c.properties AS p
```

  Format the name of the output field by aliasing it using AS. Here, instead of `propertyValue` as column name in the response, `roomfunction` is returned.

- Return aliases:

```
SELECT floor.entityName AS floorName, luminaire.entityName as luminaireName
FROM EntityGraph MATCH (floor)-[:isLocationOf]-(room)-[:hasPart]-
(lightingZone)-[:feed]-(luminaire)
WHERE floor.entityName = 'floor_0'
AND luminaire.entityName like 'lumin%'
```

  Using aliases is highly recommended to be explicit, increase readability, and avoid any ambiguities in your queries.

WHERE:

- The supported logical operators are AND, NOT, and OR.
- The supported comparison operators are <, <=, >, =>,=, and !=.
- Use the `IN` keyword if you want to specify multiple OR conditions on the same field.
- Filter on an entity, component or property field:

```
FROM EntityGraph MATCH (e), e.components AS c, c.properties AS p
WHERE e.entityName = 'room_0'
AND c.componentTypeId = 'com.example.query.construction.room',
AND p.propertyName = 'roomFunction'
AND NOT p.propertyValue = 'meeting'
```

```
OR p.propertyValue = 'office'
```

- Filter on the `configuration` property. Here `unit` is the key in the configuration map and `Celsius` is the value.

```
WHERE p.definition.configuration.unit = 'Celsius'
```

- Check if a map property contains a given key and value:

```
WHERE p.propertyValue.length = 20.0
```

- Check if a map property contains a given key:

```
WHERE NOT p.propertyValue.length IS MISSING
```

- Check if a list property contains a given value:

```
WHERE 10.0 IN p.propertyValue
```

- Use the `lower()` function for case insensitive comparisons. By default, all comparisons are case sensitive.

```
WHERE lower(p.propertyValue) = 'meeting'
```

LIKE:

Useful if you do not know the exact value for a field and can perform full text search on the specified field. `%` represents zero or more.

```
WHERE e.entityName LIKE '%room%'
```

- Infix search: `%room%`
- Prefix search: `room%`
- Suffix search: `%room`
- If you have '%' in your values, then put an escape character in the `LIKE` and specify the escape character with ESCAPE.

```
WHERE e.entityName LIKE 'room\%' ESCAPE '\'
```

DISTINCT:

```
SELECT DISTINCT c.componentTypeId
FROM EntityGraph MATCH (e), e.components AS c
```

- The `DISTINCT` keyword eliminates duplicates from the final result.

  `DISTINCT` is not supported on complex data types.

COUNT

```
SELECT COUNT(e), COUNT(c.componentTypeId)
FROM EntityGraph MATCH (e), e.components AS c
```

- The `COUNT` keyword computes the number of items in a query result.
- `COUNT` is not supported on nested complex fields and graph pattern fields.
- `COUNT` aggregation is not supported with `DISTINCT` and nested queries.

  For example, `COUNT(DISTINCT e.entityId)` is not supported.

PATH

The following pattern projections are supported in querying using path projection:

- Variable hop queries

  ```
  SELECT p FROM EntityGraph MATCH p = (a)-[]->{1, 3}(b)
  ```

  This query matches and projects nodes metadata of any patterns with outgoing edges from node *a* within 1 to 3 hops.

- Fixed hop queries

  ```
  SELECT p FROM EntityGraph MATCH p = (a)-[]->(b)<-[]-(c)
  ```

  This query matches and projects metadata of entities and incoming edges to *b*.

- Undirected queries

  ```
  SELECT p FROM EntityGraph MATCH p = (a)-[]-(b)-[]-(c)
  ```

This query matches and projects the metadata of nodes in 1 hop patterns connecting *a* and *c* via *b*.


```
{
    "columnDescriptions": [
        {
            "name": "path",
            "type": "PATH"
        }
    ],
    "rows": [
        {
            "rowData": [
                {
                    "path": [
                        {
                            "entityId": "a",
                            "entityName": "a"
                        },
                        {
                            "relationshipName": "a-to-b-relation",
                            "sourceEntityId": "a",
                            "targetEntityId": "b"
                        },
                        {
                            "entityId": "b",
                            "entityName": "b"
                        }
                    ]
                }
            ]
        },
        {
            "rowData": [
                {
                    "path": [
                        {
                            "entityId": "b",
                            "entityName": "b"
                        },
                        {
                            "relationshipName": "b-to-c-relation",
```

```
                            "sourceEntityId": "b",
                            "targetEntityId": "c"
                        },
                        {
                            "entityId": "c",
                            "entityName": "c"
                        }
                    ]
                }
            ]
        }
    ]
}
```

This PATH query response comprises of only metadata that identifies all the nodes and edges of each path/pattern between *a* and *c* via *b*.

LIMIT and **OFFSET**:

```
SELECT e.entityName
FROM EntityGraph MATCH (e)
WHERE e.entityName LIKE 'room_%'
LIMIT 10
OFFSET 5
```

LIMIT specifies the number of results to be returned in the query, and OFFSET specifies the number of results to skip.

LIMIT and **maxResults**:

The following example shows a query that returns 500 results in total, but only displays 50 at a time per API call. This pattern can be used where you need to limit the amount of displayed results, for example if you are only able to display 50 results in a UI.

```
aws iottwinmaker execute-query \
--workspace-id exampleWorkspace \
--query-statement "SELECT e FROM EntityGraph MATCH (e) LIMIT 500"\
--max-results 50
```

- The LIMIT keyword affects the query and limits the resulting rows. If you need to control the number of results returned per API call without limiting the total number of returned results, use LIMIT.

- `max-results` is an optional parameter for the ExecuteQuery API action. `max-results` only applies to the API and how results are read within the bounds of the above query.

  Using `max-results` in a query allows you to reduce the number of displayed results without limiting the actual number of returned results.

The query below iterates through the next page of results. This query uses the `ExecuteQuery` API call to return rows 51-100, where the next page of results is specified by the `next-token`– in this case the token is: `"H7kyGmvK376L"`.

```
aws iottwinmaker execute-query \
--workspace-id exampleWorkspace \
--query-statement "SELECT e FROM EntityGraph MATCH (e) LIMIT 500"\
--max-results 50
--next-token "H7kyGmvK376L"
```

- The `next-token` string specifies the next page of results. For more information, see the ExecuteQuery API action.

AWS IoT TwinMaker knowledge graph query has the following limits:

Limit Name	Quota	Adjustable
Query execution timeout	10 seconds	No
Maximum number of hops	10	Yes
Maximum number of self `JOIN`s	20	Yes
Maximum number of projected fields	20	Yes
Maximum number of conditional expressions (`AND`, `OR`, `NOT`)	10	Yes

Limit Name	Quota	Adjustable
Maximum length of a `LIKE` expression pattern (including wildcards and escapes)	20	Yes
Maximum number of items that can be specified in an `IN` clause	10	Yes
Maximum value for `OFFSET`	3000	Yes
Maximum value for `LIMIT`	3000	Yes
Maximum value for traversals (`OFFSET` + `LIMIT`)	3000	Yes

Asset synchronization with AWS IoT SiteWise

AWS IoT TwinMaker supports asset synchronization (asset sync) for your AWS IoT SiteWise assets and asset models. Using the AWS IoT SiteWise component type, asset sync takes existing AWS IoT SiteWise assets and asset models and converts these resources into AWS IoT TwinMaker entities, components, and component types. The following sections walk you through how to configure asset sync and which AWS IoT SiteWise assets and asset models can be synced to your AWS IoT TwinMaker workspace.

Topics

Using asset sync with AWS IoT SiteWise

This topic shows you how to turn on and configure AWS IoT SiteWise asset sync. Follow the appropriate procedures based on which type of workspace you're using.

> **Important**
>
> See the section called “Differences between custom and default workspaces” for information about the differences between the custom and default workspaces.

Topics

Using a custom workspace

Review these prerequisites before turning on asset sync.

Prerequisites

Before using AWS IoT SiteWise, the following must be completed:

- You have an AWS IoT TwinMaker workspace.
- You have assets and asset models in AWS IoT SiteWise. For more information, see Creating asset models.
- An existing IAM role with read permissions for the following AWS IoT SiteWise actions:
  - `ListAssets`
  - `ListAssetModels`
  - `DescribeAsset`
  - `DescribeAssetModel`
- The IAM role must have the following write permissions for AWS IoT TwinMaker:
  - `CreateEntity`
  - `UpdateEntity`
  - `DeleteEntity`
  - `CreateComponentType`
  - `UpdateComponentType`
  - `DeleteComponentType`
  - `ListEntities`
  - `GetEntity`
  - `ListComponentTypes`

  Use the following IAM role as a template for the required role:

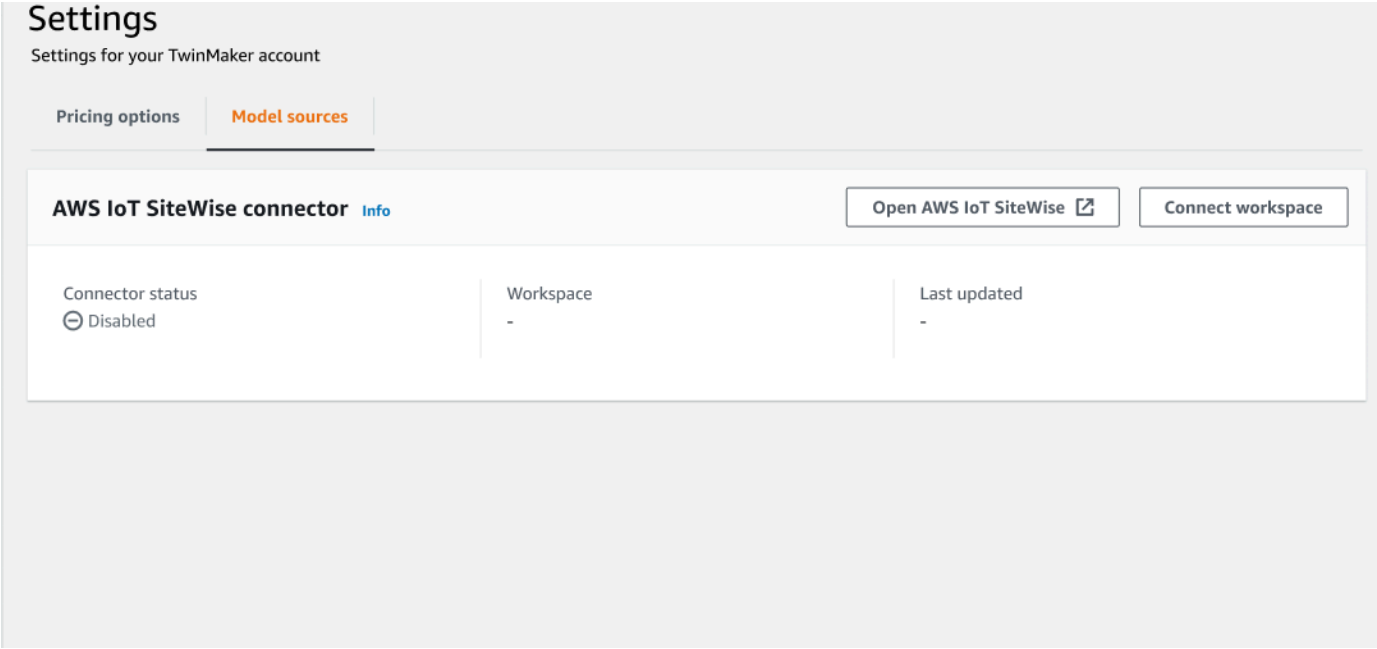
```
// trust relationships
 {
    {
        "Version": "2012-10-17",
        "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
            "Service": [
```

```
            "iottwinmaker.amazonaws.com"
            ]
        },
            "Action": "sts:AssumeRole"
        }
    ]
}

// permissions - replace ACCOUNT_ID, REGION, WORKSPACE_ID with actual values
{
    "Version": "2012-10-17",
    "Statement": [{
            "Sid": "SiteWiseAssetReadAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribeAsset"
            ],
            "Resource": [
                "arn:aws:iotsitewise:REGION:ACCOUNT_ID:asset/*"
            ]
        },
        {
            "Sid": "SiteWiseAssetModelReadAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribeAssetModel"
            ],
            "Resource": [
                "arn:aws:iotsitewise:REGION:ACCOUNT_ID:asset-model/*"
            ]
        },
        {
            "Sid": "SiteWiseAssetModelAndAssetListAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:ListAssets",
                "iotsitewise:ListAssetModels"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "TwinMakerAccess",
```

```
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:GetEntity",
                "iottwinmaker:CreateEntity",
                "iottwinmaker:UpdateEntity",
                "iottwinmaker:DeleteEntity",
                "iottwinmaker:ListEntities",
                "iottwinmaker:GetComponentType",
                "iottwinmaker:CreateComponentType",
                "iottwinmaker:UpdateComponentType",
                "iottwinmaker:DeleteComponentType",
                "iottwinmaker:ListComponentTypes"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:REGION:ACCOUNT_ID:workspace/WORKSPACE_ID",
                "arn:aws:iottwinmaker:REGION:ACCOUNT_ID:workspace/WORKSPACE_ID/*"

            ]
        }
    ]
}
```

Use the following procedure to turn on and configure AWS IoT SiteWise asset sync.

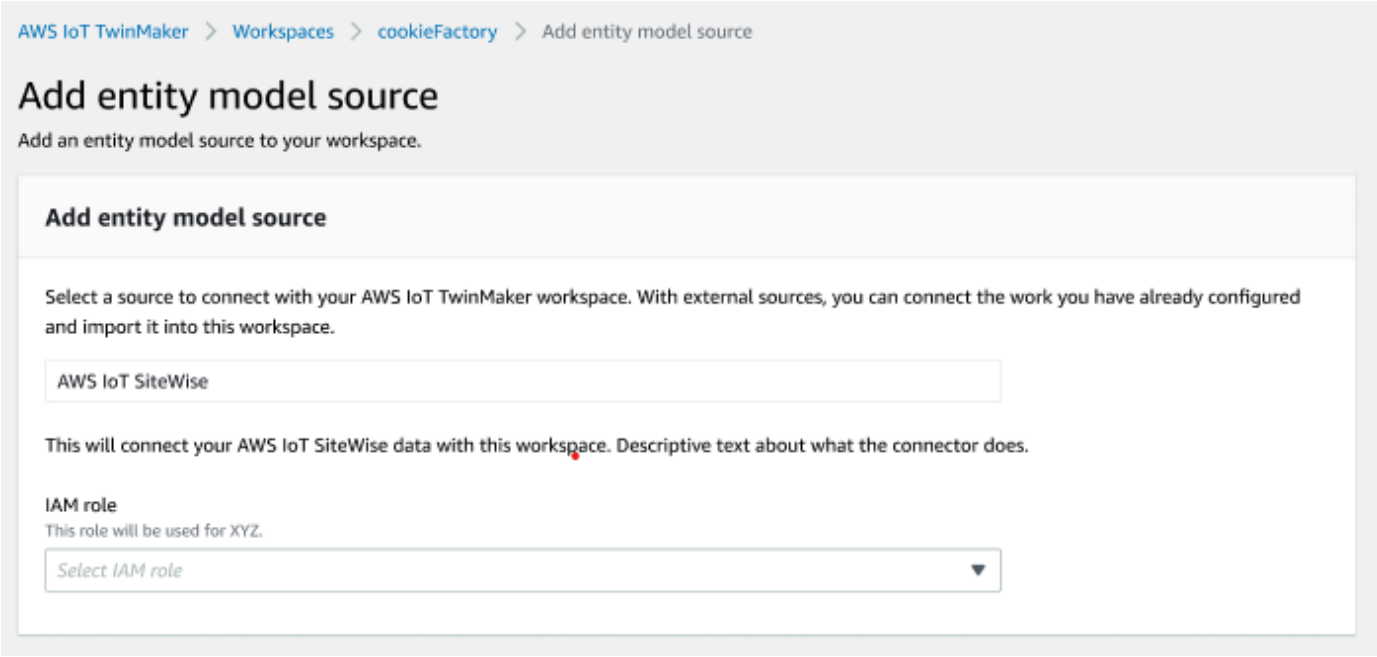
1. In the AWS IoT TwinMaker console, navigate to the **Settings** page.
2. Open the **Model sources** tab.



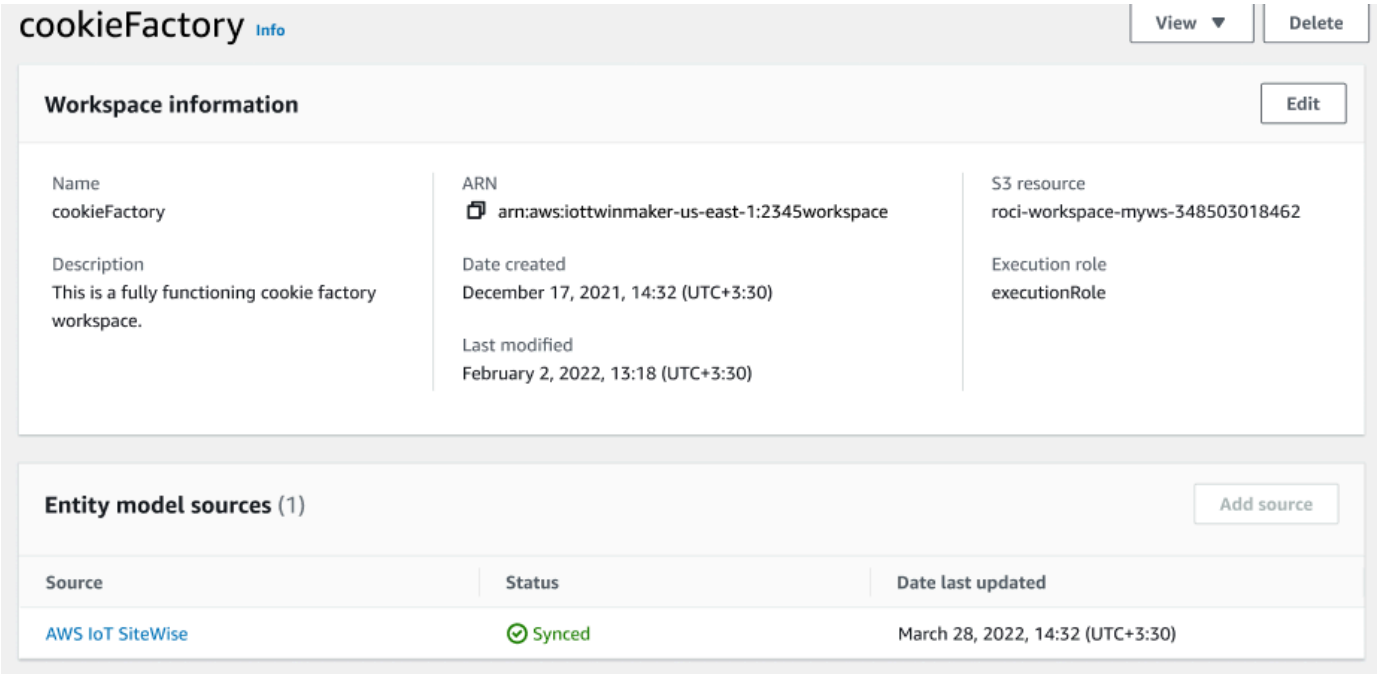
3. Choose **Connect workspace** to link your AWS IoT TwinMaker workspace to your AWS IoT SiteWise assets.

> **Note**
>
> You can only use asset sync with a single AWS IoT TwinMaker workspace. You must disconnect the sync from one workspace and connect to another workspace to if you wish to sync in a different workspace.

4. Next, navigate to the workspace in which you want to use asset sync.
5. Choose **Add sources**. This opens the **Add entity model source** page.



6. On the **Add entity model source** page, confirm that the source field displays **AWS IoT SiteWise**. Select the IAM role you created as a prerequisite for the **IAM role**.
7. You have now turned on AWS IoT SiteWise asset sync. You should see a conformation banner appear at the top of the selected **Workspace** page confirming that asset sync is active. You should also now see a sync source listed in the **Entity model sources** section.



Using the IoTSiteWiseDefaultWorkspace

When you opt in to the AWS IoT SiteWiseAWS IoT TwinMaker integration, a default workspace named `IoTSiteWiseDefaultWorkspace` is created and automatically synced with AWS IoT SiteWise.

You can also use the AWS IoT TwinMaker `CreateWorkspace` API to create a workspace named `IoTSiteWiseDefaultWorkspace`.

Prerequisites

Before creating `IoTSiteWiseDefaultWorkspace`, make sure you have done the following:

- Create an AWS IoT TwinMaker service-linked role. See Using service-linked roles for AWS IoT TwinMaker for more information.
- Open the IAM console at https://console.aws.amazon.com/iam/.

  Review the role or user and verify that it has permission to `iotsitewise:EnableSiteWiseIntegration`.

  If needed, add permission to the role or user:

  JSON

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

Differences between custom and default workspaces

> **Important**
>
> New AWS IoT SiteWise features, such as `CompositionModel`, are only available in `IoTSiteWiseDefaultWorkspace`. We encourage you to use a default workspace instead of custom workspace.

When using the `IoTSiteWiseDefaultWorkspace`, there are a few notable differences from using a custom workspace with asset sync.

- When you create a default workspace, the Amazon S3 location and IAM role are optional.

  > **Note**
  >
  > You can use `UpdateWorkspace` to provide the Amazon S3 location and IAM role.

- The `IoTSiteWiseDefaultWorkspace` doesn't have a resource count limit to sync AWS IoT SiteWise resources to AWS IoT TwinMaker.
- When you sync resources from AWS IoT SiteWise, their `SyncSource` will be `SITEWISE_MANAGED`. This includes `Entities` and `ComponentTypes`.
- New AWS IoT SiteWise features, such as `CompositionModel` are only available in the `IoTSiteWiseDefaultWorkspace`.

There are a few limitations specific to `IoTSiteWiseDefaultWorkspace`, they are:

- The default workspace can't be deleted.
- To delete resources, you must delete the AWS IoT SiteWise resources first, then the corresponding resources in AWS IoT TwinMaker are deleted.

Resources synced from AWS IoT SiteWise

This topic lists which assets you can sync from AWS IoT SiteWise to your AWS IoT TwinMaker workspace.

> **Important**
>
> See Differences between custom and default workspaces for information about the differences between the custom and default workspaces.

Custom and default workspaces

The following resources are synced and available in **both** custom and default workspaces:

Asset Models

AWS IoT TwinMaker creates a new component type for each asset model in AWS IoT SiteWise.

- The component `TypeId` for the asset model will use one of the following patterns:
  - **Custom workspace -** `iotsitewise.assetmodel:`*`assetModelId`*
  - **Default workspace -** *`assetModelId`*
- Each property in the asset model is a new property in the component type, with one of the following naming patterns:
  - **Custom workspace -** `Property_`*`propertyId`*
  - **Default workspace -** *`propertyId`*

  The property name in AWS IoT SiteWise is stored as the `displayName` in the property definition.
- Each hierarchy in the asset model is a new property of type `LIST` and the `nestedType` is `RELATIONSHIP` in the component type. The hierarchy is mapped to the property with a name prefixed by one of the following:
  - **Custom workspace -** *`Hierarchy_hierarchyId`*
  - **Default workspace -** *`hierarchyId`*

Asset

AWS IoT TwinMaker creates a new entity for each asset in AWS IoT SiteWise.

- The `entityId` is the same as the `assetId` in AWS IoT SiteWise.
- These entities have a single component called `sitewiseBase`, which has the component type corresponding to the asset model for this asset.
- Any asset level overrides, such as setting property alias or unit of measure, are reflected in the entity in AWS IoT TwinMaker.

Default workspace only

The following assets are synced and **available in the default workspace only**, `IoTSiteWiseDefaultWorkspace`.

AssetModelComponents

AWS IoT TwinMaker creates a new component type for each `AssetModelComponents` in AWS IoT SiteWise.

- The component `TypeId` for the asset model uses the following pattern: `assetModelId`.
- Each property in the asset model is a new property in the component type, with the property name as `propertyId`. The property name in AWS IoT SiteWise is stored as the `displayName` in the property definition.
- Each hierarchy in the asset model is a new property of type `LIST` and the `nestedType` is `RELATIONSHIP` in the component type. The hierarchy is mapped to the property with a name prefixed by `hierarchyId`.

AssetModelCompositeModel

AWS IoT TwinMaker creates a new component type for each `AssetModelCompositeModel` in AWS IoT SiteWise.

- The component `TypeId` for the asset model uses the following pattern: `assetModelId_assetModelCompositeModelId`.
- Each property in the asset model is a new property in the component type, with the property name as `propertyId`. The property name in AWS IoT SiteWise is stored as the `displayName` in the property definition.

AssetCompositeModels

AWS IoT TwinMaker creates a new composite component for each `AssetCompositeModel` in AWS IoT SiteWise.

- The `componentName` is the same as the `assetModelCompositeModelId` in AWS IoT SiteWise.

Resources not synced

The following resources are not synced:

Non-synced assets and asset models

- Alarm models will be synced as compositeModels, but corresponding data in the asset related to alarms are not synced.
- AWS IoT SiteWise data streams are not synced. Only properties modeled in the asset model are synced.
- Property values for attributes, measurements, transforms, aggregates, and metadata calculation such as formula and window are not synced. Only the metadata about the properties, such as alias, unit of measure, and data type are synced. The values can be queried using the regular AWS IoT TwinMaker data connector API, GetPropertyValueHistory.

Use synced entities and component types in AWS IoT TwinMaker

Once assets are synced from AWS IoT SiteWise, the synced component types are read only in AWS IoT TwinMaker. Any update or delete action must be done in AWS IoT SiteWise, and those changes are synced to AWS IoT TwinMaker if the syncJob is still active.

The synced entities and the AWS IoT SiteWise base component are also read only in AWS IoT TwinMaker. You can add additional non-synced components to the synced entity, as long as no entity-level attributes such as the description or `entityName` are updated.

Some restrictions apply to how you can interact with synced entities. You can't create child entities under a synced entity in the synced entity's hierarchy. Additionally, you can't create non-synced component types that extend from a synced component type.

> **Note**
>
> Additional components are deleted along with the entity if the asset is deleted in AWS IoT SiteWise or if you delete the sync job.

You can use these synced entities in Grafana dashboards and add them as tags in the scene composer like regular entities. You can also issue knowledge graph queries for these synced entities.

> **Note**
>
> Synced entities without modification are not charged, but you are charged for those entities if changes have been made in AWS IoT TwinMaker. For example, if you add a non-

synced component to a synced entity, that entity is now charged in AWS IoT TwinMaker. For more information, see AWS IoT TwinMaker Pricing.

Analyze sync status and errors

This topic provides guidance on how to analyze sync errors and statuses.

Important

See the section called "Differences between custom and default workspaces" for information about the differences between the custom and default workspaces.

Sync job statuses

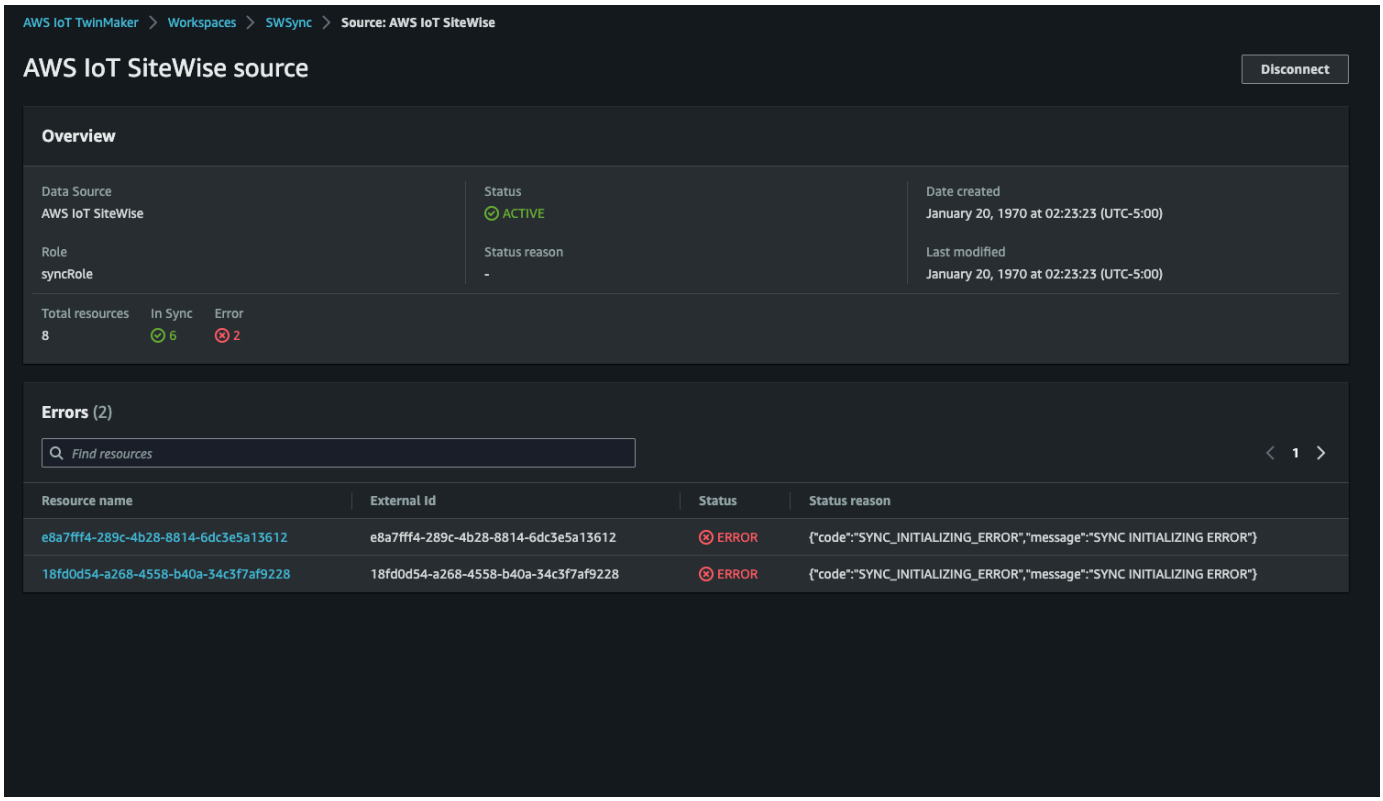
A sync job has one of the following statuses depending on its state.

- The sync job `CREATING` state means the job is checking for permissions and loading data from AWS IoT SiteWise to prepare the sync.
- The sync job `INITIALIZING` state means all the existing resources in AWS IoT SiteWise are synced to AWS IoT TwinMaker. This step can take longer to complete if the user has a large number of assets and asset models in AWS IoT SiteWise. You can monitor the number of resources that have been synced by checking on the sync job in the AWS IoT TwinMaker console, or by calling the `ListSyncResources` API.
- The sync job `ACTIVE` state means the initialization step is done. The job is now ready to sync any new updates from AWS IoT SiteWise.
- The sync job `ERROR` state indicates an error with any of the preceding states. Review the error message. There may be an issue with the IAM role setup. If you want to use a new IAM role, delete the sync job that had the error and create a new one with the new role.

Sync errors appear in the model source page, which is accessed from the **Entity model sources** table in your workspace. The model source page displays a list of resources that failed to sync. Most errors are automatically retried by the sync job, but if the resource requires an action, then it remains in the `ERROR` state. You can also obtain a list of errors by using the ListSyncResources API.

To see all the listed errors for the current source, use the following procedure.

1. Navigate to your workspace in the AWS IoT TwinMaker console.
2. Select the AWS IoT SiteWise source listed in the **Entity model sources** modal to open the asset sync details page.



3. As shown in the preceding screenshot, any resources with persisting errors are listed in the **Errors** table. You can use this table to track down and fix errors related to specific resources.

Possible errors include the following:

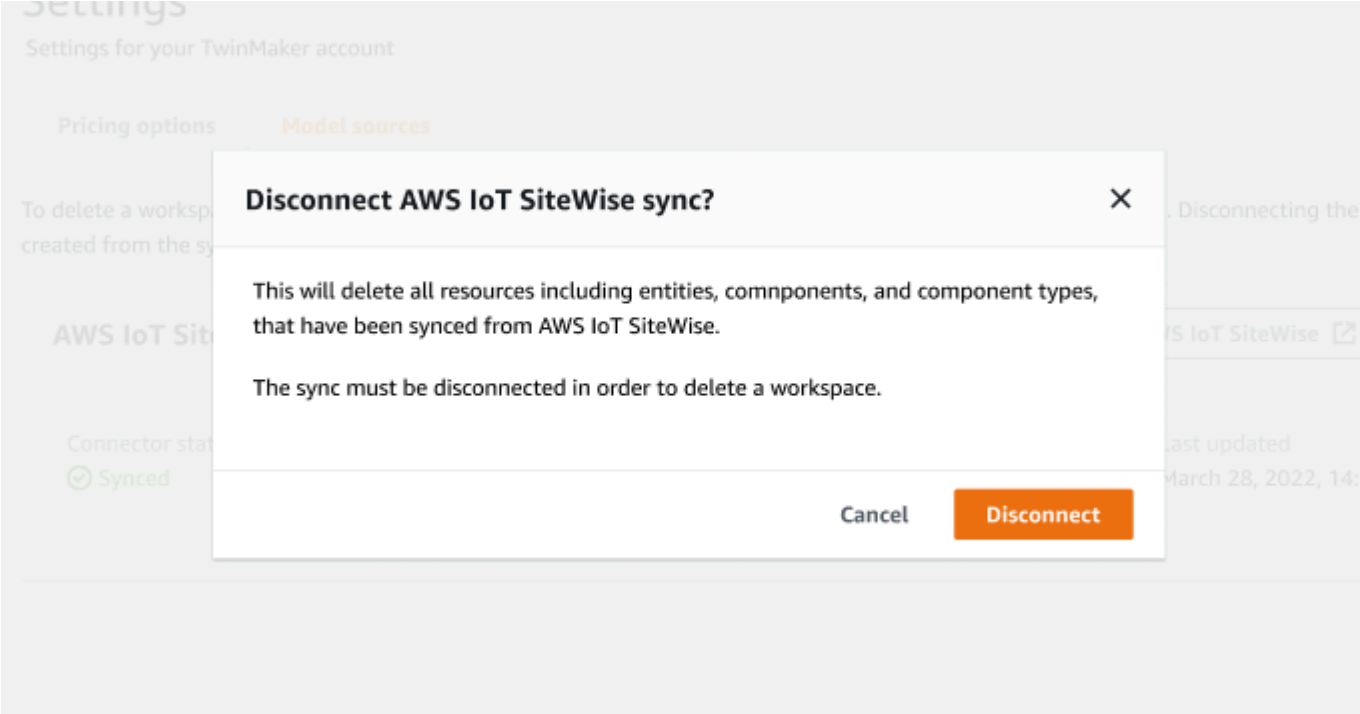
- While AWS IoT SiteWise supports duplicate asset names, AWS IoT TwinMaker only supports them at the R00T level, not under the same parent entity. If you have two assets with the same name under a parent entity in AWS IoT SiteWise, one of them fails to sync. To fix this error, either delete one of the assets or move one under a different parent asset in AWS IoT SiteWise before you sync.
- If you already have an entity with the same ID as the AWS IoT SiteWise asset ID, that asset fails to sync until you delete the existing entity.

Delete a sync job

Use the following procedure to delete a sync job.

> **Important**
>
> See the section called "Differences between custom and default workspaces" for information about the differences between the custom and default workspaces.

1. Navigate to the AWS IoT TwinMaker console.
2. Open the workspace from which you wish to delete the sync job.
3. Under **Entity model sources**, select the AWS IoT SiteWise source to open the source details page.
4. To stop the sync job, choose **Disconnect**. Confirm your choice to fully delete the sync job.



Once a sync job is deleted, you can create the sync job again in the same or a different workspace.

You can't delete a workspace if there are any sync jobs in that workspace. Delete the sync jobs first before deleting a workspace.

If there are any errors during the deletion of the sync job, the sync job remains in the `DELETING` state and is automatically retried. You can now manually delete any of the synced entities or component types if there is any error related to deleting a resource.

> **Note**
>
> Any resources that were synced from AWS IoT SiteWise are deleted first, then the sync job itself is deleted.

Asset sync limits

> **Important**
>
> See the section called "Differences between custom and default workspaces" for information about the differences between the custom and default workspaces.

Because the AWS IoT SiteWise quotas are higher than the default AWS IoT TwinMaker quotas, we are increasing the following limits for entities and component types synced from AWS IoT SiteWise.

- 1000 synced component types in a workspace, since it can only sync 1000 asset models from AWS IoT SiteWise.
- 100,000 synced entities in a workspace, since it can only sync 100,000 assets from AWS IoT SiteWise.
- 2000 maximum child entities per parent entity. It syncs 2000 child assets per single parent asset.

  > **Note**
  >
  > The GetEntity API only returns the first 50 child entities for a hierarchy property, but you can use the GetPropertyValue API to paginate and retrieve the list of all child entities.

- 600 properties per synced component from AWS IoT SiteWise, which can sync asset models with 600 total properties and hierarchies.

Note

These limits are only applicable for the synced entities. Request a quota increase if you need these limits increased for non-synced resources.

AWS IoT TwinMaker Grafana dashboard integration

AWS IoT TwinMaker supports Grafana integration through an application plugin. Use Grafana version 10.4.0 and later versions to interact with your digital twin application. The AWS IoT TwinMaker plugin provides custom panels, dashboard templates, and a datasource to connect to your digital twin data.

For more information about how to onboard with Grafana and set up permissions for your dashboard, see the following topics:

Topics

- CORS configuration for Grafana scene viewer
- Setting up your Grafana environment
- Creating a dashboard IAM role
- Creating an AWS IoT TwinMaker video player policy

> **Note**
>
> You need to modify CORS (cross-origin resource sharing) configuration of the Amazon S3 bucket to allow the Grafana user interface to load resources from the bucket. For the instructions, see CORS configuration for Grafana scene viewer.

For more information about the AWS IoT TwinMaker Grafana plugin, see the AWS IoT TwinMaker App documentation.

For more information about the key components of the Grafana plugin, see the following:

- AWS IoT TwinMaker datasource
- Dashboard templates
- Scene Viewer panel
- Video Player panel

CORS configuration for Grafana scene viewer

The AWS IoT TwinMaker Grafana plugin requires a CORS (cross-origin resource sharing) configuration, which allows the Grafana user interface to load resources from the Amazon S3 bucket. Without the CORS configuration, you will receive an error message as "Load 3D Scene failed with Network Failure" on the Scene viewer since the Grafana domain can't access the resources in the Amazon S3 bucket.

To configure your Amazon S3 bucket with CORS, use the following steps:

1. Sign in to the IAM console and open the Amazon S3 console.
2. In the **Buckets** list, choose the name of the bucket that you use as your AWS IoT TwinMaker workspace's resource bucket.
3. Choose **Permissions**.
4. In the **Cross-origin resource sharing** section, select **Edit**, to open the CORS editor.
5. In the **CORS configuration editor** text box, type or copy and paste the following JSON CORS configuration by replacing the Grafana workspace domain *GRAFANA-WORKSPACE-DOMAIN* with your domain.

> **Note**
>
> You need to keep the asterisk * character at the beginning of the `"AllowedOrigins":` JSON element.

```
    [
{
    "AllowedHeaders": [
        "*"
    ],
    "AllowedMethods": [
        "GET",
        "PUT",
        "POST",
        "DELETE",
        "HEAD"
    ],
    "AllowedOrigins": [
```

```
            "*GRAFANA-WORKSPACE-DOMAIN"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

6. Select **Save changes** to finish the CORS configuration.

For more information on CORS with Amazon S3 buckets, see Using cross-origin resource sharing (CORS).

Setting up your Grafana environment

You can use Amazon Managed Grafana for a fully managed service, or set up a Grafana environment that you manage yourself. With Amazon Managed Grafana, you can quickly deploy, operate, and scale open source Grafana for your needs. Alternatively, you can set up your own infrastructure to manage Grafana servers.

For more information about both Grafana environment options, see the following topics:

- Amazon Managed Grafana
- Self-managed Grafana

Amazon Managed Grafana

Amazon Managed Grafana provides an AWS IoT TwinMaker plugin so you can quickly integrate AWS IoT TwinMaker with Grafana. Because Amazon Managed Grafana manages Grafana servers for you, you can visualize your data without having to build, package, or deploy any hardware or any other Grafana infrastructure. For more information about Amazon Managed Grafana, see What is Amazon Managed Grafana?.

> **Note**
>
> Amazon Managed Grafana currently supports version **1.3.1** of the AWS IoT TwinMaker Grafana plugin.

Amazon Managed Grafana prerequisites

To use AWS IoT TwinMaker in an Amazon Managed Grafana dashboard, first complete the following prerequisite:

- Create an AWS IoT TwinMaker workspace. For more information about creating workspaces, see Getting started with AWS IoT TwinMaker.

> **Note**
>
> When you first create an Amazon Managed Grafana workspace in the AWS Management Console, AWS IoT TwinMaker isn't listed. However, the plugin is already installed on all workspaces. You can find the AWS IoT TwinMaker plugin on the open source Grafana plugins list. You can find the AWS IoT TwinMaker datasource by choosing **Add a datasource** on the Datasources page.

When you create an Amazon Managed Grafana workspace, an IAM role is created automatically to manage the permissions for the Grafana instance. This is called the **Workspace IAM Role**. It's the authentication provider option you'll use to configure all AWS IoT TwinMaker datasources for Grafana. Amazon Managed Grafana doesn't support automatically adding permissions for AWS IoT TwinMaker, so you must set up these permissions manually. For more information about setting up manual permissions, see Creating a dashboard IAM role.

Self-managed Grafana

You can choose to host your own infrastructure to run Grafana. For information about running Grafana locally on your machine, see Install Grafana. The AWS IoT TwinMaker plugin is available on the public Grafana catalog. For information about installing this plugin in your Grafana environment, see AWS IoT TwinMaker App.

When you run Grafana locally you can't easily share dashboards or provide access to multiple users. For a scripted quick start guide about sharing dashboards using local Grafana, see AWS IoT TwinMaker samples repository. This resource walks you through hosting a Grafana environment on Cloud9 and Amazon EC2 on a public endpoint.

You must determine which authentication provider you'll use for configuring TwinMaker datasources. You configure the credentials for the environment based on the default credentials

chain (see Using the Default Credential Provider Chain). The default credentials can be the permanent credentials of any user or role. For example, if you're running Grafana on Amazon EC2 , the default credentials chain has access to the Amazon EC2 execution role, which would then be your authentication provider. The IAM Amazon Resource Name (ARN) of the authentication provider is required in the steps to Creating a dashboard IAM role.

Creating a dashboard IAM role

With AWS IoT TwinMaker, you can control data access on your Grafana dashboards. Grafana dashboard users should have different permission scopes to view data, and in some cases, write data. For example, an alarm operator might not have permission to view videos, while an admin has permission for all resources. Grafana defines the permissions through datasources, where credentials and an IAM role are provided. The AWS IoT TwinMaker datasource fetches AWS credentials with permissions for that role. If an IAM role isn't provided, Grafana uses the scope of the credentials, which can't be reduced by AWS IoT TwinMaker.

To use your AWS IoT TwinMaker dashboards in Grafana, you create an IAM role and attach policies. You can use the following templates to help you create these policies.

Create an IAM policy

Create an IAM policy called *YourWorkspaceId*`DashboardPolicy` in the IAM Console. This policy gives your workspaces access to Amazon S3 bucket and AWS IoT TwinMaker resources. You can also decide to use AWS IoT Greengrass Edge Connector for Amazon Kinesis Video Streams, which requires permissions for the Kinesis Video Streams and AWS IoT SiteWise assets configured for the component. To fit your use case, choose one of the following policy templates.

1. No video permissions policy

If you don't want to use the Grafana Video Player panel, create the policy using the following template.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
                "iottwinmaker:List*"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId",
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        }
    ]
}
```

An Amazon S3 bucket is created for each workspace. It contains the 3D models and scenes to view on a dashboard. The SceneViewer panel loads items from this bucket.

2. Scoped down video permissions policy

To limit access on the Video Player panel in Grafana, group your AWS IoT Greengrass Edge Connector for Amazon Kinesis Video Streams resources by tags. For more information about scoping down permissions for your video resources, see Creating an AWS IoT TwinMaker video player policy.

3. All video permissions

If you don't want to group your videos, you can make them all accessible from the Grafana Video Player. Anyone with access to a Grafana workspace is able to play video for any stream in your account, and have read only access to any AWS IoT SiteWise asset. This includes any resources that are created in the future.

Create the policy with the following template:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucketName/*",
                "arn:aws:s3:::bucketName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
                "iottwinmaker:List*"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId",
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetHLSStreamingSessionURL"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iotsitewise:GetAssetPropertyValue",
                "iotsitewise:GetInterpolatedAssetPropertyValues"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:BatchPutAssetPropertyValue"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
                }
            }
        }
    ]
}
```

This policy template provides the following permissions:

- Read only access to an S3 bucket to load a scene.
- Read only access to AWS IoT TwinMaker for all entities and components in a workspace.
- Read only access to stream all Kinesis Video Streams videos in your account.
- Read only access to the property value history of all AWS IoT SiteWise assets in your account.
- Data ingestion into any property of a AWS IoT SiteWise asset tagged with the key `EdgeConnectorForKVS` and the value `workspaceId`.

Tagging your camera AWS IoT SiteWise asset request video upload from edge

Using the Video Player in Grafana , users can manually request that video is uploaded from the edge cache to Kinesis Video Streams. You can turn on this feature for any AWS IoT SiteWise asset that's associated with your AWS IoT Greengrass Edge Connector for Amazon Kinesis Video Streams and that is tagged with the key `EdgeConnectorForKVS`.

The tag value can be a list of workspaceIds delimited by any of the following characters: `. : + = @ _ / -`. For example, if you want to use an AWS IoT SiteWise asset associated with an AWS IoT Greengrass Edge Connector for Amazon Kinesis Video Streams across AWS IoT TwinMaker workspaces, you can use a tag that follows this pattern: `WorkspaceA/WorkspaceB/WorkspaceC`. The Grafana plugin enforces that the AWS IoT TwinMaker workspaceId is used to group AWS IoT SiteWise asset data ingestion.

Add more permissions to your dashboard policy

The AWS IoT TwinMaker Grafana plugin uses your authentication provider to call AssumeRole on the dashboard role you create. Internally, the plugin restricts the highest scope of permissions you have access to by using a session policy in the AssumeRole call. For more information about session policies, see Session policies.

This is the maximum permissive policy you can have on your dashboard role for an AWS IoT TwinMaker workspace:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucketName/*",
                "arn:aws:s3:::bucketName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
                "iottwinmaker:List*"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId",
```

```
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetHLSStreamingSessionURL"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:GetAssetPropertyValue",
                "iotsitewise:GetInterpolatedAssetPropertyValues"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:BatchPutAssetPropertyValue"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
                }
            }
        }
    ]
}
```

If you add statements that `Allow` more permissions, they won't work on the AWS IoT TwinMaker plugin. This is by design to ensure the minimum necessary permissions are used by the plugin.

However, you can scope down permissions further. For information, see Creating an AWS IoT TwinMaker video player policy.

Creating the Grafana Dashboard IAM role

In the IAM Console, create an IAM role called *YourWorkspaceId*DashboardRole. Attach the *YourWorkspaceId*DashboardPolicy to the role.

To edit the trust policy of the dashboard role, you must give permission for the Grafana authentication provider to call AssumeRole on the dashboard role. Update the trust policy with the following template:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "ARN of Grafana authentication provider"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For more information about creating a Grafana environment and finding your authentication provider, see Setting up your Grafana environment.

Creating an AWS IoT TwinMaker video player policy

The following is a policy template with all of the video permissions you need for the AWS IoT TwinMaker plugin in Grafana:

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
                "iottwinmaker:List*"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId",
                "arn:aws:iottwinmaker:us-
east-1:111122223333:workspace/workspaceId/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetHLSStreamingSessionURL"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:GetAssetPropertyValue",
                "iotsitewise:GetInterpolatedAssetPropertyValues"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotsitewise:BatchPutAssetPropertyValue"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
                }
            }
        }
    ]
}
```

For more information about the full policy, see the **All video permissions** policy template in the Create an IAM policy topic.

Scope down access to your resources

The Video Player panel in Grafana directly calls Kinesis Video Streams and IoT SiteWise to provide a complete video playback experience. To avoid unauthorized access to resources that aren't associated with your AWS IoT TwinMaker workspace, add conditions to the IAM policy for your workspace dashboard role.

Scope down GET permissions

You can scope down the access of your Amazon Kinesis Video Streams and AWS IoT SiteWise assets by tagging resources. You might have already tagged your AWS IoT SiteWise camera asset based on the AWS IoT TwinMaker workspaceId to enable the video upload request feature, see the Upload video from the edge topic. You can use the same tag key-value pair to limit GET access to AWS IoT SiteWise assets, and also to tag your Kinesis Video Streams the same way.

You can then add this condition to the kinesisvideo and iotsitewise statements in the *YourWorkspaceId*DashboardPolicy:

```
"Condition": {
  "StringLike": {
    "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
```

```
  }
}
```

Real-life use case: Grouping cameras

In this scenario, you have a large array of cameras monitoring the process of baking cookies in a factory. Batches of cookie batter are made in the Batter Room, batter is frozen in the Freezer Room, and cookies are baked in the Baking Room. There are cameras in each of these rooms with different teams of operators separately monitoring each process. You want each group of operators to be authorized for their respective room. When building a digital twin for the cookie factory, a single workspace is used, but the camera permissions need to be scoped by room.

You can achieve this permission separation by tagging groups of cameras based on their groupingId. In this scenario, the groupingIds are BatterRoom, FreezerRoom, and BakingRoom. The camera in each room is connected to Kinesis Video Streams and should have a tag with: Key = `EdgeConnectorForKVS`, Value = `BatterRoom`. The value can be a list of groupings delimited by any of the following characters: `. : + = @ _ / -`

To amend the *`YourWorkspaceId`*`DashboardPolicy`, use the following policy statements:

```
...,
{
  "Effect": "Allow",
  "Action": [
    "kinesisvideo:GetDataEndpoint",
    "kinesisvideo:GetHLSStreamingSessionURL"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:ResourceTag/EdgeConnectorForKVS": "*groupingId*"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "iotsitewise:GetAssetPropertyValue",
    "iotsitewise:GetInterpolatedAssetPropertyValues"
  ],
  "Resource": "*",
  "Condition": {
```

```
    "StringLike": {
      "aws:ResourceTag/EdgeConnectorForKVS": "*groupingId*"
    }
  }
},
...
```

These statements restrict streaming video playback and AWS IoT SiteWise property history access to specific resources in a grouping. The *groupingId* is defined by your use case. In the our scenario, it would be the roomId.

Scope down AWS IoT SiteWise BatchPutAssetPropertyValue permission

Providing this permission turns on the video upload request feature in the Video Player. When you upload video, you can specify a time range and submit the request from by choosing **Submit** on the panel on the Grafana dashboard.

To give iotsitewise:BatchPutAssetPropertyValue permissions, use the default policy:

```
...,
{
  "Effect": "Allow",
  "Action": [
    "iotsitewise:BatchPutAssetPropertyValue"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
    }
  }
},
...
```

By using this policy, users can call BatchPutAssetPropertyValue for any property on the AWS IoT SiteWise camera asset. You can restrict authorization for a specific propertyId by specifying it in the statement's condition.

```
{
  ...
  "Condition": {
```

```
    "StringEquals": {
      "iotsitewise:propertyId": "propertyId"
    }
  }
  ...
}
```

The Video Player panel in Grafana ingests data into the measurement property, named VideoUploadRequest, to initiate the uploading of video from the edge cache to Kinesis Video Streams. Find the propertyId of this property in the AWS IoT SiteWise Console. To amend the *YourWorkspaceId*DashboardPolicy, use the following policy statement:

```
...,
{
  "Effect": "Allow",
  "Action": [
    "iotsitewise:BatchPutAssetPropertyValue"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:ResourceTag/EdgeConnectorForKVS": "*workspaceId*"
    },
    "StringEquals": {
      "iotsitewise:propertyId": "VideoUploadRequestPropertyId"
    }
  }
},
...
```

This statement restricts ingesting data to a specific property of your tagged AWS IoT SiteWise camera asset. For more information, see How AWS IoT SiteWise works with IAM.

Connect AWS IoT SiteWise Alarms to AWS IoT TwinMaker Grafana dashboards

> **Note**
>
> This feature is in **public preview** and is subject to change.

AWS IoT TwinMaker is able import AWS IoT SiteWise and Events alarms into AWS IoT TwinMaker components. This allows you to be able to query alarm status and configure alarm thresholds without implementing a custom data connector for AWS IoT SiteWise data migration. You can use the AWS IoT TwinMaker Grafana plugin to visualize the alarm status and configure the alarm threshold in Grafana, without making API calls to AWS IoT TwinMaker or interacting directly with AWS IoT SiteWise alarms.

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

AWS IoT SiteWise alarm configuration prerequisites

Before creating alarms and integrating them into your Grafana dashboard, make sure you have reviewed the following prerequisites:

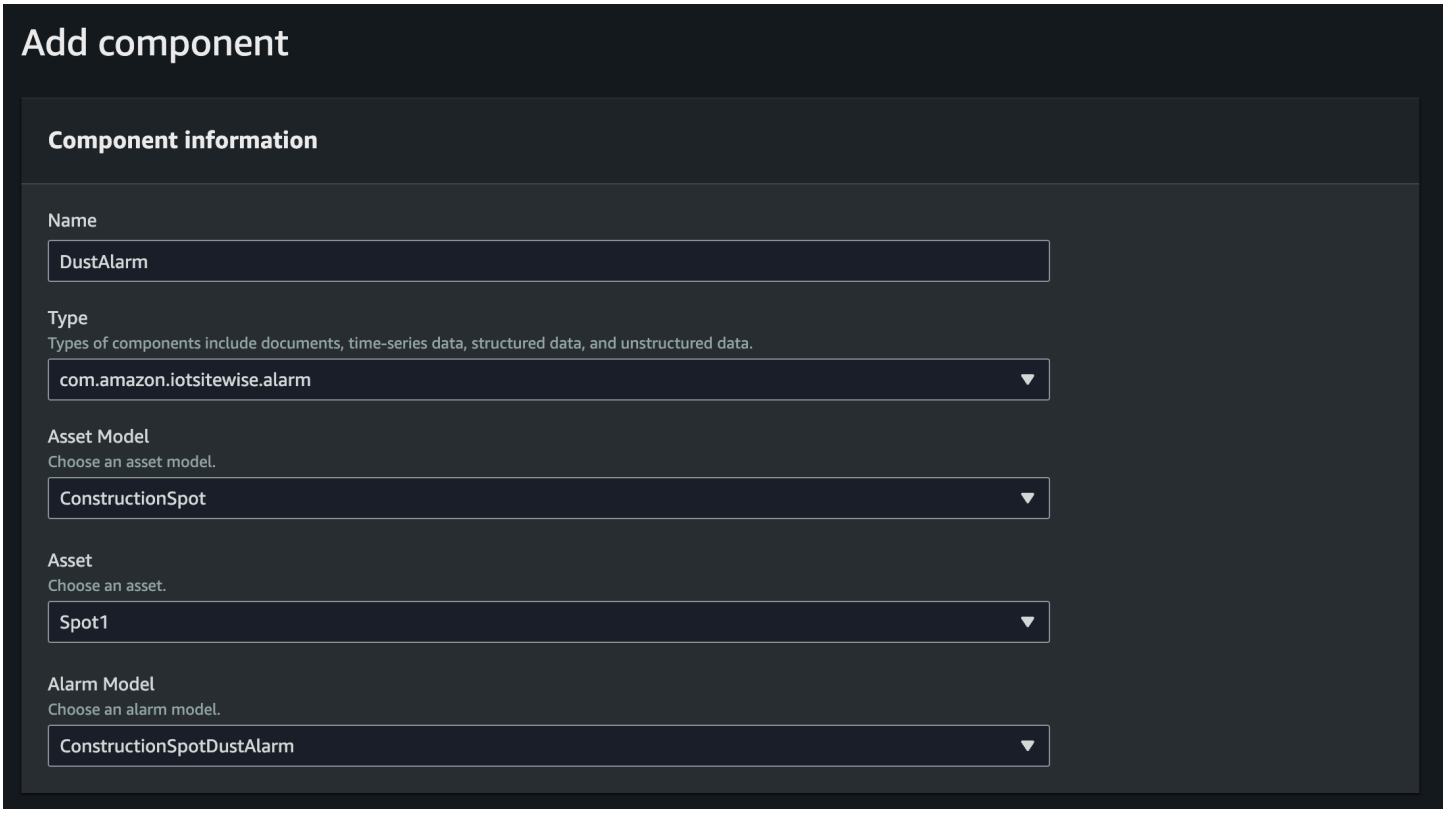
- Become familiar with AWS IoT SiteWise's model and asset system. For more information, see Creating asset models and Creating assets in the *AWS IoT SiteWise User Guide*.
- Become familiar with the IoT Events alarm models and how to attach them to an AWS IoT SiteWise model. For more information, see Defining AWS IoT Events alarms in the *AWS IoT SiteWise User Guide*.
- Integrate AWS IoT TwinMaker with Grafana so you can access your AWS IoT TwinMaker resources in Grafana. For more information see, AWS IoT TwinMaker Grafana dashboard integration.

Define the AWS IoT SiteWise alarm component IAM role

AWS IoT TwinMaker uses the workspace IAM role to query and configure the alarm threshold in Grafana. The following permissions are required in the AWS IoT TwinMaker workspace role, in order to interact with AWS IoT SiteWise alarms in Grafana:

```
{
"Effect": "Allow",
    "Action": [
        "iotevents:DescribeAlarmModel",
    ],
    "Resource": ["{IoTEventsAlarmModelArn}"]
},{
"Effect": "Allow",
    "Action": [
        "iotsitewise:BatchPutAssetPropertyValue"
    ],
    "Resource": ["{IoTSitewiseAssetArn}"]
}
```

In the AWS IoT TwinMaker console, create an entity that represents your AWS IoT SiteWise asset. Make sure you add a component for that entity using `com.amazon.iotsitewise.alarm` as the component type, and pick the corresponding asset and alarm models.



The above screenshot is example of creating this entity with the type `com.amazon.iotsitewise.alarm`.

When you create this component, AWS IoT TwinMaker automatically imports the related alarm properties from AWS IoT SiteWise and AWS IoT Events. You can the repeat this alarm component type pattern to create alarm components for all the assets needed in your workspace.

Query and update through the AWS IoT TwinMaker API

After creating alarm components, you can query the alarm status, threshold, and update alarm thresholds through the AWS IoT TwinMaker API.

Below is a sample request to query alarm status:

```
aws iottwinmaker get-property-value-history --cli-input-json \
'{
    "workspaceId": "{workspaceId}",
    "entityId": "{entityId}",
    "componentName": "{componentName}",
    "selectedProperties": ["alarm_status"],
```

```
    "startTime": "{startTimeIsoString}",
    "endTime": "{endTimeIsoString}"
}'
```

Below is a sample request to query the alarm threshold.

```
aws iottwinmaker get-property-value-history --cli-input-json \
'{
    "workspaceId": "{workspaceId}",
    "entityId": "{entityId}",
    "componentName": "{componentName}",
    "selectedProperties": ["alarm_threshold"],
    "startTime": "{startTimeIsoString}",
    "endTime": "{endTimeIsoString}"
}'
```

Below is a sample request to update the alarm threshold:

```
aws iottwinmaker batch-put-property-values --cli-input-json \
'{
    "workspaceId": "{workspaceId}",
    "entries": [
        {
            "entityPropertyReference": {
                "entityId": "{entityId}",
                "componentName": "{componentName}",
                "propertyName": "alarm_threshold"
            },
            "propertyValues": [
                {
                    "value": {
                        "doubleValue": "{newThreshold}"
                    },
                    "time": "{effectiveTimeIsoString}"
                }
            ]
        }
    ]
}'
```

Configure your Grafana dashboard for alarms

A second write enabled dashboard IAM role needs to be created , that is a normal role but with permission for the action `iottwinmaker:BatchPutPropertyValues` to add to the TwinMaker workspace arn like in the example below.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:Get*",
               "iottwinmaker:List*",
                "iottwinmaker:BatchPutPropertyValues"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iottwinmaker:ListWorkspaces",
            "Resource": "*"
        }
    ]
}
```

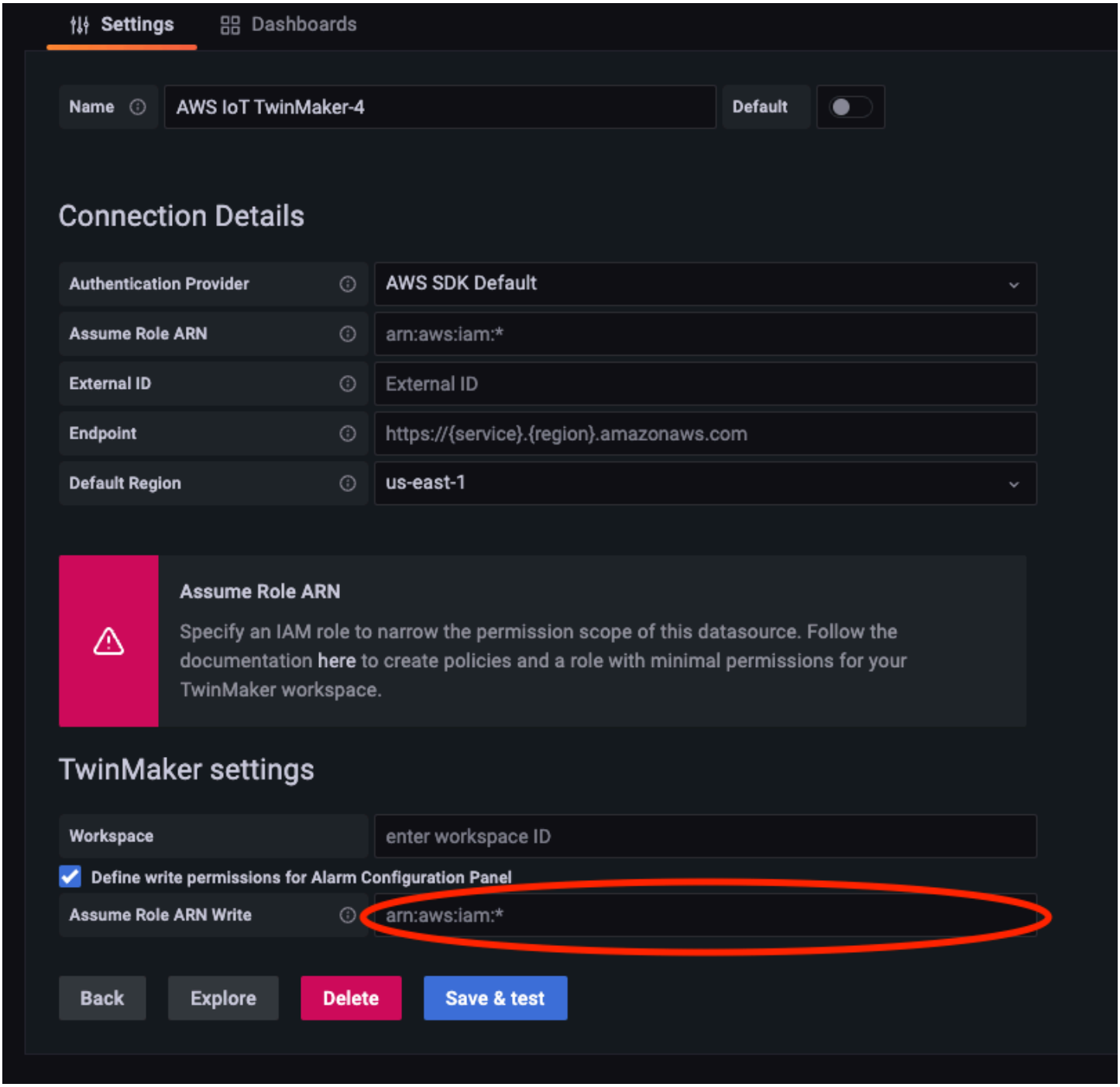
Alternatively you can add this statement at the end of your IAM role instead:

```
{
    "Effect": "Allow",
    "Action": [
        "iottwinmaker:BatchPutPropertyValues"
    ],
    "Resource": [
        "{workspaceArn}",
```

```
        "{workspaceArn}/*"
    ]
}
```

The datasource needs to have it's write arn set with the dashboard write role you created.

After you modify your IAM role, login into your Grafana dashboard to assume the updated role arn. Select the checkbox for **Define write permissions for Alarm Configuration Panel** and copy in the arn for the Write role.



Use Grafana dashboard for alarm visualization

Use the following procedure to add an alarm configuration panel to your dashboard and configure it: :

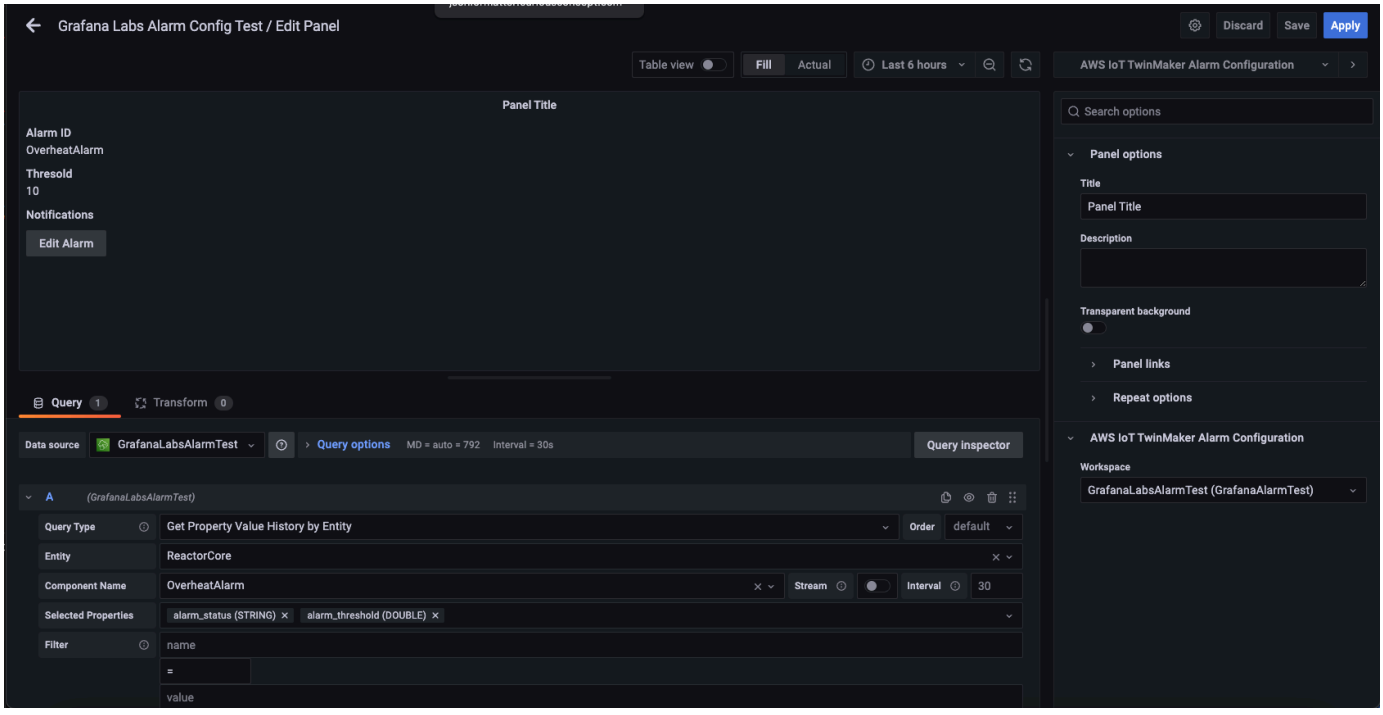
1. Select the workspace in the panel options.

2. Set your datasource in the query configuration.
3. Use the following query type: `Get Property Value History by Entity`.
4. Select an entity or entity variable, you wish to add an alarm to.
5. Once you have selected the entity, select a component or component variable, to apply a property to.
6. For the property, choose: `alarm_status` and `alarm_threshold`.

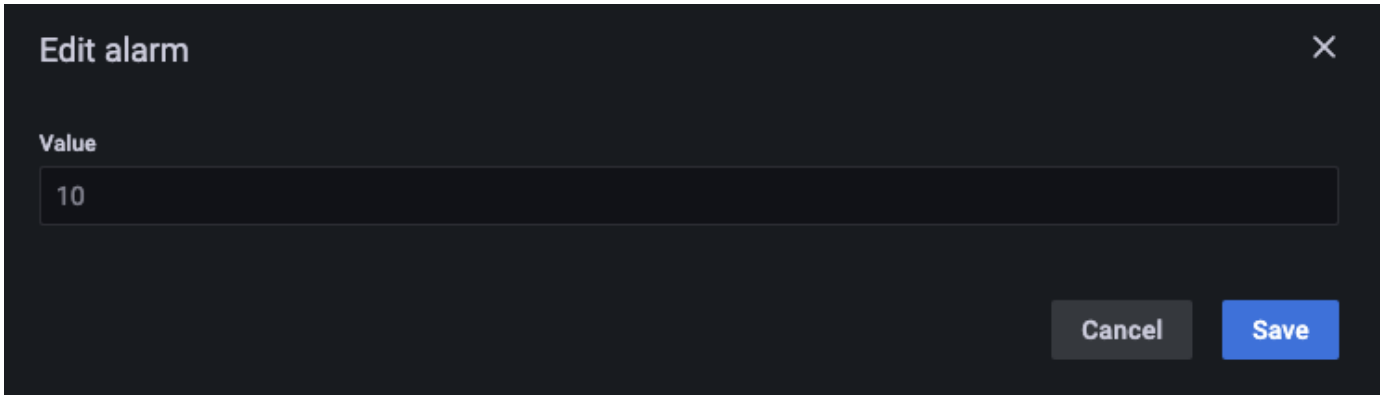
   When it's connected you should see the Id for the alarm Id and it's current threshold.

   > **Note**
   >
   > For the public preview, no notifications are shown. You should review your alarm status and threshold to make sure the properties were applied correctly.

7. The default Query Order of Ascending should be used so the latest value shows.
8. The filter section of the Query can be left empty. A complete configuration is pictured below:



9. By using the **Edit Alarm** button you can bring up a dialog to change the current alarm threshold.
10. Select **Save** to set the new threshold value.



Note

This panel should only be used with a live time range that includes the present. Using it with time ranges that end and start in the past may show unexpected values when editing alarm thresholds as the current threshold always.

AWS IoT TwinMaker Matterport integration

Matterport provides a variety of capture options to scan real-world environments and create immersive 3D models, also known as Matterport digital twins. These models are called Matterport spaces. AWS IoT TwinMaker supports Matterport integration, allowing you to import your Matterport digital twins into your AWS IoT TwinMaker scenes. By pairing Matterport digital twins with AWS IoT TwinMaker, you can visualize and monitor your digital twin system in a virtual environment.

For more information about using Matterport, read Matterport's documentation on AWS IoT TwinMaker and Matterport page.

Integration topics

- Integration overview
- Matterport integration prerequisites
- Generate and record your Matterport credentials
- Store your Matterport credentials in AWS Secrets Manager
- Import Matterport spaces into AWS IoT TwinMaker scenes
- Use Matterport spaces in your AWS IoT TwinMaker Grafana dashboard

- Use Matterport spaces in your AWS IoT TwinMaker web application

Integration overview

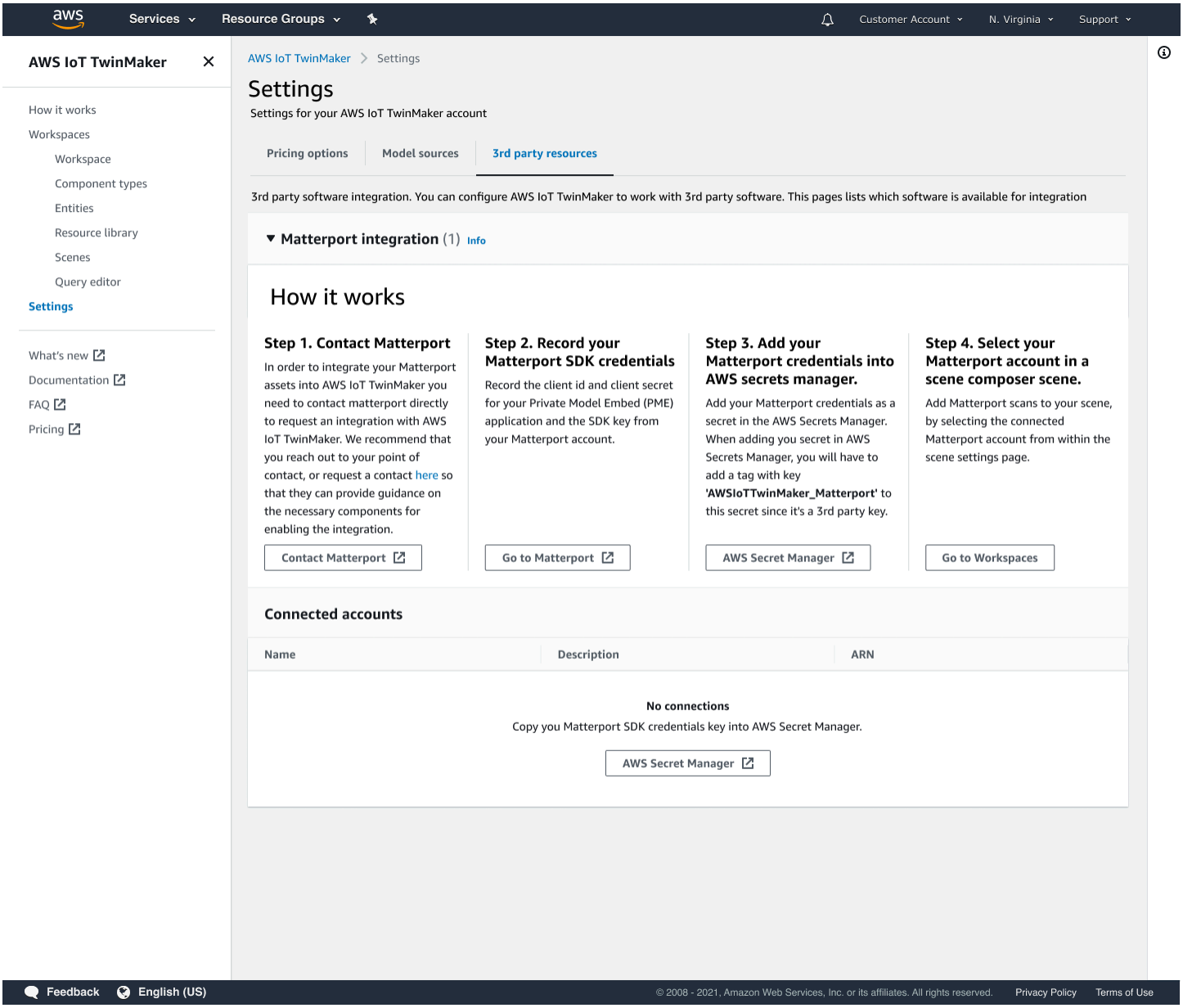
This integration enables you to do the following:

- Use your Matterport tags and spaces in the AWS IoT TwinMaker app kit.
- View your imported matterport data in your AWS IoT TwinMaker Grafana dashboard. For more information on using AWS IoT TwinMaker and Grafana, read the Grafana dashboard integration documentation.
- Import your Matterport spaces into your AWS IoT TwinMaker scenes.
- Select and import your Matterport tags that you'd like to bind to data in your AWS IoT TwinMaker scene.
- Automatically surface your Matterport space and tag changes in your AWS IoT TwinMaker scene and approve which to synchronize.

The integration process is comprised of 3 critical steps.

1. **Generate and record your Matterport credentials**
2. **Store your Matterport credentials in AWS Secrets Manager**
3. **Import Matterport spaces into AWS IoT TwinMaker scenes**

You start your integration in the AWS IoT TwinMaker console. In the console's **Settings** page, under **3rd party resources**, open **Matterport integration** to navigate between the different resources required for the integration.



Matterport integration prerequisites

Before integrating Matterport with AWS IoT TwinMaker please make sure you meet the following prerequisites:

- You have purchased an Enterprise-level Matterport account and the Matterport products necessary for the AWS IoT TwinMaker integration.
- You have an AWS IoT TwinMaker workspace. For more information, see Getting started with AWS IoT TwinMaker.

- You have updated your AWS IoT TwinMaker workspace role. For more information on creating a workspace role, see Create and manage a service role for AWS IoT TwinMaker.

  Add the following to your workspace role:

  ```
  {
    "Effect": "Allow",
    "Action": "secretsmanager:GetSecretValue",
    "Resource": [
     "AWS Secrets Manager secret ARN"
    ]
  }
  ```

- You must contact Matterport to configure the necessary licensing for enabling the integration. Matterport will also enable a Private Model Embed (PME) for the integration.

  If you already have a Matterport account manager, contact them directly.

  Use the following procedure to contact Matterport and request an integration if you don't have a Matterport point of contact:

  1. Open the **Matterport and AWS IoT TwinMaker** page.
  2. Press the **Contact us** button, to open the contact form.
  3. Fill out the required information on the form.
  4. When you're ready, choose **SAY HELLO** to send your request to Matterport.

  Once you have requested integration, you can generate the required Matterport SDK and Private Model Embed (PME) credentials needed to continue the integration process.

  > **Note**
  >
  > This may involve you incurring a fee for purchasing new products or services.

Generate and record your Matterport credentials

To integrate Matterport with AWS IoT TwinMaker, you must provide AWS Secrets Manager with Matterport credentials. Use the following procedure to generate the Matterport SDK credentials.

1. Log in to your Matterport account.
2. Navigate to your account settings page.
3. Once in the settings page, select the **Developer tools** option.
4. On the **Developer tools** page, go to the **SDK Key Management** section.
5. Once in the **SDK Key Management** section, select the option to add a new SDK key.
6. Once you have the Matterport SDK key, add domains to the key for AWS IoT TwinMaker and your Grafana server. If you are using the AWS IoT TwinMaker app kit, then make sure to add your custom domain as well.
7. Next, find the **Application integration Management** section, you should see your **PME application listed**. Record the following information:

   - The **Client ID**
   - The **Client Secret**

   > **Note**
   >
   > Since the **Client Secret** is only presented to you once, we strongly recommend that you record your **Client Secret**. You must present your **Client Secret** in the AWS Secrets Manager console to continue with the Matterport integration.

   These credentials are automatically created when you have purchased the necessary components and the PME for your account has been enabled by Matterport. If these credentials do not appear, contact Matterport. To request contact, see the **Matterport and AWS IoT TwinMaker** contact form.

For more information on Matterport SDK credentials, see Matterport's official SDK documentation SDK Docs Overview.

Store your Matterport credentials in AWS Secrets Manager

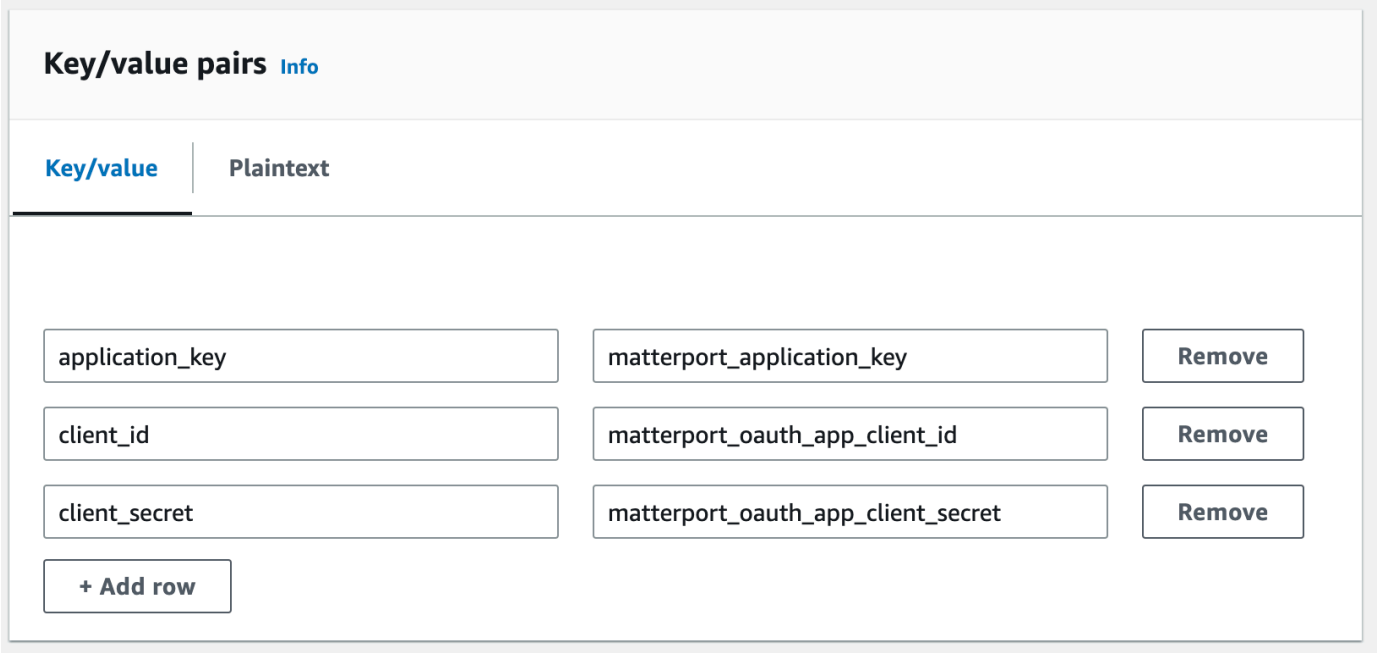
Use the following procedure to store your Matterport credentials in AWS Secrets Manager.

> **Note**
>
> You need the **Client Secret** created from the procedure in the Generate and record your Matterport credentials topic to continue with the Matterport integration.

1. Log in to the AWS Secrets Manager console.
2. Navigate to the **Secrets** page and select **Store a new secret**.
3. For the **Secret type**, select **Other type of secret**.
4. In the **Key/value pairs** section, add in the following key-value pairs, with your Matterport credentials as the values:

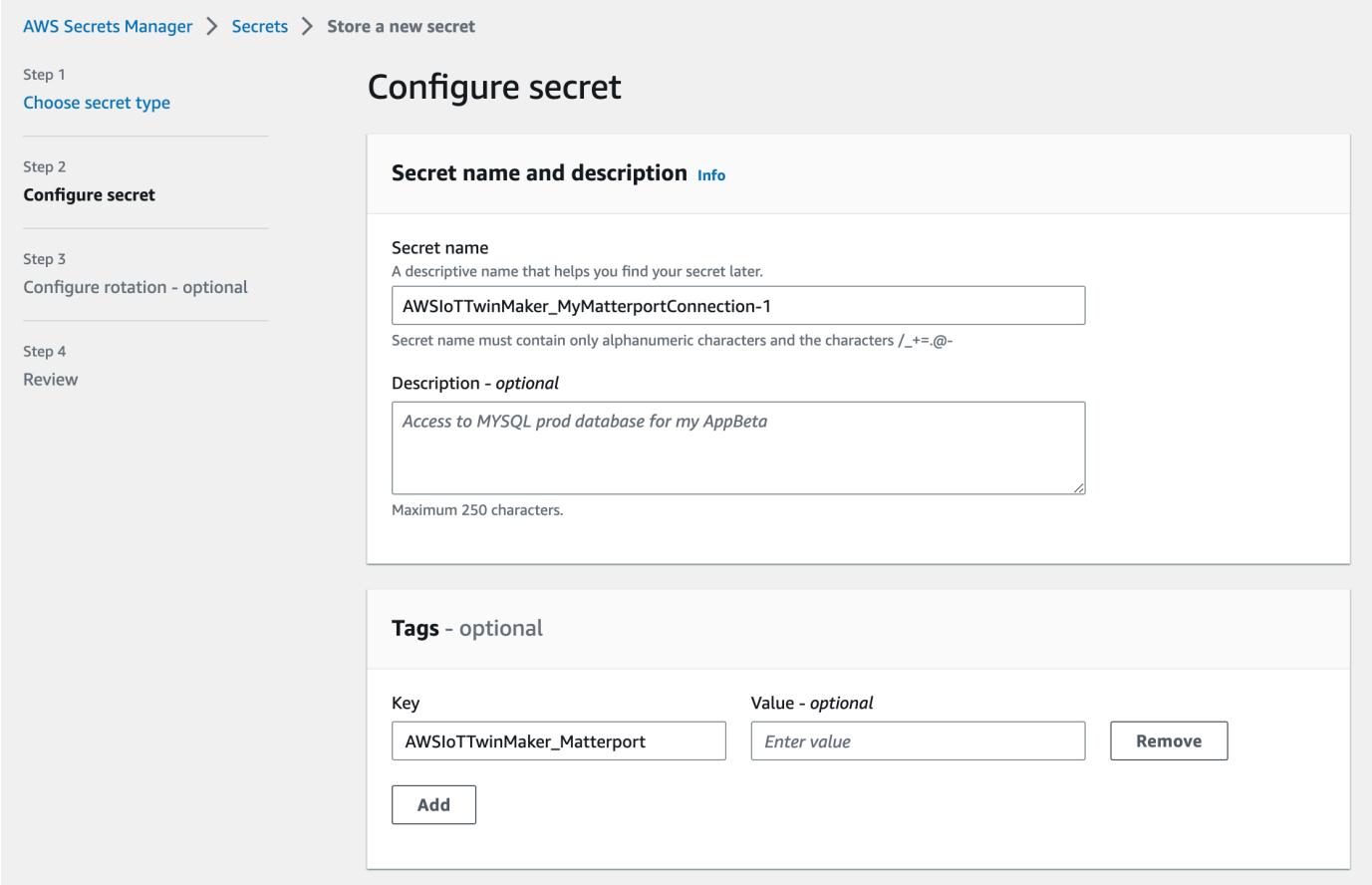
   - Create a key-value pair, with **Key:** `application_key`, and **Value:** *<your Matterport credentials>*.
   - Create a key-value pair, with **Key:** `client_id`, and **Value:** *<your Matterport credentials>*.
   - Create a key-value pair, with **Key:** `client_secret`, and **Value:** *<your Matterport credentials>*.

   When completed, you should have a configuration similar to the following example:



5. For the **Encryption key**, you can leave the default encryption key `aws/secretsmanager` selected.
6. Choose **Next** to move on to the **Configure secret** page.
7. Fill out the field for **Secret name** and the **Description**.
8. Add a tag to this secret in the **Tags** section.

   When creating the tag, assign the key as `AWSIoTTwinMaker_Matterport` as shown in the following screenshot:



> **Note**
>
> You must add a tag. Tags are required when adding 3rd party secrets into AWS Secrets Manager, despite **Tags** being listed as optional.

The **Value** field is optional. Once you have provided a **Key**, you can select **Add** to move on to the next step.

9. Choose **Next** to move on to the **Configure rotation** page. Setting up a secret rotation is optional. If you wish to finish adding your secret and don't need a rotation, choose **Next** again. For more information on secret rotation, see Rotate AWS Secrets Manager secrets.
10. Confirm your secret configuration on the **Review** page. Once you're ready to add your secret, choose **Store**.

For more information about using AWS Secrets Manager, see the following AWS Secrets Manager documentation:

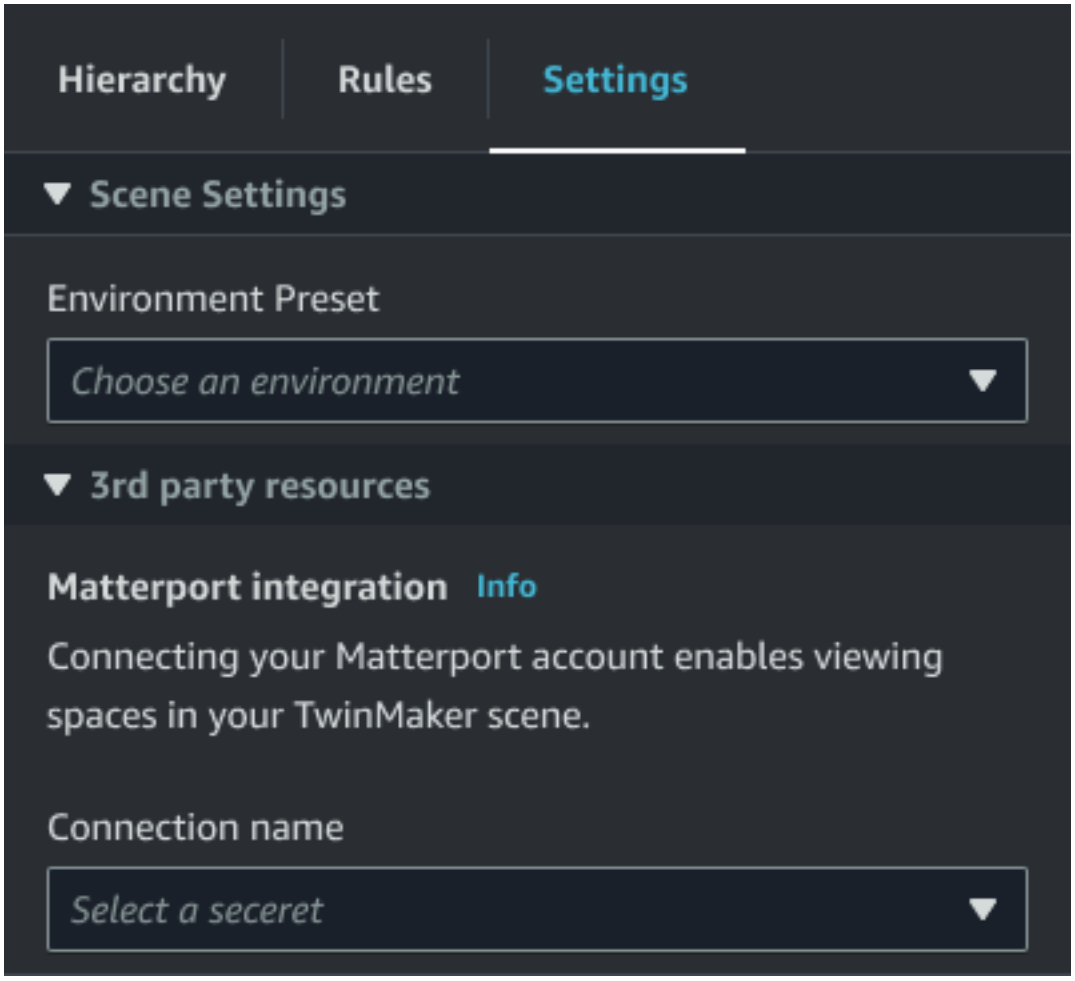
- Create and manage secrets with AWS Secrets Manager
- What is AWS Secrets Manager?
- Rotate AWS Secrets Manager secrets

Now you are ready to import your Matterport assets into AWS IoT TwinMaker scenes. See the procedure in the following section, Import Matterport spaces into AWS IoT TwinMaker scenes

Import Matterport spaces into AWS IoT TwinMaker scenes

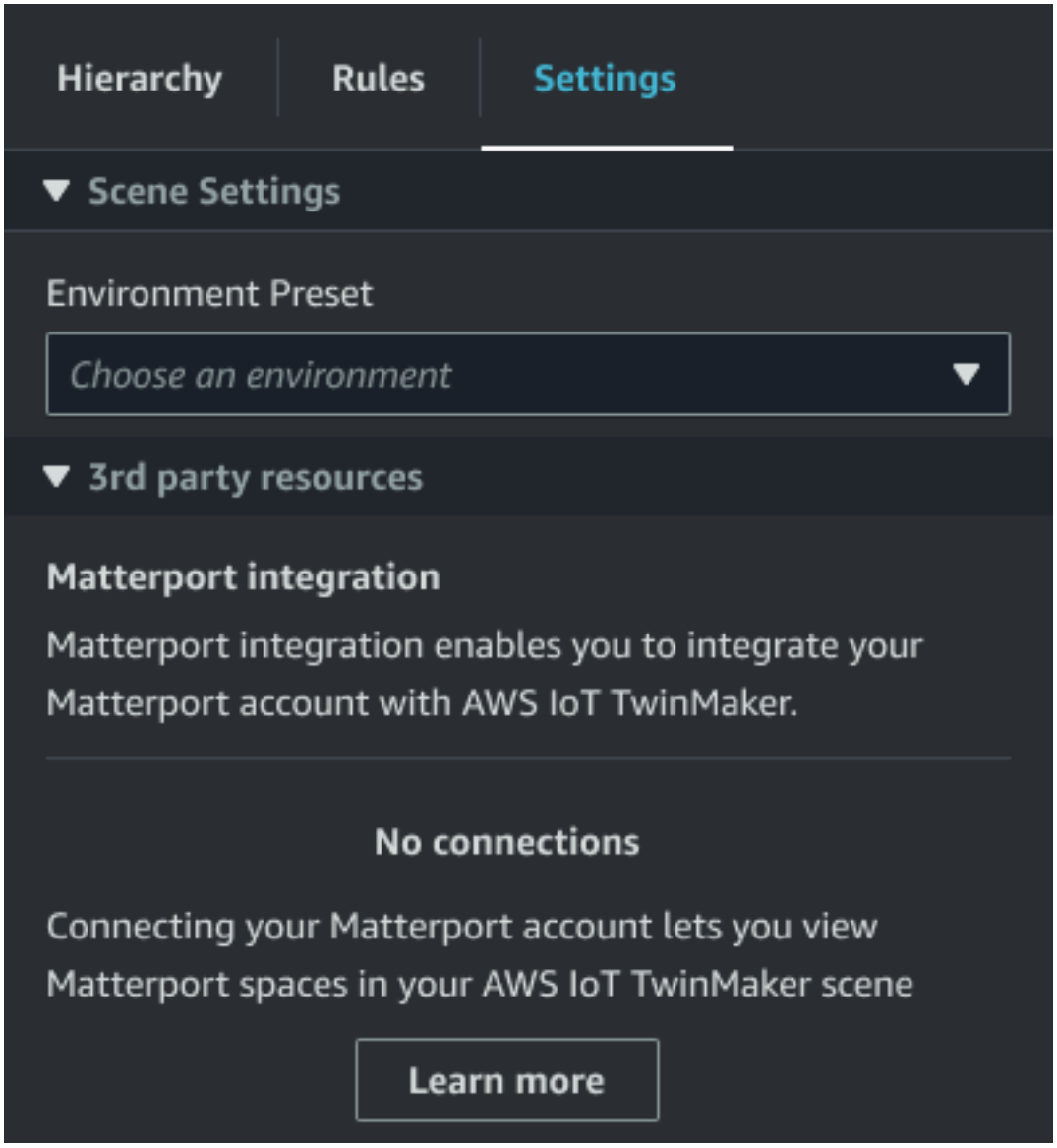
Add Matterport scans to your scene by selecting the connected Matterport account from within the scene settings page. Use the following procedure to import your Matterport scans and tags:

1. Log in to the AWS IoT TwinMaker console.
2. Create or open an existing AWS IoT TwinMaker scene in which you want to use a Matterport space.
3. Once the scene has opened, navigate to the **Settings** tab.
4. In **Settings**, under **3rd party resources**, find the **Connection name** and enter the secret you created in the procedure from Store your Matterport credentials in AWS Secrets Manager.

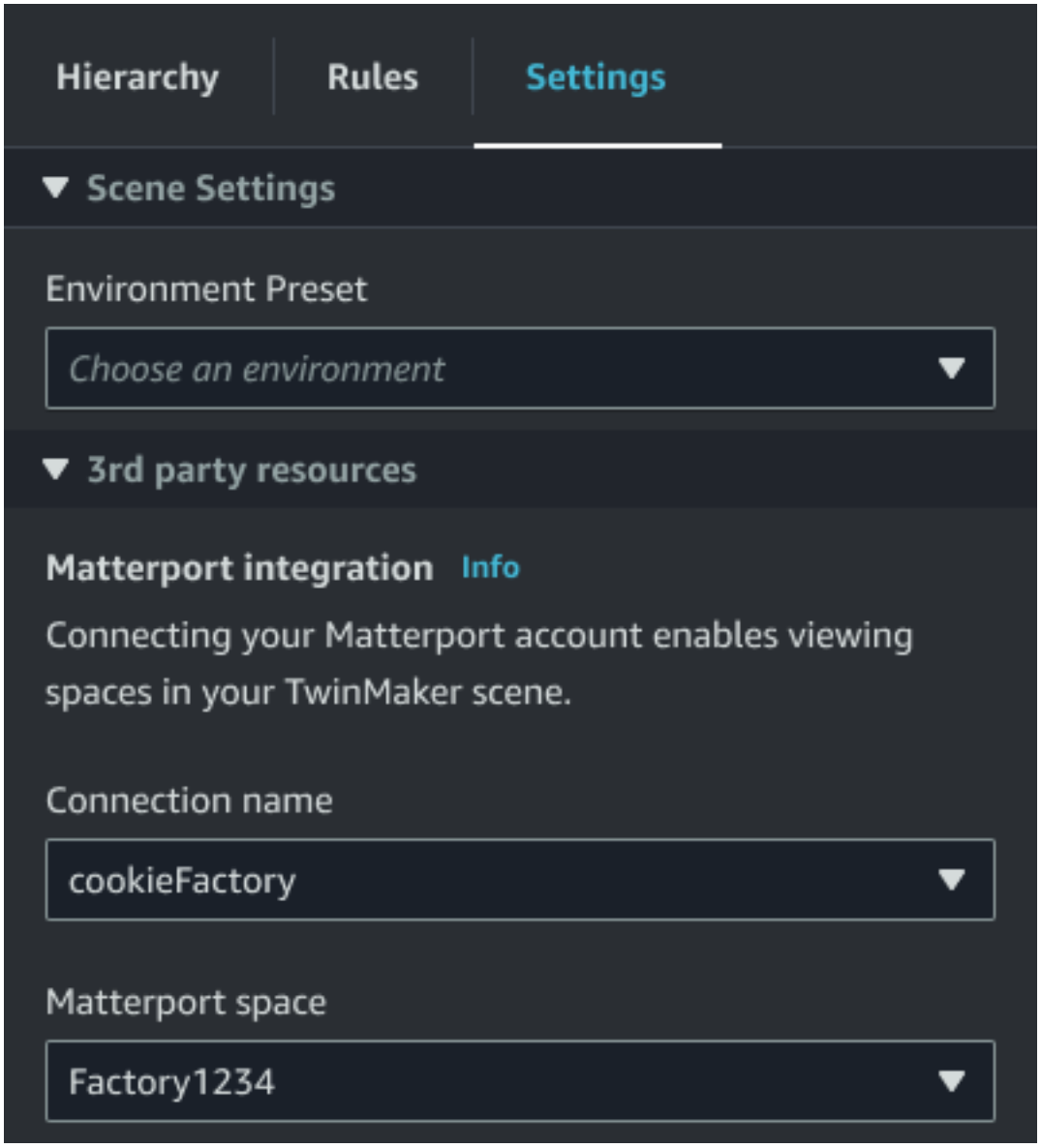


Note

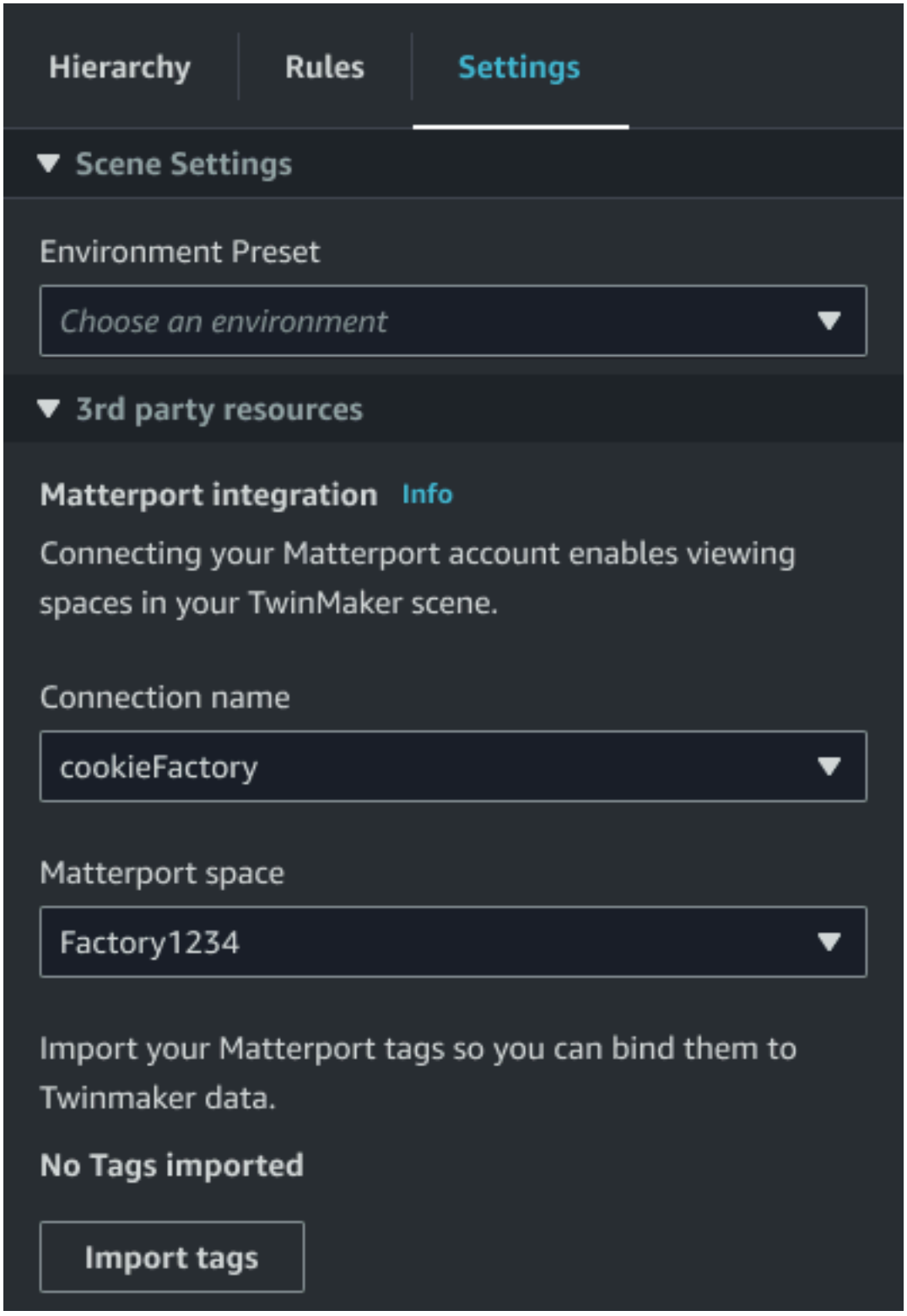
If you see a message that states **No connections**, navigate to the AWS IoT TwinMaker console settings page to begin the process for Matterport integration.



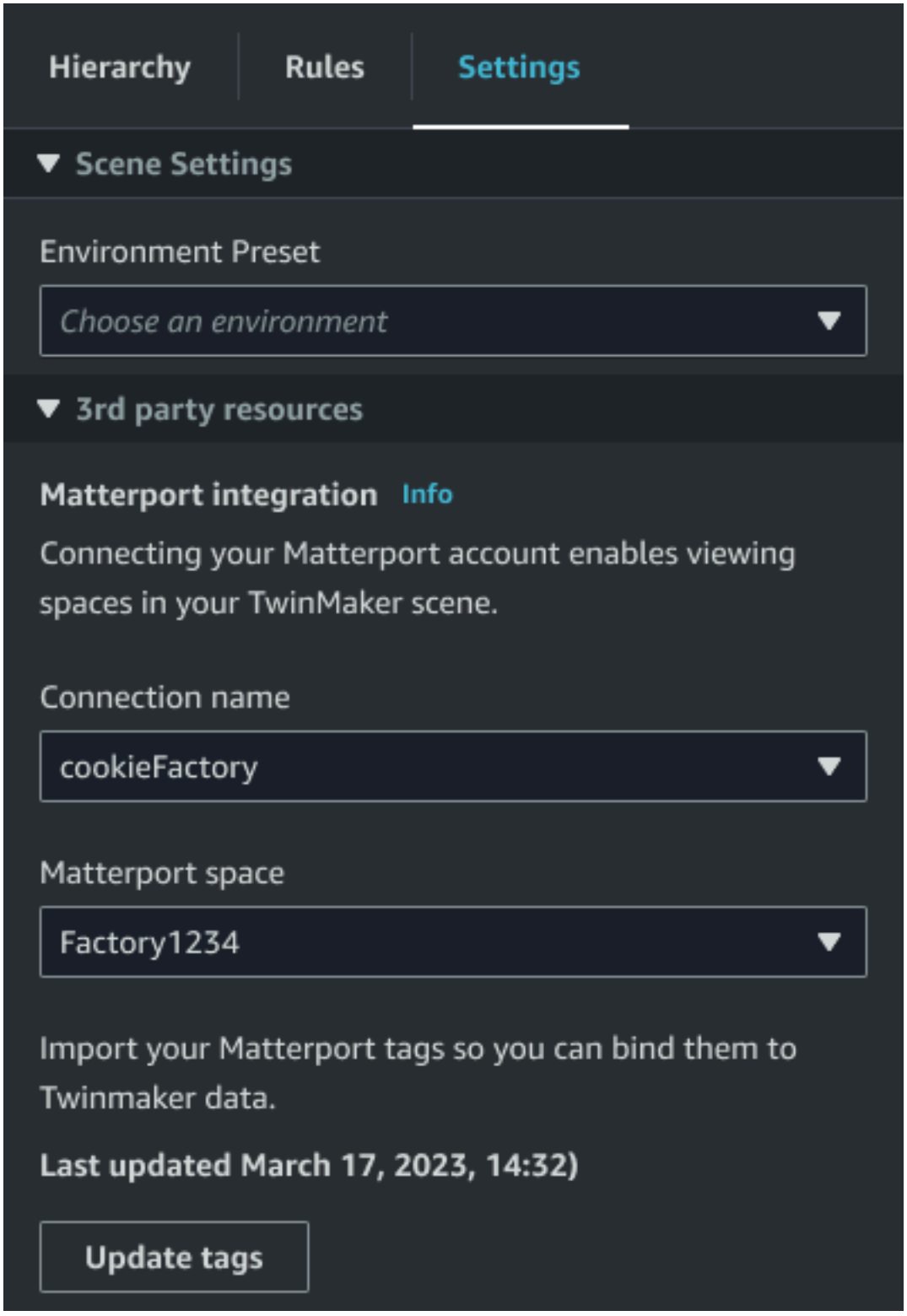
5. Next, choose the Matterport space you'd like to use in your scene by selecting it in the **Matterport space** drop-down.



6. After selecting a space, you can import your Matterport tags and convert them to AWS IoT TwinMaker scene tags by pressing the **Import tags** button.



After you have imported Matterport tags, the button is replaced by an **Update tags** button. You can continually update your Matterport tags in AWS IoT TwinMaker so that they always reflect the most recent changes in your Matterport account.



7. You have successfully integrated AWS IoT TwinMaker with Matterport, and now your AWS IoT TwinMaker scene has both your imported Matterport space and tags. You can work within this scene as you would with any other AWS IoT TwinMaker scene.

For more information on working with AWS IoT TwinMaker scenes, see Creating and editing AWS IoT TwinMaker scenes.

Use Matterport spaces in your AWS IoT TwinMaker Grafana dashboard

Once you have imported your Matterport space into an AWS IoT TwinMaker scene, you can view that scene with the Matterport space in your Grafana dashboard. If you have already configured Grafana with AWS IoT TwinMaker, then you can simply open the Grafana dashboard to view your scene with the imported Matterport space.

If you have not configured AWS IoT TwinMaker with Grafana yet, complete the Grafana integration process first. You have two choices when integrating AWS IoT TwinMaker with Grafana. You can use a self-managed Grafana instance or you can use Amazon Managed Grafana.

See the following documentation to learn more about the Grafana options and integration process:

- AWS IoT TwinMaker Grafana dashboard integration.
- Amazon Managed Grafana.
- Self-managed Grafana.

Use Matterport spaces in your AWS IoT TwinMaker web application

Once you have imported your Matterport space into an AWS IoT TwinMaker scene, you can view that scene with the Matterport space in your AWS IoT app kit web application.

See the following documentation to learn more about using the AWS IoT application kit:

- To learn more about using AWS IoT TwinMaker with the AWS IoT app kit, see Create a customized web application using AWS IoT TwinMaker UI Components.
- To learn more about using AWS IoT application kit, please visit AWS IoT Application kit Github page.
- For instructions on how to start a new web application using AWS IoT application kit, please visit the official IoT App Kit documentation page.

AWS IoT TwinMaker video integration

Video cameras present a good opportunity for digital twin simulation. You can use AWS IoT TwinMaker to simulate your camera's location and physical conditions. Create entities in AWS IoT TwinMaker for your on-site cameras, and use video components to stream live video and metadata from your site to your AWS IoT TwinMaker scene or to a Grafana dashboard.

AWS IoT TwinMaker can capture video from edge devices in two ways. You can stream video from edge devices with the edge connector for Kinesis video stream, or you can save video on the edge device and initiate video uploading with MQTT messages. Use this component to stream video data from your devices for use with AWS IoT services. To generate the required resources and deploy the edge connector for Kinesis Video Streams, see the Getting started with the edge connector for Kinesis video stream on GitHub. For more information about the AWS IoT Greengrass component, see the AWS IoT Greengrass documentation on edge connector for Kinesis Video Streams.

After you've created the required AWS IoT SiteWise models and configured the Kinesis Video Streams Greengrass component, you can stream or record video on the edge to your digital twin application in the AWS IoT TwinMaker console. You can also view livestreams and metadata from your devices in a Grafana dashboard. For more information about integrating Grafana and AWS IoT TwinMaker, see AWS IoT TwinMaker Grafana dashboard integration.

Use the edge connector for Kinesis video stream to stream video in AWS IoT TwinMaker

With the edge connector for Kinesis video stream, you can stream video and data to an entity in your AWS IoT TwinMaker scene. You use a video component to do this. To create the video component for use in your scenes, complete the following procedure.

Prerequisites

Before you create the video component in your AWS IoT TwinMaker scene, make sure you've completed the following prerequisites.

- Created the required AWS IoT SiteWise models and assets for the edge connector for Kinesis video stream. For more information about creating the AWS IoT SiteWise assets for the connector, see Getting started with the edge connector for Kinesis video stream.

- Deployed the Kinesis video stream edge connector on your AWS IoT Greengrass device. For more information about deploying the Kinesis video stream edge connector component, see the deployment README.

Create video components for AWS IoT TwinMaker scenes

Complete the following steps to create the edge connector for the Kinesis video stream component for your scene.

1. In the AWS IoT TwinMaker console, open the scene you want to add the video component to.
2. After the scene is opens, choose an existing entity or create the entity you want to add the component to, and then choose **Add component**.
3. In the **Add component** pane, enter a name for the component, and for the **Type**, choose **com.amazon.iotsitewise.connector.edgevideo**.
4. Choose an **Asset Model** by selecting the name of the AWS IoT SiteWise camera model you created. This name should have the following format: `EdgeConnectorForKVSCameraModel-0abc`, where the string of letters and numbers at the end matches your own asset name.
5. For **Asset**, choose the AWS IoT SiteWise camera assets you want to stream video from. A small window appears showing you a preview of the current video stream.

   > **Note**
   >
   > To test your video streaming, choose **test**. This test sends out an MQTT event to initiate video live streaming. Wait for a few moments to see the video show up in the player.

6. To add the video component to your entity, choose **Add component**.

Add video and metadata from Kinesis video stream to a Grafana dashboard

After you've created a video component for your entity in your AWS IoT TwinMaker scene, you can configure the video panel in Grafana to see live streams. Make sure you have properly integrated AWS IoT TwinMaker with Grafana. For more information, see AWS IoT TwinMaker Grafana dashboard integration.

> **Important**
>
> To view video in your Grafana dashboard, you must make sure the Grafana datasources have the proper IAM permissions. To create the required role and policy see Creating a dashboard IAM role.

Complete the following steps to see Kinesis Video Streams and metadata in your Grafana dashboard.

1. Open the AWS IoT TwinMaker dashboard.
2. Choose **Add panel**, and then choose **Add an empty panel**.

   > **Note**
   >
   > For Grafana v10.4, the AWS IoT TwinMaker video player is found under **Widget**. Select **Add >> Widget**.

3. From the panels list, choose the **AWS IoT TwinMaker video player** panel.
4. In the **AWS IoT TwinMaker video player** panel, enter the **stream name** of the **KinesisVideoStreamName**, with the name of the Kinesis video stream you want to stream video from.

   > **Note**
   >
   > To stream metadata to the Grafana video panel, you must first have created an entity with a video streaming component.

5. **Optional:** To stream metadata from AWS IoT SiteWise assets to the video player, for **Entity**, choose the AWS IoT TwinMaker entity that you created in your AWS IoT TwinMaker scene. For the **Component name**, choose the video component you created for the entity in your AWS IoT TwinMaker scene.

Using the AWS IoT TwinMaker Flink library

AWS IoT TwinMaker provides a Flink library that you can use to read and write data to external data stores used in your digital twins.

You use the AWS IoT TwinMaker Flink library by installing it as a custom connector in Managed Service for Apache Flink and performing Flink SQL queries in a Zeppelin notebook in Managed Service for Apache Flink. The notebook can be promoted to a continuously running stream processing application. The library leverages AWS IoT TwinMaker components to retrieve data from your workspace.

The AWS IoT TwinMaker Flink library requires the following.

Prerequisites

1. A fully populated workspace with scenes and components. Use the built-in component types for data from AWS services (AWS IoT SiteWise and Kinesis Video Streams). Create custom component types for data from third-party sources. For more information, see ???.
2. An understanding of Studio notebooks with Managed Service for Apache Flink for Apache Flink. These notebooks are powered by Apache Zeppelin and use the Apache Flink framework. For more information, see Using a Studio notebook with Managed Service for Apache Flink for Apache Flink.

For instructions on using the library, see the AWS IoT TwinMaker Flink library user guide.

For instructions on setting up AWS IoT TwinMaker with the quick start in AWS IoT TwinMaker samples, see README file for the sample insights application.

Logging and monitoring in AWS IoT TwinMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT TwinMaker and your other AWS solutions. AWS IoT TwinMaker supports the following monitoring tools to watch the service, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors in real time your AWS resources and the applications that you run on AWS. 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 for your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* monitors, stores, and provides access to your log files from AWS IoT TwinMaker 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 TwinMaker with Amazon CloudWatch metrics

You can monitor AWS IoT TwinMaker by 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 TwinMaker publishes the metrics and dimensions listed in the following sections to the `AWS/IoTTwinMaker` namespace.

> **Tip**
>
> AWS IoT TwinMaker 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** to see the highest available resolution of your metric data.

Contents

Metrics

AWS IoT TwinMaker publishes the following metrics.

Metrics

Metric	Description
`ComponentTypeCreationFailure`	This metric reports whether the component type creation is successful. The metric is published when a component type is in `CREATING` state. This happens when a component type is created with the required properties in the schema initializer and these properties are instantiated with default values. The metric value can be either `0` for success or 1 for failure. **Dimensions**: ComponentTypeId, WorkspaceId. **Units**: Count

Metric	Description
`ComponentTypeUpdateFailure`	This metric reports whether the component type update is successful. The metric is published when a component type is in `UPDATING` state. This happens when a component type is updated with the required properties in the schema initializer and these properties are instantiated with default values. The metric value can be either `0` for success or 1 for failure. **Dimensions**: ComponentTypeId, WorkspaceId. **Units**: Count
`EntityCreationFailure`	This metric reports whether the entity creation is successful. The metric is published when an entity is in `CREATING` state. This happens when an entity is created with a component. The metric value can be either `0` for success or 1 for failure. **Dimensions**: EntityName, EntityId, Workspace Id. **Units**: Count

Metric	Description
`EntityUpdateFailure`	This metric reports whether the entity update is successful. The metric is published when an entity is in `UPDATING` state. This happens when an entity is updated. The metric value can be either `0` for success or `1` for failure. **Dimensions**: EntityName, EntityId, WorkspaceId. **Units**: Count
`EntityDeletionFailure`	This metric reports whether the entity deletion is successful. The metric is published when an entity is in `DELETING` state. This happens when an entity is deleted. The metric value can be either `0` for success or `1` for failure. **Dimensions**: EntityName, EntityId, WorkspaceId. **Units**: Count

Tip

All metrics are published to the `AWS/IoTTwinMaker` namespace.

Logging AWS IoT TwinMaker API calls with AWS CloudTrail

AWS IoT TwinMaker 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 TwinMaker. CloudTrail captures API calls for AWS IoT TwinMaker as events. The calls captured include calls from the AWS IoT TwinMaker

console and code calls to the AWS IoT TwinMaker 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 TwinMaker. 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 TwinMaker, 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 TwinMaker information in CloudTrail

When you create your AWS account, CloudTrail is automatically enabled. CloudTrail records support event activity that occurs in AWS IoT TwinMaker, 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 TwinMaker, 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. CloudTrail 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 TwinMaker operations are logged by CloudTrail and are documented in the AWS IoT TwinMaker API Reference.

The following data plane operations aren't logged by CloudTrail:

- GetPropertyValue
- GetPropertyValueHistory
- BatchPutPropertyValues

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 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.

Security in AWS IoT TwinMaker

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 TwinMaker, 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 TwinMaker. The following topics show you how to configure AWS IoT TwinMaker 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 TwinMaker resources.

Topics

Data protection in AWS IoT TwinMaker

The AWS shared responsibility model applies to data protection in AWS IoT TwinMaker. As described in this model, AWS is responsible for protecting the global infrastructure that runs all

of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT TwinMaker or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

AWS IoT TwinMaker stores your workspace information in an Amazon S3 bucket that the service creates for you, if you choose. The bucket that the service creates for you has default server-side encryption enabled. If you choose to use your own Amazon S3 bucket when you create a new

workspace, we recommend that you enable default server-side encryption. For more information about default encryption in Amazon S3, see Setting default server-side encryption behavior for Amazon S3 buckets.

Encryption in transit

All data sent to AWS IoT TwinMaker is sent over a TLS connection using the HTTPS protocol, so it's secure by default while in transit.

> **Note**
>
> We recommend that you use HTTPS on Amazon S3 bucket addresses as a control to enforce encryption in transit when AWS IoT TwinMaker interacts with an Amazon S3 bucket. For more information on Amazon S3 buckets, see Creating, configuring, and working with Amazon S3 buckets.

Identity and Access Management for AWS IoT TwinMaker

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 TwinMaker resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS IoT TwinMaker identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS IoT TwinMaker works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS IoT TwinMaker)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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 TwinMaker works with IAM

Before you use IAM to manage access to AWS IoT TwinMaker, learn what IAM features are available to use with AWS IoT TwinMaker.

IAM features you can use with AWS IoT TwinMaker

IAM feature	AWS IoT TwinMaker support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how AWS IoT TwinMaker and other AWS services work with most IAM features, see AWS services that work with IAM in the *AWS IAM Identity Center User Guide*.

Identity-based policies for AWS IoT TwinMaker

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS IoT TwinMaker

To view examples of AWS IoT TwinMaker identity-based policies, see Identity-based policy examples for AWS IoT TwinMaker.

Resource-based policies within AWS IoT TwinMaker

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS IoT TwinMaker

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS IoT TwinMaker actions, see Actions defined by AWS IoT TwinMaker in the *Service Authorization Reference*.

Policy actions in AWS IoT TwinMaker use the following prefix before the action:

```
iottwinmaker
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
   "iottwinmaker:action1",
   "iottwinmaker:action2"
         ]
```

To view examples of AWS IoT TwinMaker identity-based policies, see Identity-based policy examples for AWS IoT TwinMaker.

Policy resources for AWS IoT TwinMaker

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS IoT TwinMaker resource types and their ARNs, see Resources defined by AWS IoT TwinMaker in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS IoT TwinMaker.

To view examples of AWS IoT TwinMaker identity-based policies, see Identity-based policy examples for AWS IoT TwinMaker.

Policy condition keys for AWS IoT TwinMaker

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS IoT TwinMaker condition keys, see Condition keys for AWS IoT TwinMaker in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS IoT TwinMaker.

To view examples of AWS IoT TwinMaker identity-based policies, see Identity-based policy examples for AWS IoT TwinMaker.

Access control lists (ACLs) in AWS IoT TwinMaker

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with AWS IoT TwinMaker

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using Temporary credentials with AWS IoT TwinMaker

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for AWS IoT TwinMaker

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS IoT TwinMaker

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break AWS IoT TwinMaker functionality. Edit service roles only when AWS IoT TwinMaker provides guidance to do so.

Service-linked roles for AWS IoT TwinMaker

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS IoT TwinMaker

By default, users and roles don't have permission to create or modify AWS IoT TwinMaker resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS IoT TwinMaker, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS IoT TwinMaker in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the AWS IoT TwinMaker console
- Allow users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS IoT TwinMaker resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies

that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS IoT TwinMaker console

To access the AWS IoT TwinMaker console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT TwinMaker 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 (users or roles) with that policy.

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 they're trying to perform.

To ensure that users and roles can still use the AWS IoT TwinMaker console, also attach the AWS IoT TwinMaker `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *AWS IAM Identity Center User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting AWS IoT TwinMaker identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT TwinMaker and IAM.

Topics

I am not authorized to perform an action in AWS IoT TwinMaker

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `iottwinmaker:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iottwinmaker:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `iottwinmaker:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS IoT TwinMaker.

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 TwinMaker. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS IoT TwinMaker 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 TwinMaker supports these features, see How AWS IoT TwinMaker works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Using service-linked roles for AWS IoT TwinMaker

AWS IoT TwinMaker 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 TwinMaker.

Service-linked roles are predefined by AWS IoT TwinMaker 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 TwinMaker easier because you don't have to manually add the necessary permissions. AWS IoT TwinMaker defines the permissions of its service-linked roles, and unless defined otherwise, only AWS IoT TwinMaker can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot 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 TwinMaker 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 roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS IoT TwinMaker

AWS IoT TwinMaker uses the service-linked role named **AWSServiceRoleForIoTTwinMaker** – Allows AWS IoT TwinMaker to call other AWS services and to sync their resources on your behalf.

The AWSServiceRoleForIoTTwinMaker service-linked role trusts the following services to assume the role:

- `iottwinmaker.amazonaws.com`

The role permissions policy named AWSIoTTwinMakerServiceRolePolicy allows AWS IoT TwinMaker to complete the following actions on the specified resources:

- Action: `iotsitewise:DescribeAsset, iotsitewise:ListAssets, iotsitewise:DescribeAssetModel, and iotsitewise:ListAssetModels, iottwinmaker:GetEntity, iottwinmaker:CreateEntity, iottwinmaker:UpdateEntity, iottwinmaker:DeleteEntity, iottwinmaker:ListEntities, iottwinmaker:GetComponentType, iottwinmaker:CreateComponentType, iottwinmaker:UpdateComponentType, iottwinmaker:DeleteComponentType, iottwinmaker:ListComponentTypes` on `all your iotsitewise asset and asset-model resources`

You must configure permissions to allow your users, groups, or roles 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 TwinMaker

You don't need to manually create a service-linked role. When you synchronize your AWS IoT SiteWise assets and asset models (asset sync) in the AWS Management Console, the AWS CLI, or the AWS API, AWS IoT TwinMaker 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 synchronize your AWS IoT SiteWise assets and asset models (asset sync), AWS IoT TwinMaker creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **"IoT TwinMaker - Managed Role"** use case. In the AWS CLI or the AWS API, create a service-linked role with the `iottwinmaker.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 TwinMaker

AWS IoT TwinMaker does not allow you to edit the AWSServiceRoleForIoTTwinMaker service-linked role. After you create a service-linked role, you cannot 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 TwinMaker

If you no longer need to use 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 is not actively monitored or maintained. However, you must clean up any serviceLinked-workspaces that are still using your service-linked role before you can manually delete the role.

> **Note**
>
> If the AWS IoT TwinMaker 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 the operation again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForIoTTwinMaker service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for AWS IoT TwinMaker service-linked roles

AWS IoT TwinMaker supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Regions and endpoints.

AWS managed policies for AWS IoT TwinMaker

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AWSIoTTwinMakerServiceRolePolicy

You can't attach AWSIoTTwinMakerServiceRolePolicy to your IAM entities. This policy is attached to a service-linked role that allows to perform actions on your behalf. For more information, see Service-linked role permissions for AWS IoT TwinMaker.

The role permissions policy named AWSIoTTwinMakerServiceRolePolicy allows AWS IoT TwinMaker to complete the following actions on the specified resources:

- Action: `iotsitewise:DescribeAsset, iotsitewise:ListAssets, iotsitewise:DescribeAssetModel, and iotsitewise:ListAssetModels, iottwinmaker:GetEntity, iottwinmaker:CreateEntity, iottwinmaker:UpdateEntity, iottwinmaker:DeleteEntity, iottwinmaker:ListEntities, iottwinmaker:GetComponentType, iottwinmaker:CreateComponentType, iottwinmaker:UpdateComponentType, iottwinmaker:DeleteComponentType, iottwinmaker:ListComponentTypes` on `all your iotsitewise asset and asset-model resources`

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Sid": "SiteWiseAssetReadAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribeAsset"
            ],
            "Resource": [
                "arn:aws:iotsitewise:*:*:asset/*"
            ]
        },
        {
            "Sid": "SiteWiseAssetModelReadAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:DescribeAssetModel"
            ],
            "Resource": [
```

```
                "arn:aws:iotsitewise:*:*:asset-model/*"
            ]
        },
        {
            "Sid": "SiteWiseAssetModelAndAssetListAccess",
            "Effect": "Allow",
            "Action": [
                "iotsitewise:ListAssets",
                "iotsitewise:ListAssetModels"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "TwinMakerAccess",
            "Effect": "Allow",
            "Action": [
                "iottwinmaker:GetEntity",
                "iottwinmaker:CreateEntity",
                "iottwinmaker:UpdateEntity",
                "iottwinmaker:DeleteEntity",
                "iottwinmaker:ListEntities",
                "iottwinmaker:GetComponentType",
                "iottwinmaker:CreateComponentType",
                "iottwinmaker:UpdateComponentType",
                "iottwinmaker:DeleteComponentType",
                "iottwinmaker:ListComponentTypes"
            ],
            "Resource": [
                "arn:aws:iottwinmaker:*:*:workspace/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iottwinmaker:linkedServices": [
                        "IOTSITEWISE"
                    ]
                }
            }
        }
    ]
}
```

AWS IoT TwinMaker updates to AWS managed policies

View details about updates to AWS managed policies for since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Document history page.

Change	Description	Date
AWSIoTTwinMakerServiceRolePolicy – Added a policy	AWS IoT TwinMaker added the role permissions policy named AWSIoTTwinMakerServiceRolePolicy which allows AWS IoT TwinMaker to complete the following actions on the specified resources: • Action: `iotsitewise:DescribeAsset, iotsitewise:ListAssets, iotsitewise:DescribeAssetModel, and iotsitewise:ListAssetModels, iottwinmaker:GetEntity, iottwinmaker:CreateEntity, iottwinmaker:UpdateEntity, iottwinmaker:DeleteEntity, iottwinmaker:ListEntities, iottwinmaker:GetComponentType, iottwinmaker:Creat`	November 17, 2023

Change	Description	Date
	eComponentType, iottwinmaker:Updat eComponentType, iottwinmaker:Delet eComponentType, iottwinmaker:ListC omponentTypes on all your iotsitewise asset and asset-mod el resources For more information, see Service-linked role permissio ns for AWS IoT TwinMaker.	
started tracking changes	started tracking changes for its AWS managed policies.	May 11, 2022

AWS IoT TwinMaker and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your virtual private cloud (VPC) and AWS IoT TwinMaker by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, which you can use to privately access AWS IoT TwinMaker APIs without an internet gateway, network address translation (NAT) device, VPN connection, or AWS Direct Connect connection. AWS IoT TwinMaker supports both IPv4 and IPv6 (dual-stack) through its interface endpoints. Instances in your VPC don't need public IP addresses to communicate with AWS IoT TwinMaker APIs. Traffic between your VPC and AWS IoT TwinMaker doesn't leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for AWS IoT TwinMaker VPC endpoints

Before you set up an interface VPC endpoint for AWS IoT TwinMaker, review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

AWS IoT TwinMaker supports making calls to all of its API actions from your VPC.

- For data plane API operations, use the following endpoint:

  ```
  data.iottwinmaker.region.amazonaws.com
  ```

  The data plane API operations include the following:

  - GetPropertyValue
  - GetPropertyValueHistory
  - BatchPutPropertyValues
- For the control plane API operations, use the following endpoint:

  ```
  api.iottwinmaker.region.amazonaws.com
  ```

  The supported control plane API operations include the following:

  - CreateComponentType
  - CreateEntity
  - CreateScene
  - CreateWorkspace
  - DeleteComponentType
  - DeleteEntity
  - DeleteScene
  - DeleteWorkspace
  - GetComponentType
  - GetEntity
  - GetScene
  - GetWorkspace
  - ListComponentTypes
  - ListComponentTypes

- ListEntities
- ListScenes
- ListTagsForResource
- ListWorkspaces
- TagResource
- UntagResource
- UpdateComponentType
- UpdateEntity
- UpdateScene
- UpdateWorkspace

Creating an interface VPC endpoint for AWS IoT TwinMaker

You can create a VPC endpoint for the AWS IoT TwinMaker service by using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for AWS IoT TwinMaker that uses the following service name.

- For data plane API operations, use the following service name:

```
com.amazonaws.region.iottwinmaker.data
```

- For control plane API operations, use the following service name:

```
com.amazonaws.region.iottwinmaker.api
```

If you enable private DNS for the endpoint, you can make API requests to AWS IoT TwinMaker by using its default DNS name for the Region, for example, `iottwinmaker.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

AWS IoT TwinMaker PrivateLink is supported in the following regions:

- **us-east-1**

  The ControlPlane service is supported in the following availability zones: `use1-az1`, `use1-az2`, and `use1-az6`.

  The DataPlane service is supported in the following availability zones: `use1-az1`, `use1-az2`, and `use1-az4`.
- **us-west-2**

  The ControlPlane and DataPlane services are supported in the following availability zones: `usw2-az1`, `usw2-az2`, and `usw2-az3`.
- **eu-west-1**
- **eu-central-1**
- **ap-southeast-1**
- **ap-southeast-2**

For more information on availability zones, see Availability Zone IDs for your AWS resources - AWS Resource Access Manager.

Accessing AWS IoT TwinMaker through an interface VPC endpoint

When you create an interface endpoint, AWS IoT TwinMaker generates endpoint-specific DNS hostnames that you can use to communicate with AWS IoT TwinMaker. The private DNS option is enabled by default. For more information, see Using private hosted zones in the *Amazon VPC User Guide*.

If you enable private DNS for the endpoint, you can make API requests to AWS IoT TwinMaker through one of the following VPC endpoints.

- For the data plane API operations, use the following endpoint. Replace *`region`* with your AWS Region.

  ```
  data.iottwinmaker.region.amazonaws.com
  ```
- For the control plane API operations, use the following endpoint. Replace *`region`* with your AWS Region.

  ```
  api.iottwinmaker.region.amazonaws.com
  ```

If you disable private DNS for the endpoint, you must do the following to access AWS IoT TwinMaker through the endpoint:

- Specify the VPC endpoint URL in API requests.
  - For the data plane API operations, use the following endpoint URL. Replace *vpc-endpoint-id* and *region* with your VPC endpoint ID and Region.

    ```
    vpc-endpoint-id.data.iottwinmaker.region.vpce.amazonaws.com
    ```

  - For the control plane API operations, use the following endpoint URL. Replace *vpc-endpoint-id* and *region* with your VPC endpoint ID and Region.

    ```
    vpc-endpoint-id.api.iottwinmaker.region.vpce.amazonaws.com
    ```

- Disable host prefix injection. The AWS CLI and AWS SDKs prepend the service endpoint with various host prefixes when you call each API operation. This causes the AWS CLI and AWS SDKs to produce invalid URLs for AWS IoT TwinMaker when you specify a VPC endpoint.

  > **Important**
  >
  > You can't disable host prefix injection in AWS CLI or AWS Tools for PowerShell. This means that if you've disabled private DNS, you won't be able to use AWS CLI or AWS Tools for PowerShell to access AWS IoT TwinMaker through the VPC endpoint. If you want to use these tools to access AWS IoT TwinMaker through the endpoint, enable private DNS.

  For more information about how to disable host prefix injection in the AWS SDKs, see the following documentation sections for each SDK:

  - AWS SDK for C++
  - AWS SDK for Go
  - AWS SDK for Go v2
  - AWS SDK for Java
  - AWS SDK for Java 2.x
  - AWS SDK for JavaScript
  - AWS SDK for .NET

- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for AWS IoT TwinMaker

You can attach an endpoint policy to your VPC endpoint that controls access to AWS IoT TwinMaker. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for AWS IoT TwinMaker actions

The following is an example of an endpoint policy for AWS IoT TwinMaker. When attached to an endpoint, this policy grants access to the listed AWS IoT TwinMaker actions for the IAM user `iottwinmakeradmin` in the AWS account `123456789012` on all resources.

```
{
   "Statement":[
      {
        "Principal": {
            "AWS": "arn:aws:iam::123456789012:user/role"
                },
         "Resource": "*",
         "Effect":"Allow",
         "Action":[
            "iottwinmaker:CreateEntity",
            "iottwinmaker:GetScene",
            "iottwinmaker:ListEntities"
         ]
        }
```

```
    ]
}
```

Compliance Validation for AWS IoT TwinMaker

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. 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 services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS IoT TwinMaker

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 TwinMaker offers several features to help support your data resiliency and backup needs.

Infrastructure Security in AWS IoT TwinMaker

As a managed service, AWS IoT TwinMaker 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 TwinMaker through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.3 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.

Endpoints and quotas

AWS IoT TwinMaker endpoints and quotas

You can find information about AWS IoT TwinMaker endpoints and quotas in the AWS General Reference.

- For information about service endpoints, see AWS IoT TwinMaker service endpoints.
- For information about quotas, see AWS IoT TwinMaker service quotas.
- For information about API throttling limits, see AWS IoT TwinMaker API throttling limits.

Additional information about AWS IoT TwinMaker endpoints

To connect programmatically to AWS IoT TwinMaker, use an endpoint. If you use an HTTP client, you need to prefix control plane and data plane APIs as follows. However, it is unnecessary to add a prefix to AWS SDK and AWS Command Line Interface commands because they automatically add the necessary prefix.

- Use the `api` prefix for control plane APIs. For example, `api.iottwinmaker.us-west-1.amazonaws.com`.
- Use the `data` prefix for data plane APIs. For example, `data.iottwinmaker.us-west-1.amazonaws.com`.

Document history for the AWS IoT TwinMaker User Guide

The following table describes the documentation releases for AWS IoT TwinMaker.

Change	Description	Date
New service-linked role and new IAM policy	AWS IoT TwinMaker added a new service-linked role, called AWSServiceRoleForIoTTwinMaker. AWS IoT TwinMaker added this new service-linked role to allow AWS IoT TwinMaker to call other AWS services and to sync their resources on your behalf. The new AWSIoTTwinMakerServiceRolePolicy IAM policy is attached to this role, and the policy grants permission to AWS IoT TwinMaker to call other AWS services and to sync their resources on your behalf.	November 17, 2023
Initial release	Initial release of the AWS IoT TwinMaker User Guide	November 30, 2021