AWS IoT TwinMaker

User Guide

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.

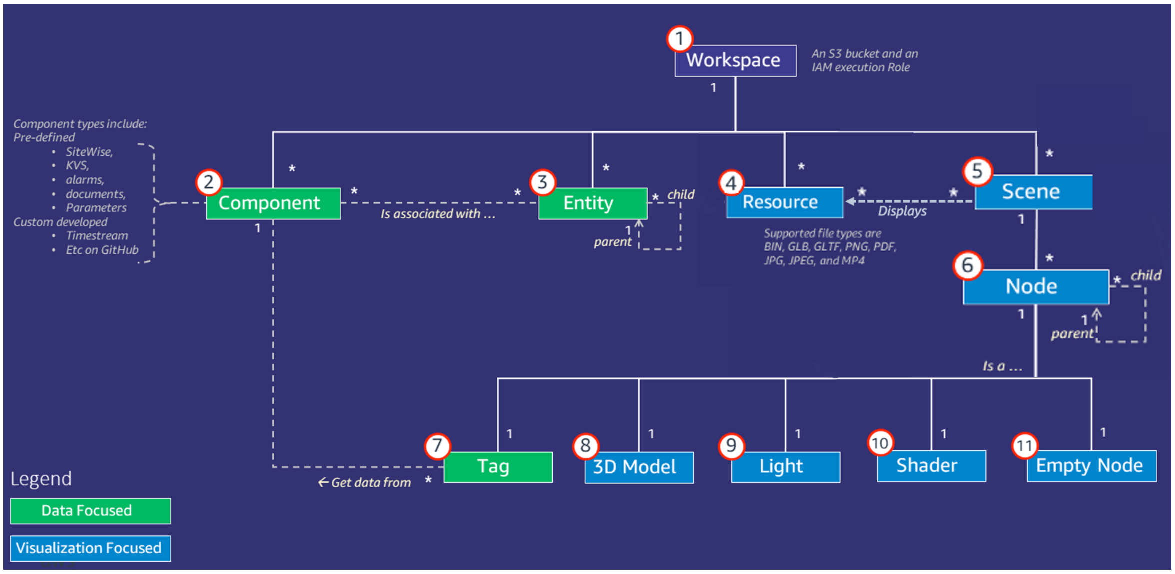
  This knowledge graph contains metadata about the system and can connect to data in different locations. AWS IoT TwinMaker provides built-in connectors for data stored in AWS IoT SiteWise and Kinesis Video Streams. You can also create custom connectors to data stored in other locations.

  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.

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 visualisations. 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 ??? (p. 6).

  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.

```
{
  "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.

```
{
  "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.

```
{
  "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 ??? (p. 18).

> **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 ??? (p. 11).

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

```
{
  "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": "asset ARN"
        },
    {
        "Effect": "Allow",
        "Action": [
            "iotsitewise:DescribeAssetModel"
        ],
        "Resource": "asset model ARN"
        },
    {
      "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 Connnector for Kinesis Video
 Streams"
    ],
    "Effect": "Allow"
},
{
    "Action": [
        "iotsitewise:DescribeAssetModel"
    ],
    "Resource": [
        "AWS IoT SiteWise model resource ARN for the Edge Connnector 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 ??? (p. 18).

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

```
{
  "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/*"
      ]
    },
    {
        "Action": [
            "lambda:invokeFunction"
        ],
        "Resource": [
            "Lambda function ARN"
        ],
        "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.

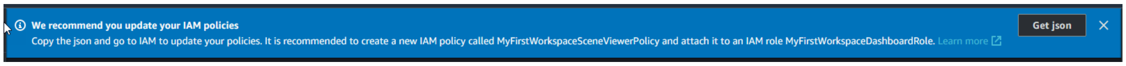
Create a workspace

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 resoucres, 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 ??? (p. 6).
7. Choose **Create Workspace**. The following banner appears at the top of the **Workspaces** page.

   We recommend you update your IAM policies
   Copy the json and go to IAM to update your policies. It is recommended to create a new IAM policy called MyFirstWorkspaceSceneViewerPolicy and attach it to an IAM role MyFirstWorkspaceDashboardRole. Learn more

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.

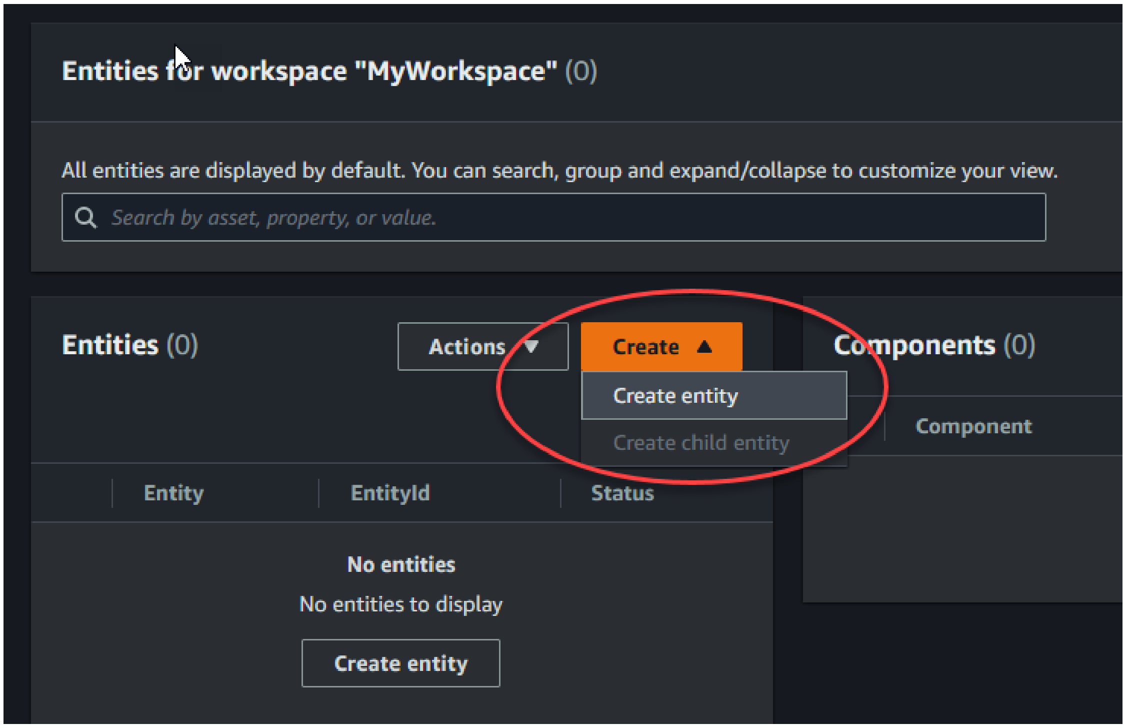
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::iottwinmaker-workspace-workspace-name-lower-case-account-id",
        "arn:aws:s3:::iottwinmaker-workspace-workspace-name-lower-case-account-id/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iottwinmaker:Get*",
        "iottwinmaker:List*"
      ],
      "Resource": [
        "arn:aws:iottwinmaker:us-east-1:account-id:workspace/workspace-name",
        "arn:aws:iottwinmaker:us-east-1:account-id: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 (p. 13).

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**.



Entities for workspace "MyWorkspace" (0)
All entities are displayed by default. You can search, group and expand/collapse to customize your view.
Search by asset, property, or value.
Entities (0)
Actions
Create
Create entity
Create child entity
Components (0)
Component
Entity
EntityId
Status
No entities
No entities to display
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.

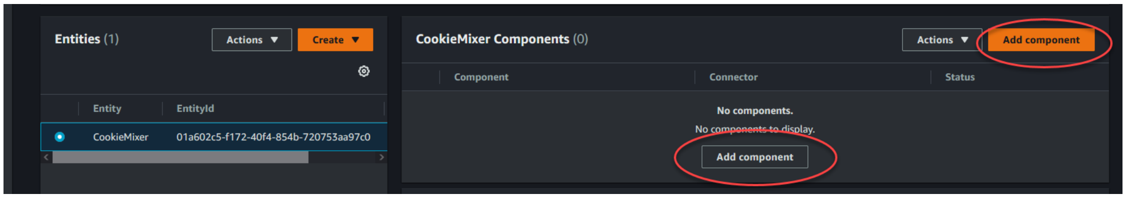
- **Parameters**: Adds a set of key-value properties.

- **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 (p. 78).

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 (p. 18).

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

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

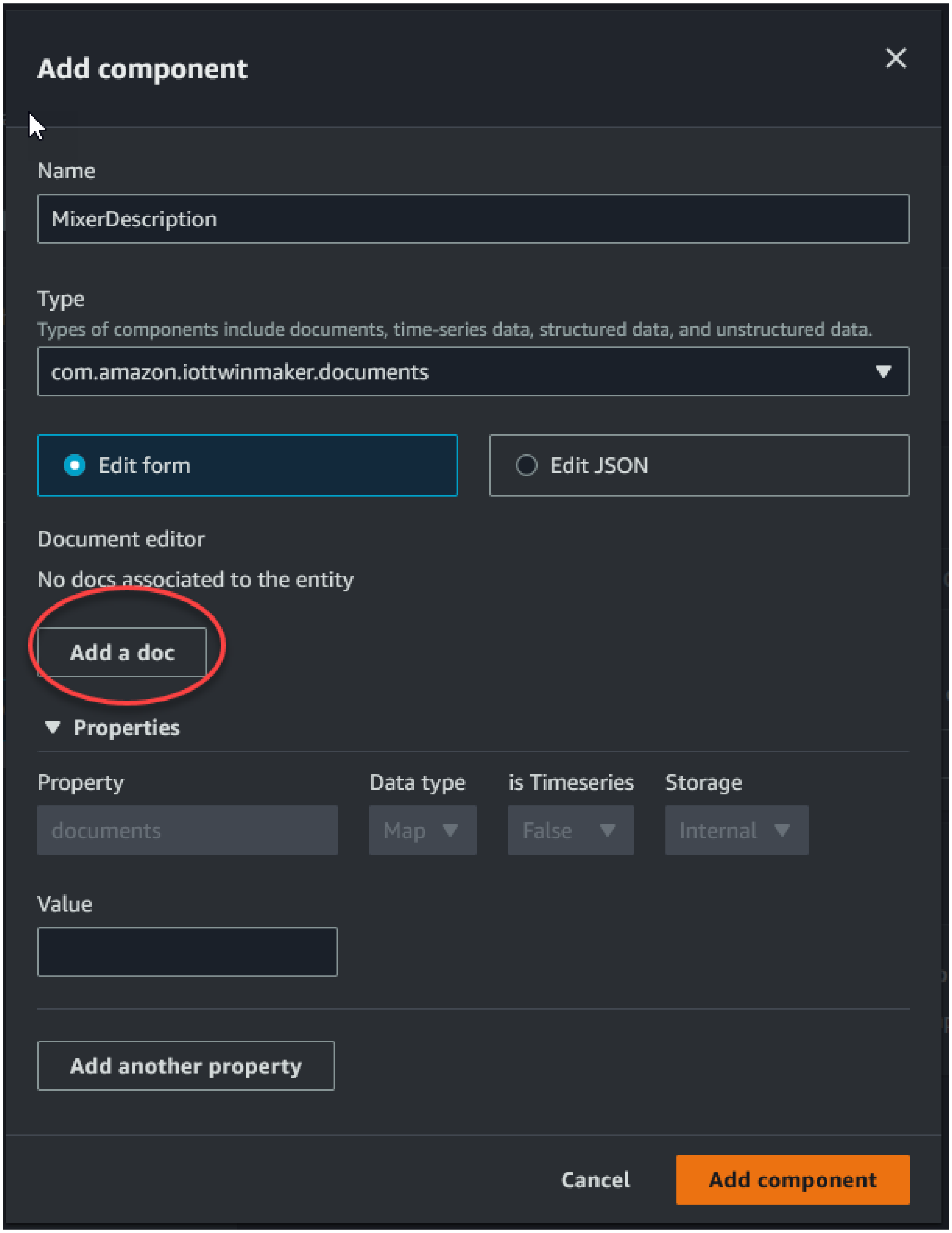


	Entity	EntityId
●	CookieMixer	01a602c5-f172-40f4-854b-720753aa97c0

Component	Connector	Status

No components.
No components to display.

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**. The following window appears.



Add component

Name

MixerDescription

Type

Types of components include documents, time-series data, structured data, and unstructured data.

com.amazon.iottwinmaker.documents

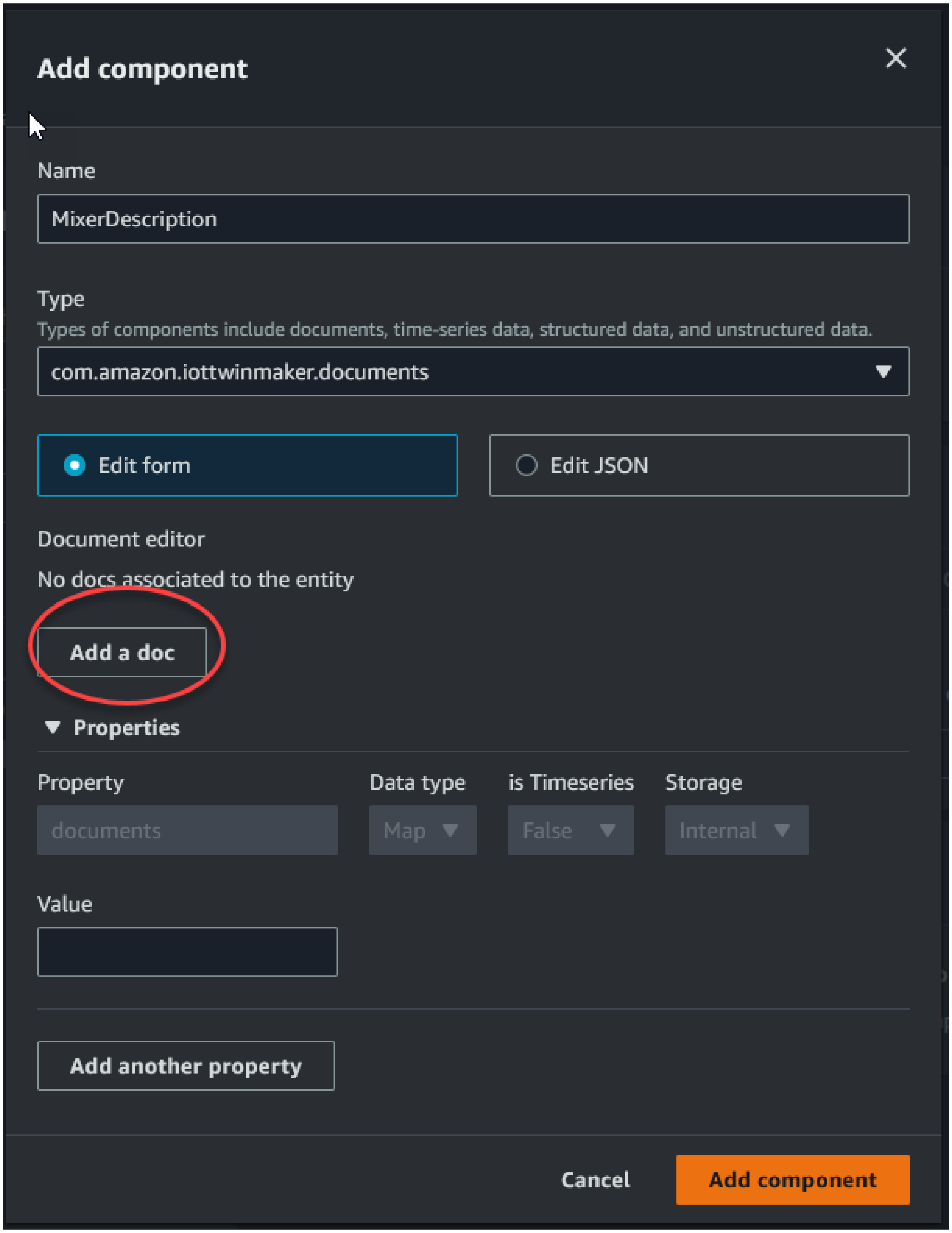
● Edit form ○ Edit JSON

Document editor

No docs associated to the entity

Property	Data type	is Timeseries	Storage
documents	Map	False	Internal

4. Choose **Add a doc**. The following fields appear in the window.



Name: MixerDescription

Type: Types of components include documents, time-series data, structured data, and unstructured data.
com.amazon.iottwinmaker.documents

● Edit form ○ Edit JSON

Document editor

No docs associated to the entity

Property	Data type	is Timeseries	Storage
documents	Map	False	Internal

Value:

Enter values for **Name** and **External Url**. With the documents component, you can store a list of external Urls that contain important information about the entity.

5. 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 (p. 54).

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 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 an administrative user, and use only the root user to perform tasks that require root user access.

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

Choose one way to manage your administra	**To**	**By**	**You can also**
In IAM Identity Center (Recommended)	Use short-term credentials to access AWS. This aligns with the security best practices. For information about best practices, see Security best practices in IAM in the *IAM User Guide*.	Following the instructions in Getting started in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.	Configure programmatic access by Configuring the AWS CLI to use AWS IAM Identity Center (successor to AWS Single Sign-On) in the *AWS Command Line Interface User Guide*.
In IAM (Not recommended)	Use long-term credentials to access AWS.	Following the instructions in Creating your first IAM admin user and user group in the *IAM User Guide*.	Configure programmatic access by Managing access keys for IAM users in the *IAM User Guide*.

Using and creating component types

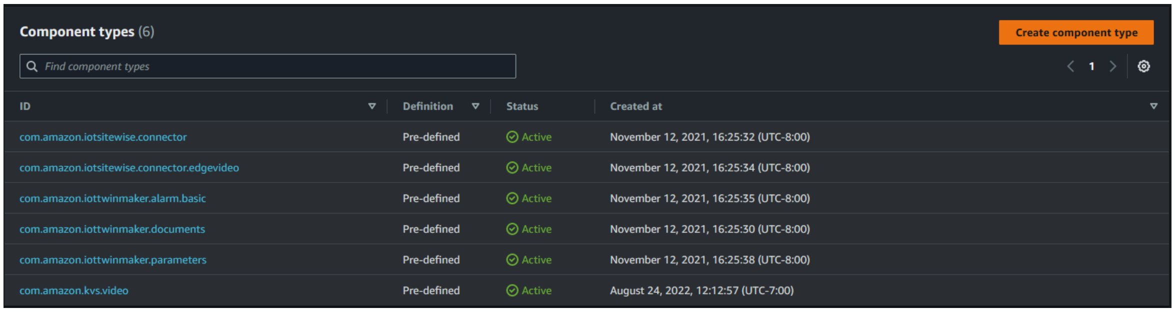
This topic walks you through the values and structures that you use to create an AWS IoT TwinMaker component type. It shows you how to create a request object that you can pass to the CreateComponentType API or by using the component type editor in the AWS IoT TwinMaker console.

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.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.



Component types (6)

ID	Definition	Status	Created at
com.amazon.iotsitewise.connector	Pre-defined	✓ Active	November 12, 2021, 16:25:32 (UTC-8:00)
com.amazon.iotsitewise.connector.edgevideo	Pre-defined	✓ Active	November 12, 2021, 16:25:34 (UTC-8:00)
com.amazon.iottwinmaker.alarm.basic	Pre-defined	✓ Active	November 12, 2021, 16:25:35 (UTC-8:00)
com.amazon.iottwinmaker.documents	Pre-defined	✓ Active	November 12, 2021, 16:25:30 (UTC-8:00)
com.amazon.iottwinmaker.parameters	Pre-defined	✓ Active	November 12, 2021, 16:25:38 (UTC-8:00)
com.amazon.kvs.video	Pre-defined	✓ Active	August 24, 2022, 12:12:57 (UTC-7:00)

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 SiteWise

Parameter	Description
	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 additional 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 ??? (p. 21).

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 asset (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": {
        "telemetryAssetType": {
            "dataType": {
                "type": "STRING"
            },
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "telemetryAssetId": {
            "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": {
        "telemetryAssetType": {
            "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": {
        "telemetryAssetType": {
            "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": {
        "telemetryAssetType": {
            "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 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 Lambda connectors with the AWS IoT TwinMaker data connector interface so that AWS IoT TwinMaker can invoke your connector when necessary.

Topics

Developing AWS IoT TwinMaker time-series data connectors

In this topic you will learn 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 AWS IoT TwinMaker samples Github repository .

Topics

AWS IoT TwinMaker time-series data connector prerequisites

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

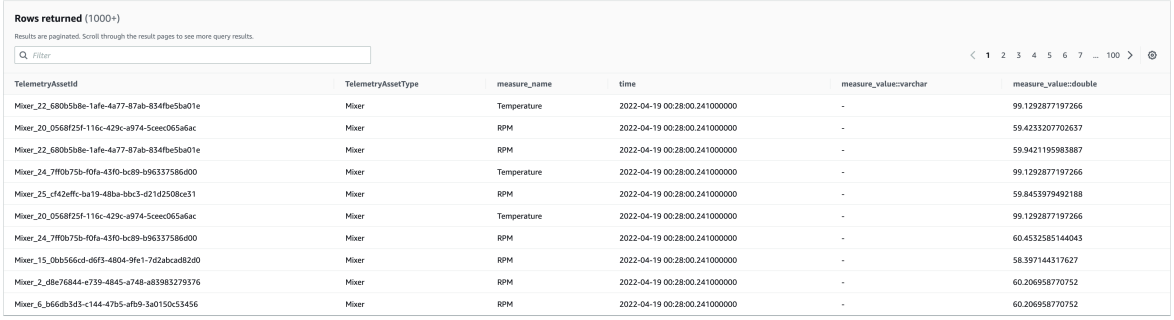
- You have created an AWS IoT TwinMaker workspace.
- You have created AWS IoT TwinMaker component types.
- You have created AWS IoT TwinMaker entities.
- (Optional) we recommend reading Using and creating component types.
- (Optional) we recommend reading AWS IoT TwinMaker data connector interface to get a general understanding of TwinMaker Data-connectors.

> **Note**
> If you want to see 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 assets 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 an Timestream database. You want to be able to view and organize the measurement data in TwinMaker with minimal overhead. You can accomplish this task by using a time-series data connector. Below is an example telemetry table, which is being 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.

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 above result.

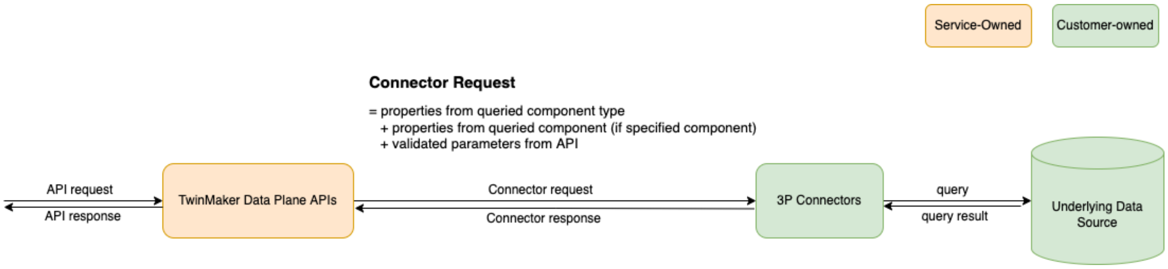
Time-series data connector data flow

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

TwinMaker uses Lambda functions to access and resolve queries from data sources and return results of those queries. The lambda functions uses 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 image below outlines the basic data flow described in the previous paragraphs.


Service-Owned
Customer-owned
Connector Request
= properties from queried component type
+ properties from queried component (if specified component)
+ validated parameters from API
API request
API response
TwinMaker Data Plane APIs
Connector request
Connector response
3P Connectors
query
query result
Underlying Data Source

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. If you are unfamiliar with defining component types, we recommend you read the documentation for Using and creating component types.

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

   Below is a Timestream template component type with one property:

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

   - The `external-id` property is used to identify the unique key of the asset in the corresponding data source. This property is used in data connector as a filter condition to only query values of the given asset. Additionally, if you include the `external-id` property value in the data plane API response, then the client side takes the id and can perform a reverse look-up if necessary.
   - The `lambdaarn` field is used to identify the lambda function the component type engages with.
   - The `isRequiredInEntity` flag is used to enforce the id creation. This flag is required so that when the component is created, the asset id will also be instantiated.
   - The `TelemetryAssetId` is added to the component type as an `external-id`. This is done that the asset 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 componenet and attacht it to the entity you wish to retrive data from. The following steps detial the process of creating that component:

   a. Navigate to the AWS IoT TwinMaker console.
   b. Select and open the same workspace you created the component types in.
   c. Navigate to the entity page.
   d. Create a new entity, or select the an already made entity from the table.
   e. Once you have selected the entity you wish to use, select the **Add component** button, 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. A json request template that would sent to the Lambda is shown below:

```
{
  "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": {
      "telemetryAssetType": {
          "definition": {
              "dataType": {
                  "type": "STRING"
              },
              "isExternalId": false,
              "isFinal": false,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": false,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "Mixer"
          }
      },
      "telemetryAssetId": {
          "definition": {
              "dataType": {
                  "type": "STRING"
              },
              "isExternalId": true,
              "isFinal": true,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": true,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "asset_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 you want timestream measurements for.
- The `startDateTime`, `startTime`, `EndDateTime`, and `endTime` fields are used to specify a time range for the request. This determines the sample range for the measurements returned.
- The `entityId` is the name of the entity you are querying data from.
- The `componentName` is the name of the component you are querying data from.
- Use the `orderByTime`field to organize the order in which the results are displayed.

We can see from the request above, we would expect to get series of samples for the selected properties during the given time window for the given asset, with the selected time order. The response statement can be summarized as:

```
{
  "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": {"telemetryAssetType": {"dataType": {"type": "STRING"
            },
            "isExternalId": false,
            "isStoredExternally": false,
            "isTimeSeries": false,
            "isRequiredInEntity": true
        },
        "telemetryAssetId": {"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 will follow a single request to connector by giving 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": {
      "telemetryAssetType": {
          "definition": {
              "dataType": {
                  "type": "STRING"
              },
              "isExternalId": false,
              "isFinal": false,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": false,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "Mixer"
          }
      },
      "telemetryAssetId": {
          "definition": {
              "dataType": {
                  "type": "STRING"
              },
              "isExternalId": true,
              "isFinal": true,
              "isImported": false,
              "isInherited": false,
              "isRequiredInEntity": true,
              "isStoredExternally": false,
              "isTimeSeries": false
          },
          "value": {
              "stringValue": "asset_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:

```
{
  "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, which means with five properties in the query and a page size of 100, if there is 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 refer to the Snowflake connector sample on Github.

Improving you 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 will wait for the Lambda to return a response. If the connector implementation throws an exception then, IoT TwinMaker will translate the exception type to be `ConnectorFailure`, making the API client aware that an issue happened inside the connector.

Handling pagination

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

When the new token is returned to AWS IoT TwinMaker through connector response interface, the token will be encrypted before being returned to API client. And when the token is included in another request, AWS IoT TwinMaker will decrypt it before forwarding to Lambda connector. But 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 and you can check testing Lambda in the Lambda console or testing Lambda locally in the CDK.

For more information on testing your Lambda functions see the following documentation 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.

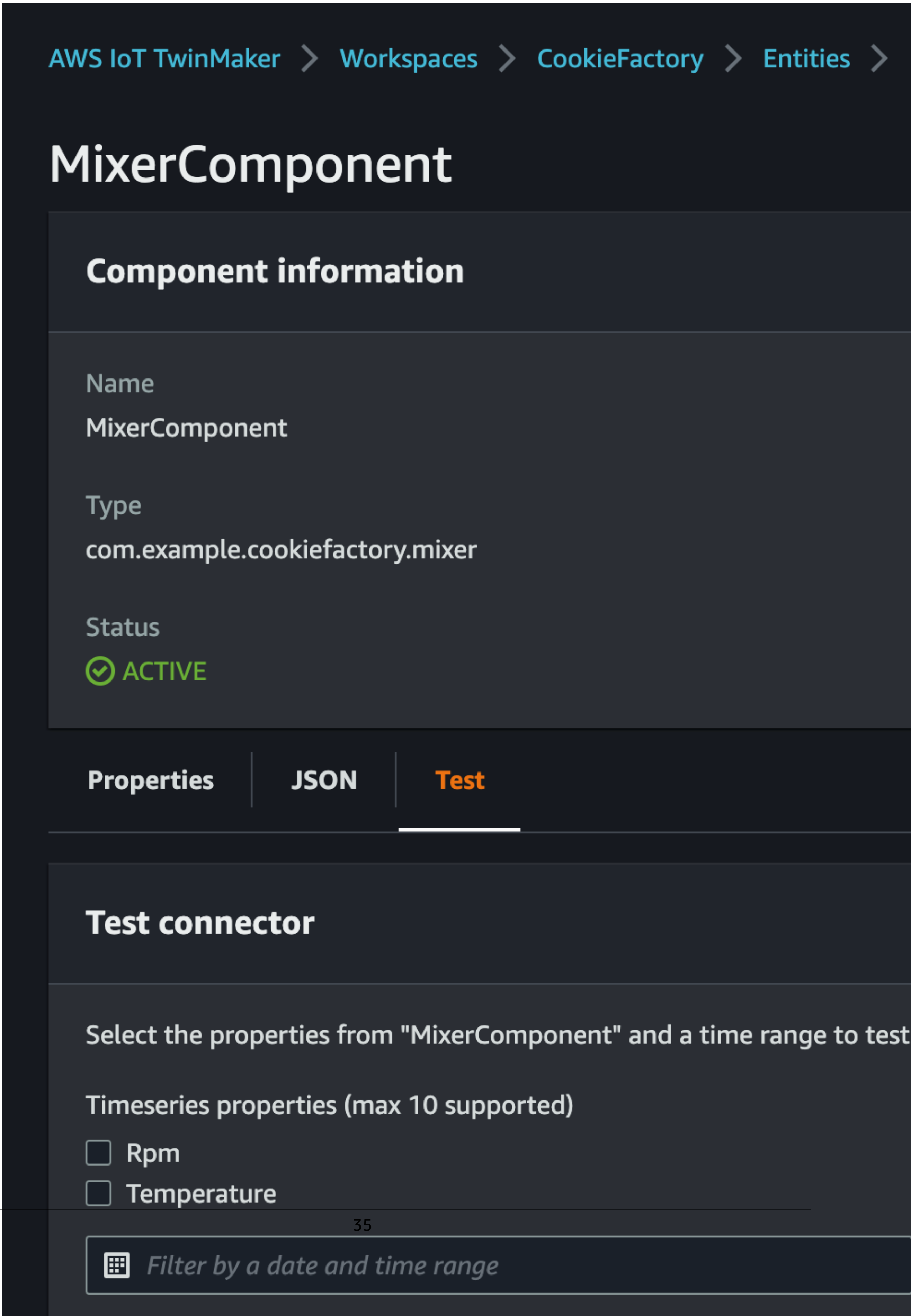
Creating AWS IoT TwinMaker resources

Once the Lambda is implemented, you can create AWS IoT TwinMaker resources like component types, entities and components through the AWS IoT TwinMaker console or API.

Note
If you follow the setup instructions of the Github sample, all AWS IoT TwinMaker resources will be made 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.



AWS IoT TwinMaker > Workspaces > CookieFactory > Entities >

MixerComponent

Component information

Name
MixerComponent

Type
com.example.cookiefactory.mixer

Status
ACTIVE

Test connector

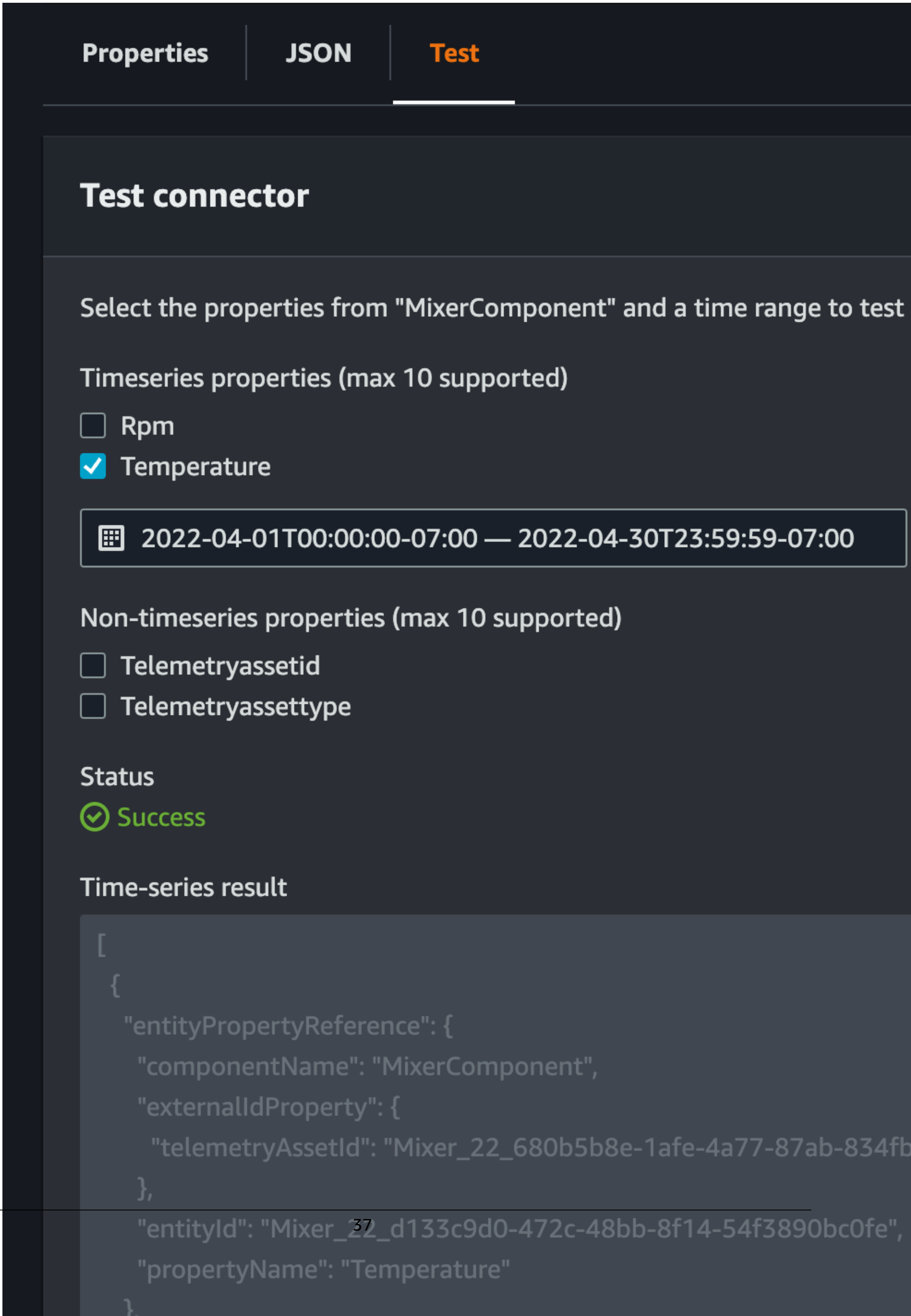
Select the properties from "MixerComponent" and a time range to test

Timeseries properties (max 10 supported)

□ Rpm
□ Temperature

Filter by a date and time range

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.



Test connector

Select the properties from "MixerComponent" and a time range to test

Timeseries properties (max 10 supported)

□ Rpm
☑ Temperature

2022-04-01T00:00:00-07:00 — 2022-04-30T23:59:59-07:00

Non-timeseries properties (max 10 supported)

□ Telemetryassetid
□ Telemetryassettype

Status

✓ Success

Time-series result

```
[
 {
  "entityPropertyReference": {
   "componentName": "MixerComponent",
   "externalIdProperty": {
    "telemetryAssetId": "Mixer_22_680b5b8e-1afe-4a77-87ab-834fb
   },
   "entityId": "Mixer_22_d133c9d0-472c-48bb-8f14-54f3890bc0fe",
   "propertyName": "Temperature"
  }
```

What's next

You can now setup a 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 it fits your use case.

AWS IoT TwinMaker Cookie factory example time-series connector

The complete code of the cookie factory Lambda 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. There are multiple ways to test your Lambda and you can check testing Lambda in the Lambda console or testing Lambda locally in the 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 assets under the same asset type share the same measurements, so we can define the measurements schema in the component type. As an example, the mixer type is defined as following.

```
{
    "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 asset might have measurements in a timestream database, maintenance records in SQL database, or alarm data in alarm systems. Creating multiple components and associating them with an entity will link different data sources to the entity, and populate the entity-component graph. In this context, each component will need an `external-id` property, to identify the unique key of the asset in the corresponding data source. Specifying the `external-id` property will have two benefits, first it can be used in data connector as a filter condition to only query values of the given asset. Second if you include the `external-id` property value in the data plane API response, then the client side takes the id and can perform a reverse look-up if necessary.

So if you add the `TelemetryAssetId` to the component type as an `external-id`, it identifies the asset in the `TimeStream` table.

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

The `TelemetryAssetType`code> would be an optional property in the component type if it was 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 `TelemetryAssetType` is defined and you extract common properties like `TelemetryAssetId` and `TelemetryAssetType` 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 as shown in the json example below.

```
{
    'workspaceId': 'CookieFactory',
    'selectedProperties': ['Temperature'],
    'startDateTime': 1648796400,
    'startTime': '2022-04-01T07:00:00.000Z',
    'endDateTime': 1650610799,
    'endTime': '2022-04-22T06:59:59.000Z',
    'properties': {
        'telemetryAssetId': {
            '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 is to query historical measurement data for a given entity. AWS IoT TwinMaker provides a component-properties map, and you should put an instantiated value for asset id. If we plan to handle the component type level query (which is common for alarm use case), for example, return alarm status of all assets in the workspace, then the properties map will have component type properties definitions.

For the most straightforward case, as we can see from the request above, we would like to get series of temperature samples during the given time window for the given asset, with ascending time order. The query statement can be summarized as

```
...
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 TelemetryAssetId = '{telemetry_asset_id}'
    AND measure_name = '{selected_property}'
    ORDER BY time {request.orderByTime}
...
```

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

The Schema initializer connector is used in the component type or entity lifecycle to fetch the component type or component properties from the underlying data source. With the schema initializer, properties of the component type or component are automatically imported without explicitly calling an API action to set up properties schemas.

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 PropertyResponse. For more information, see PropertyResponse.

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 PropertyRequest. For more information, see PropertyRequest.

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
      "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
                            }
                        ]
                    }
                }
            ]
        }
    ]
}
```

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 CAD assets to glTF for use with AWS IoT TwinMaker
- Optimize your 3D models for web content
- Optimizing scenes for better WebGL performance

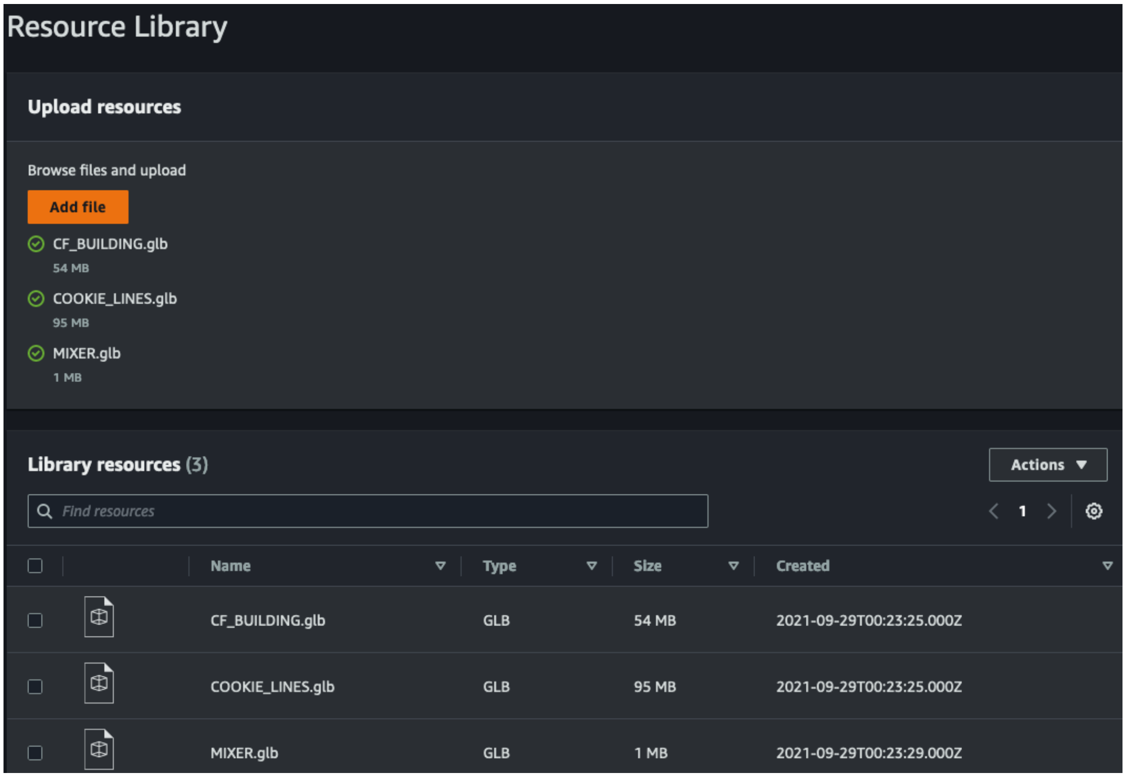
Upload resources to the AWS IoT TwinMaker Resource Library

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 through the Resource Library console page.

Upload files to the Resource Library through the console

To add files to the Resource Library by using the console, follow these steps.

1. In the left navigation menu, to open the **Upload resources** pane, choose **Resource Library**.
2. Choose **Add file** and select the files you want to upload.



Resource Library

Upload resources

Browse files and upload

✓ CF_BUILDING.glb
54 MB

✓ COOKIE_LINES.glb
95 MB

✓ MIXER.glb
1 MB

Library resources (3)

□		Name	Type	Size	Created
□		CF_BUILDING.glb	GLB	54 MB	2021-09-29T00:23:25.000Z
□		COOKIE_LINES.glb	GLB	95 MB	2021-09-29T00:23:25.000Z
□		MIXER.glb	GLB	1 MB	2021-09-29T00:23:29.000Z

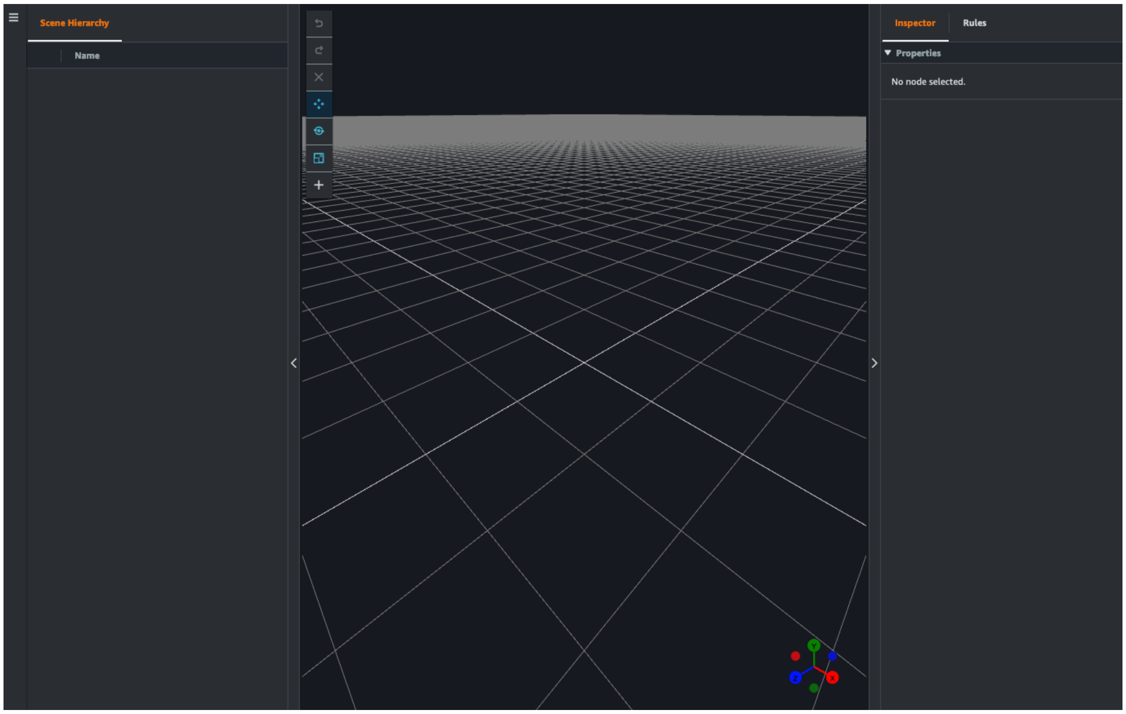
Create your scenes

In this section, you'll set up a scene so that you can edit your digital twin. You'll need to create one or more scenes before you create your digitial twin . There are two main approaches to creating a scene: you can import a single glTF file that has the entire model for your site. Or, you can import an environment model, such as a building or a space, and then import separate glTF files for each piece of equipment and position them relative to their physical location.

> **Note**
> Before you create a scene, you must have created 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 **Scene Composer**.
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. When you're ready to create the scene, choose **Create scene**. The new scene opens and is ready for you to work with it.

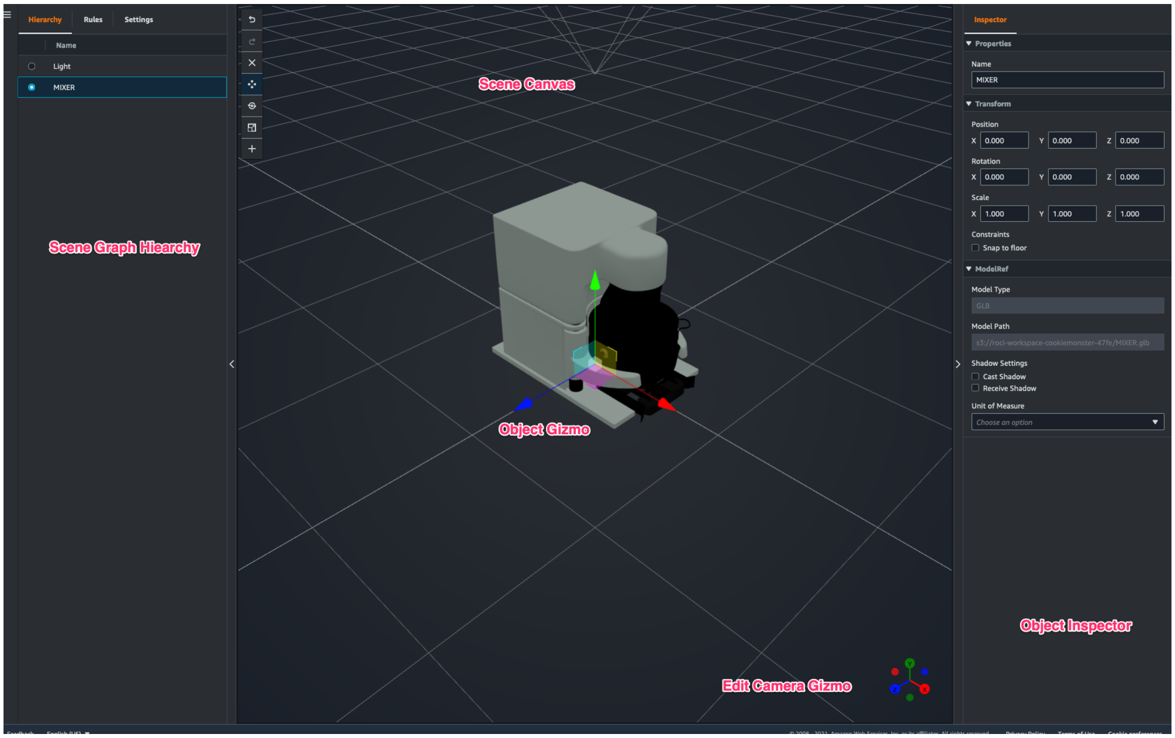


No node selected.

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.



▼ Properties

Name: MIXER

▼ Transform

Position: X 0.000 Y 0.000 Z 0.000

Rotation: X 0.000 Y 0.000 Z 0.000

Scale: X 1.000 Y 1.000 Z 1.000

Constraints

□ Snap to floor

▼ ModelRef

Model Type: GLB

Model Path: s3://roci-workspace-cookiemonster-47fe/MIXER.glb

Shadow Settings

□ Cast Shadow

□ Receive Shadow

Unit of Measure: Choose an option

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.

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.

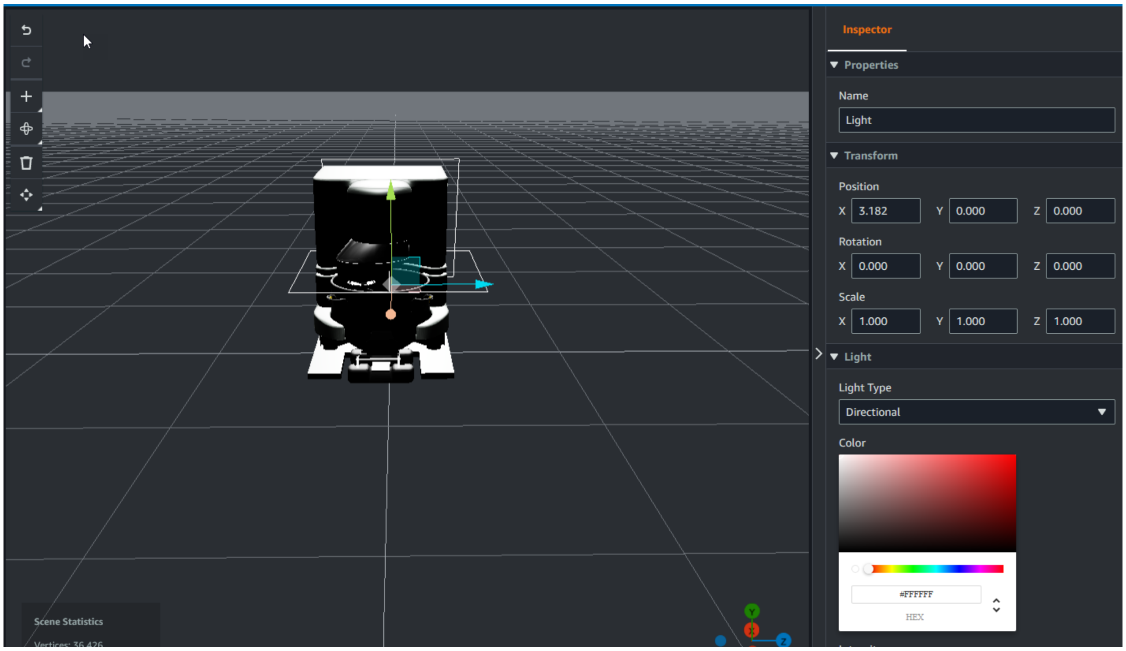
Add models to your scenes

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 (p. 55).

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.



Inspector

▼ Properties

Name: Light

▼ Transform

Position: X 3.182 Y 0.000 Z 0.000

Rotation: X 0.000 Y 0.000 Z 0.000

Scale: X 1.000 Y 1.000 Z 1.000

▼ Light

Light Type: Directional

Color: #FFFFFF HEX

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

Add color-overlay augmented UI widgets to your scene

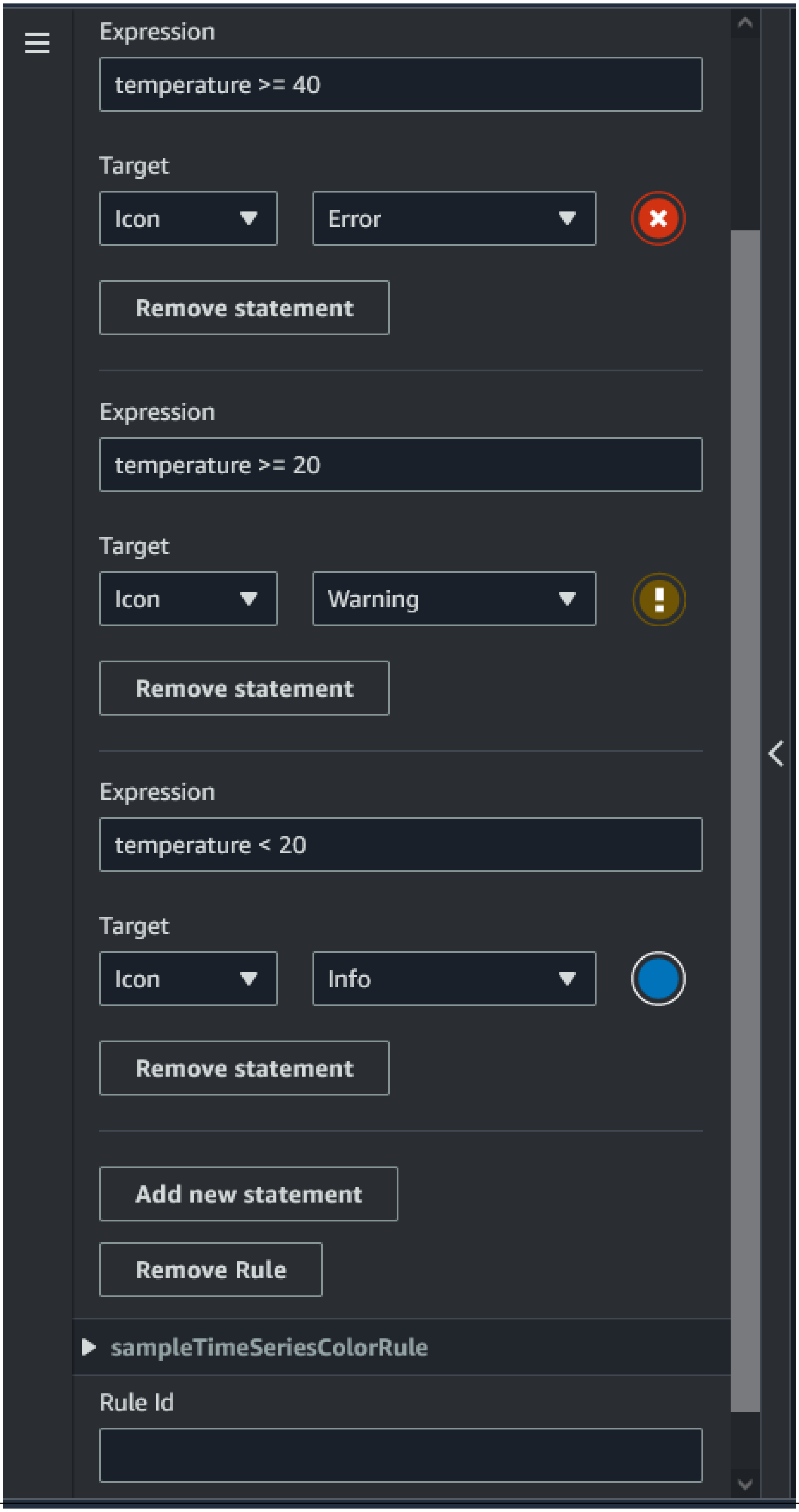
Color overlay widgets can change the color of an object or display icons around objects, under conditions that you define. For example, you can create a color widget that changes the color of a cookie mixer in your scene based on the mixer's temperature data.

Use the following procedure to add color overlay 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 Color Overlay.
2. To add a new visual rule group, in the Inspector panel for the object of the Visual Group ID, choose **ColorRule**.
3. Select the entityID, ComponentName, or PropertyName you want to bind the color overlay to, and then choose **CreateFrameLabel**.

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 color overlay. There are sample rules provided, but you can also create your own. The following example shows visual rule.



Expression

temperature >= 40

Target

Icon ▼ Error ▼

Expression

temperature >= 20

Target

Icon ▼ Warning ▼

Expression

temperature < 20

Target

Icon ▼ Info ▼

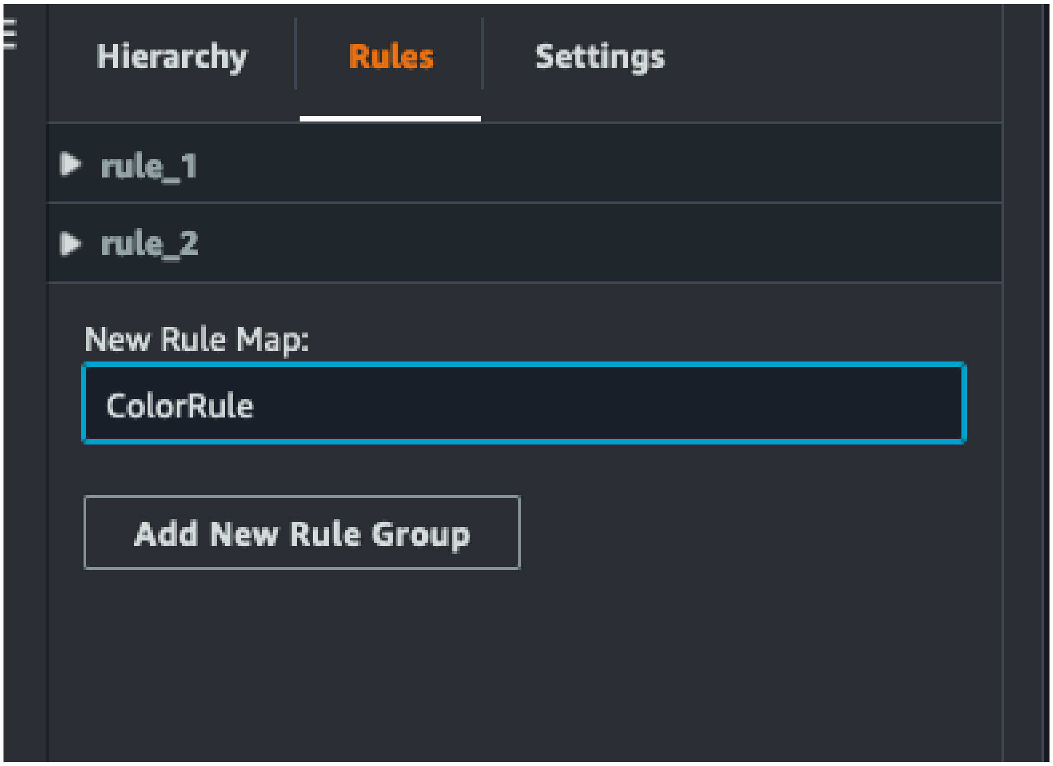
▶ sampleTimeSeriesColorRule

Rule Id

In the image **rule_1**, it shows a rule for when the data property **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 temperature, where the temperature is less than 20. Use the following syntax for rule expressions: Less than is **<**, greater than is **>**, less than equal is **=>**, greater than equal is **=<**, and equal is **==**.
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**.

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
`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, WorkspaceId. **Units**: Count
`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.

Metric	Description
	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 AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

AWS IoT TwinMaker Grafana dashboard integration

AWS IoT TwinMaker supports Grafana integration through an application plugin. You can use Grafana version 8.2.0 and later 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

> **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 (p. 67).

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 CROS, 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. Selet **Save changes** to finish the CORS configuration.

For more inforamtion 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 (p. 68)
- Self-managed Grafana (p. 69)

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?.

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 (p. 69).

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 IAM 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 (p. 69).

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.

```
{
  "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:region:accountId:workspace/workspaceId",
        "arn:aws:iottwinmaker:region:accountId: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 assets 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 (p. 74).

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:

```
{
  "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:region:accountId:workspace/workspaceId",
        "arn:aws:iottwinmaker:region:accountId: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:

```
{
  "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:region:accountId:workspace/workspaceId",
        "arn:aws:iottwinmaker:region:accountId: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 (p. 74).

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:

```
{
  "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 (p. 68).

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:

```
{
  "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:region:accountId:workspace/workspaceId",
        "arn:aws:iottwinmaker:region:accountId: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 (p. 70) 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 `YourWorkspaceIdDashboardPolicy`, 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.

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 (p. 67).

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 (p. 67).

Important
To view video in your Grafana dashboard, you must make sure the Grafana datasoucres have the proper IAM permissions. To create the required role and policy see Creating a dashboard IAM role (p. 69).

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 pannel**, and then choose **Add an empty panel**.
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 Kinesis Data Analytics and performing Flink SQL queries in a Zeppelin notebook in Kinesis Data Analytics. 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 ??? (p. 18).
2. An understanding of Studio notebooks with Kinesis Data Analytics 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 Kinesis Data Analytics 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.

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. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.

- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form 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 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 adresses 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

- How AWS IoT TwinMaker works with IAM (p. 87)
- Identity-based policy examples for AWS IoT TwinMaker (p. 91)
- Troubleshooting AWS IoT TwinMaker identity and access (p. 93)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS IoT TwinMaker.

Service user – If you use the AWS IoT TwinMaker service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT TwinMaker features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS IoT TwinMaker, see Troubleshooting AWS IoT TwinMaker identity and access (p. 93).

Service administrator – If you're in charge of AWS IoT TwinMaker resources at your company, you probably have full access to AWS IoT TwinMaker. It's your job to determine which AWS IoT TwinMaker features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT TwinMaker, see How AWS IoT TwinMaker works with IAM (p. 87).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS IoT TwinMaker. To view example AWS IoT TwinMaker identity-based policies that you can use in IAM, see Identity-based policy examples for AWS IoT TwinMaker (p. 91).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *AWS General Reference*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center (successor to AWS Single Sign-On). You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS IoT TwinMaker in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. By default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS IoT 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 (p. 87)	Yes
Resource-based policies (p. 88)	No
Policy actions (p. 88)	Yes
Policy resources (p. 89)	Yes
Policy condition keys (p. 89)	Yes
ACLs (p. 90)	No
ABAC (tags in policies) (p. 90)	Partial
Temporary credentials (p. 90)	Yes
Principal permissions (p. 91)	Yes
Service roles (p. 91)	Yes
Service-linked roles (p. 91)	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 *IAM 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 Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

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 (p. 91).

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. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies 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. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

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 (p. 91).

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. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

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 (p. 91).

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 (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

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 (p. 91).

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. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using Temporary credentials with AWS IoT TwinMaker

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AWS IoT TwinMaker

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS IoT TwinMaker in the *Service Authorization Reference*.

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 Creating 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 IAM 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. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies for users that require them.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies 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

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 AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or root users in your account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS IoT 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 (IAM 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 you'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 *IAM 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 the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not 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, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `iottwinmaker:`*`GetWidget`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS IoT 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 view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access AWS IoT TwinMaker

To allow others to access AWS IoT TwinMaker, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in AWS IoT TwinMaker.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my AWS IoT 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 (p. 87).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

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. 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

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

To learn whether or other AWS services are within the scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.

- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  Note
  Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

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) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Document history for the AWS IoT TwinMaker User Guide

The following table describes the documentation releases for AWS IoT TwinMaker.

Change	Description	Date
Initial release (p. 101)	Initial release of the AWS IoT TwinMaker User Guide	November 30, 2021