Universität der Bundeswehr München

AWS-Based Deployment of a Control Dashboard and Digital Twin for a Model Factory: A Step-by-Step Guide

Thema: AWS-Based Deployment of a Control Dashboard and Digital Twin for a Model Factory: A Step-by-Step Guide

Autor: Fabian Bellgardt

Betreuer: Prof. Dr. rer. nat. Antje Neve (Universität der Bundeswehr München)
Prof. Bryan Donyanavard (San Diego State University)

Abstract

This document serves as an appendix to a master's thesis and is not a thesis itself. It provides a detailed, step-by-step guide for reproducing the Amazon Web Services (AWS)-based control dashboard and digital twin developed in the main thesis work. The guide enables readers to replicate the setup of a vendor-independent, cloud-based monitoring and control system for a Fischertechnik model factory. The first part outlines the creation of a Node-RED dashboard hosted on an AWS Elastic Compute Cloud (EC2) instance, including the configuration of Virtual Private Cloud (VPC) networking components, EC2 setup, Docker-based Node-RED deployment, and secure integration with the factory via AWS Internet of Things (IoT) Core. The second part details the implementation of a digital twin using AWS IoT Core, IoT SiteWise, IoT TwinMaker, and Amazon Managed Grafana. This includes Message Queuing Telemetry Transport (MQTT) message ingestion, asset modeling, rule-based data routing, Three-Dimensional (3D) scene creation, annotation, and real-time data visualization. The resulting documentation offers a reproducible, extensible framework for Industry 4.0 applications, allowing future users to rebuild the system in full detail.

Contents

1 Step-by-Step Guide to Creating a Node-RED Dashboard on AWS EC2

This section provides detailed step-by-step instructions on how to deploy a Node-RED based dashboard using AWS services and how to connect it to the Fischertechnik factory in order to visualize and control the factory via the dashboard.

1.1 Create a VPC

A VPC is a logically isolated network environment within AWS. It allows you to define and control a virtual network that is logically separated from other AWS customers, giving you full control over networking components such as IP address ranges, subnets, routing tables, internet gateways, and security settings. For this VPC, a couple of other things have to be implemented, like a internet gateway and a security group.

1.1.1 Create a Internet Gateway

A internet gateway is needed to connect the VPC with the internet. It can later be attached to the VPC.

Figure 1.1: This screenshot shows the where and how an internet gateway can be created in AWS.

1.1.2 Create the VPC and attach the Internet Gateway to it

The VPC has to be created manually and the internet gateway has to be attached.

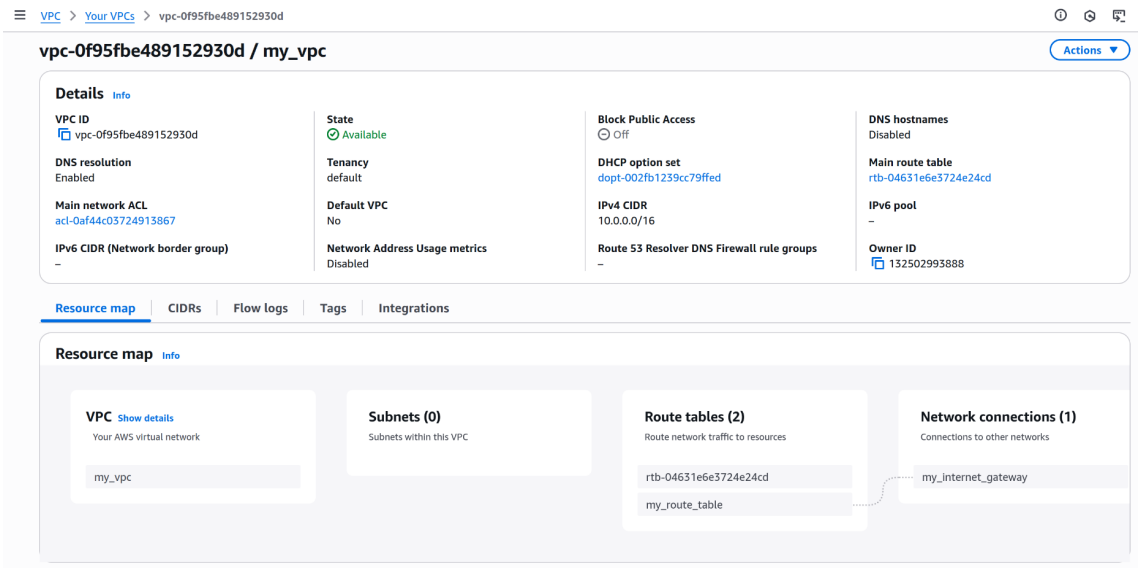

Figure 1.2: This screenshot shows where and how to create the VPC itself.

The internet gateway has to be attached to the VPC. The next screenshot shows how.

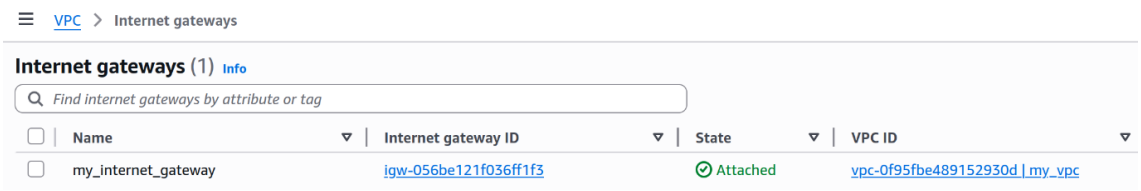

Figure 1.3: This screenshot shows how to attach a internet gateway to the VPC.

1.1.3 Create and Connect a Route Table

In an AWS VPC, a route table defines how network traffic is directed within the VPC and to external destinations. Each subnet is associated with a route table that contains rules mapping Internet Protocol (IP) address ranges to targets such as internet gateways or Network Address Translation (NAT) gateways.

Route tables are essential for enabling communication between subnets and with the internet. For instance, a public subnet requires a route to an internet gateway, while a private subnet may use a NAT gateway for secure outbound access. Proper route configuration ensures controlled and reliable network behaviour in cloud environments.

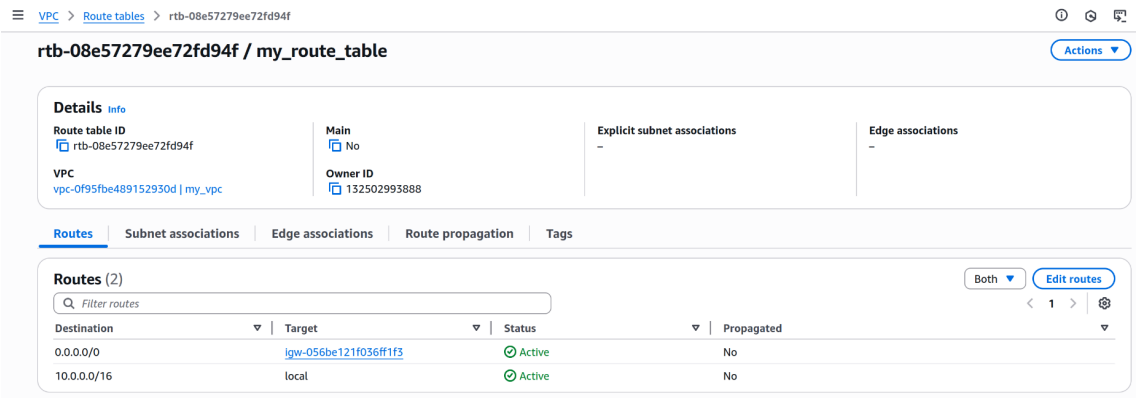

Figure 1.4: This screenshot shows how to create and attach a route table to the VPC.

1.1.4 Create a Security Group

A security group in an AWS VPC acts as a virtual firewall that controls inbound and outbound traffic for resources such as EC2 instances. Unlike traditional firewalls, security groups are stateful, meaning that return traffic is automatically allowed, regardless of inbound or outbound rules.

Security groups define access rules based on protocol, port, and source or destination IP addresses. They are essential for enforcing fine-grained network security, ensuring that only trusted traffic can reach or leave specific resources within the VPC.

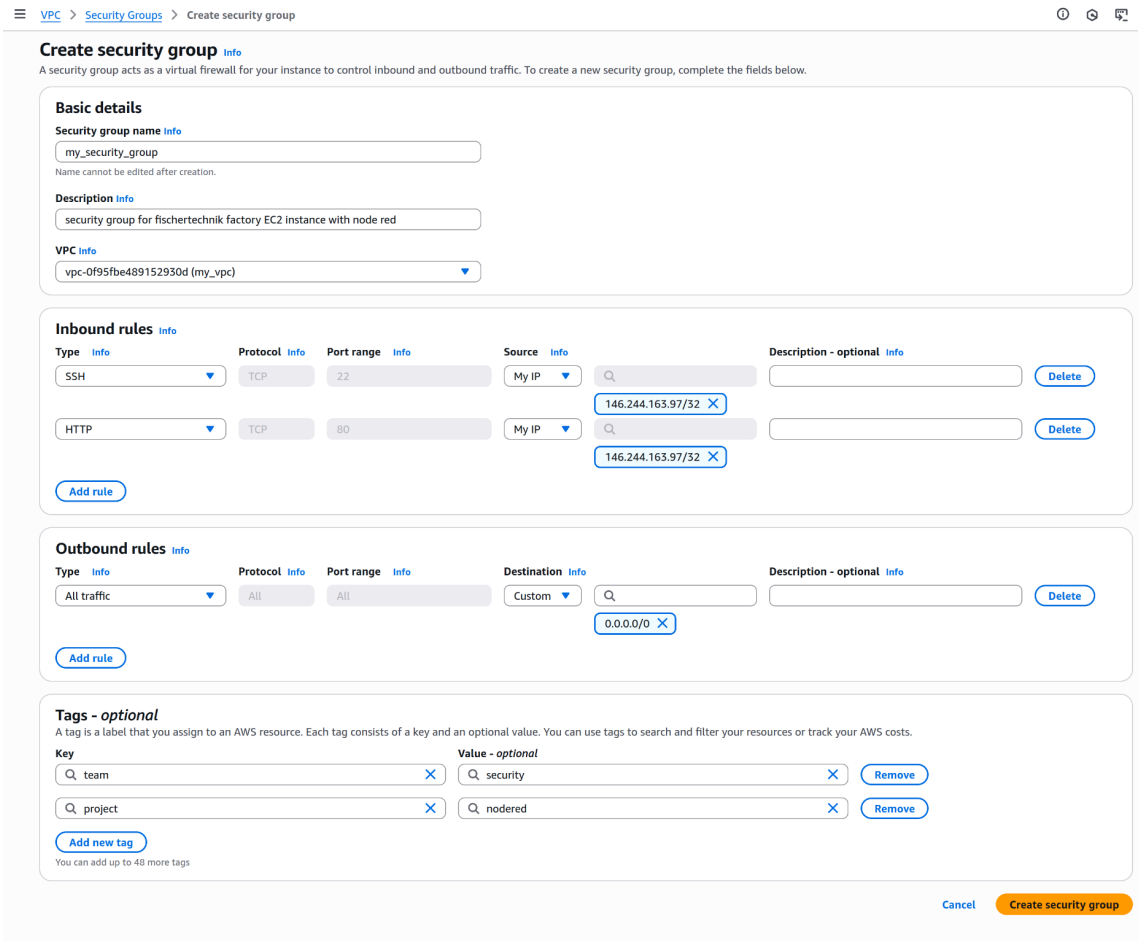

Figure 1.5: This screenshot shows how and where to create a security group.

It is important to use Hypertext Transfer Protocol (HTTP) for the inbound rule, since the used Port 80 has to be matching. Also Secure Shell (SSH) is important to create a connection using the Linux terminal and configure and install software on the EC2 that runs on this VPC.

1.2 Create EC2 Instance

An Amazon EC2 instance running Amazon Linux serves as the hosting environment for the Node-RED application, which provides the user interface to monitor and control the Fischertechnik factory. The instance is deployed within a VPC to ensure secure and isolated network communication. Node-RED runs inside a Docker container on the EC2 instance. The dashboard is accessible via a web browser and enables global, vendor-independent access to the factory system, replacing the need for the proprietary Fischertechnik Cloud.

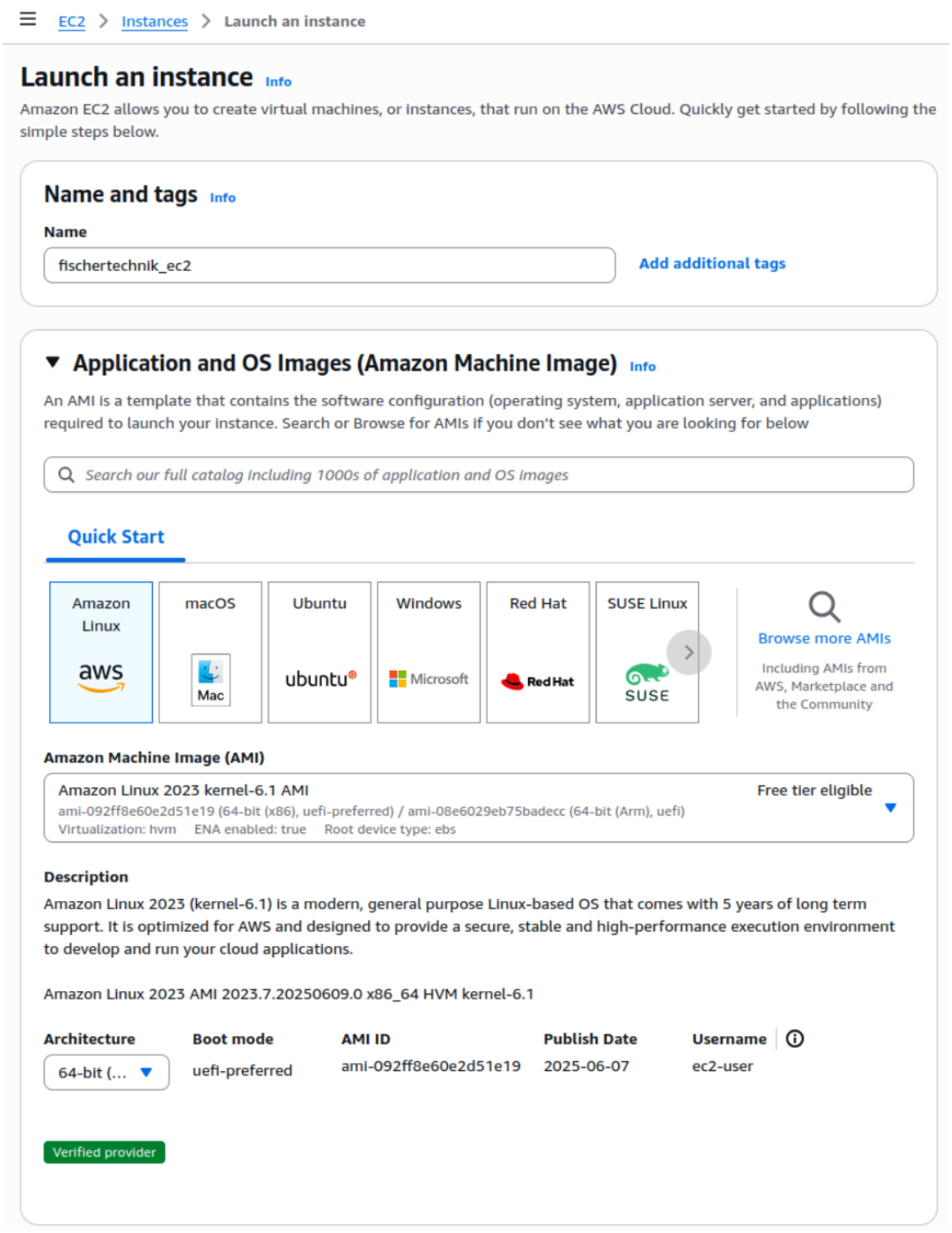

Figure 1.6: This screenshot shows how and where to create a EC2 instance.

A name has to be set, and the Operating System (OS) has to be specified. The Machine

and the architecture of the system can also be set.

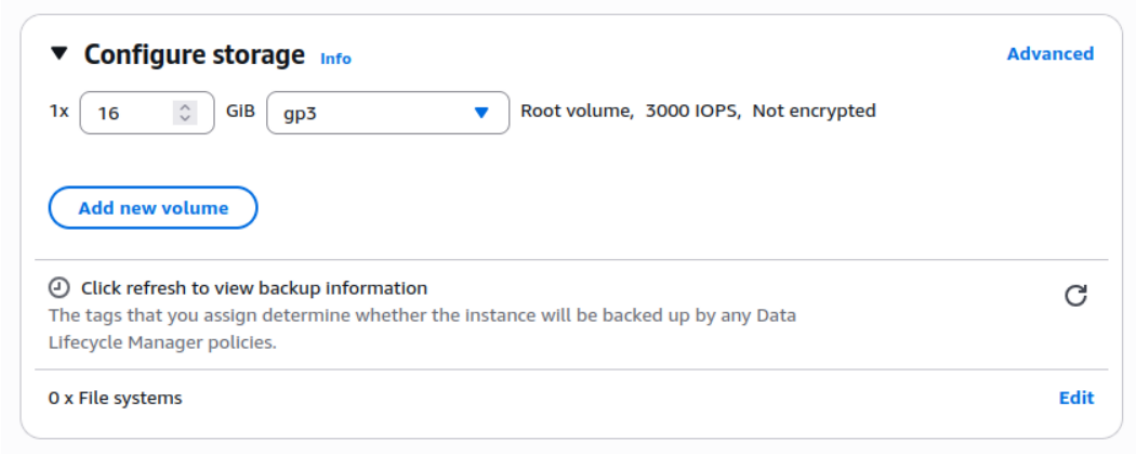

Figure 1.7: This screenshot shows how to set the storage needed for the OS on the EC2 instance.

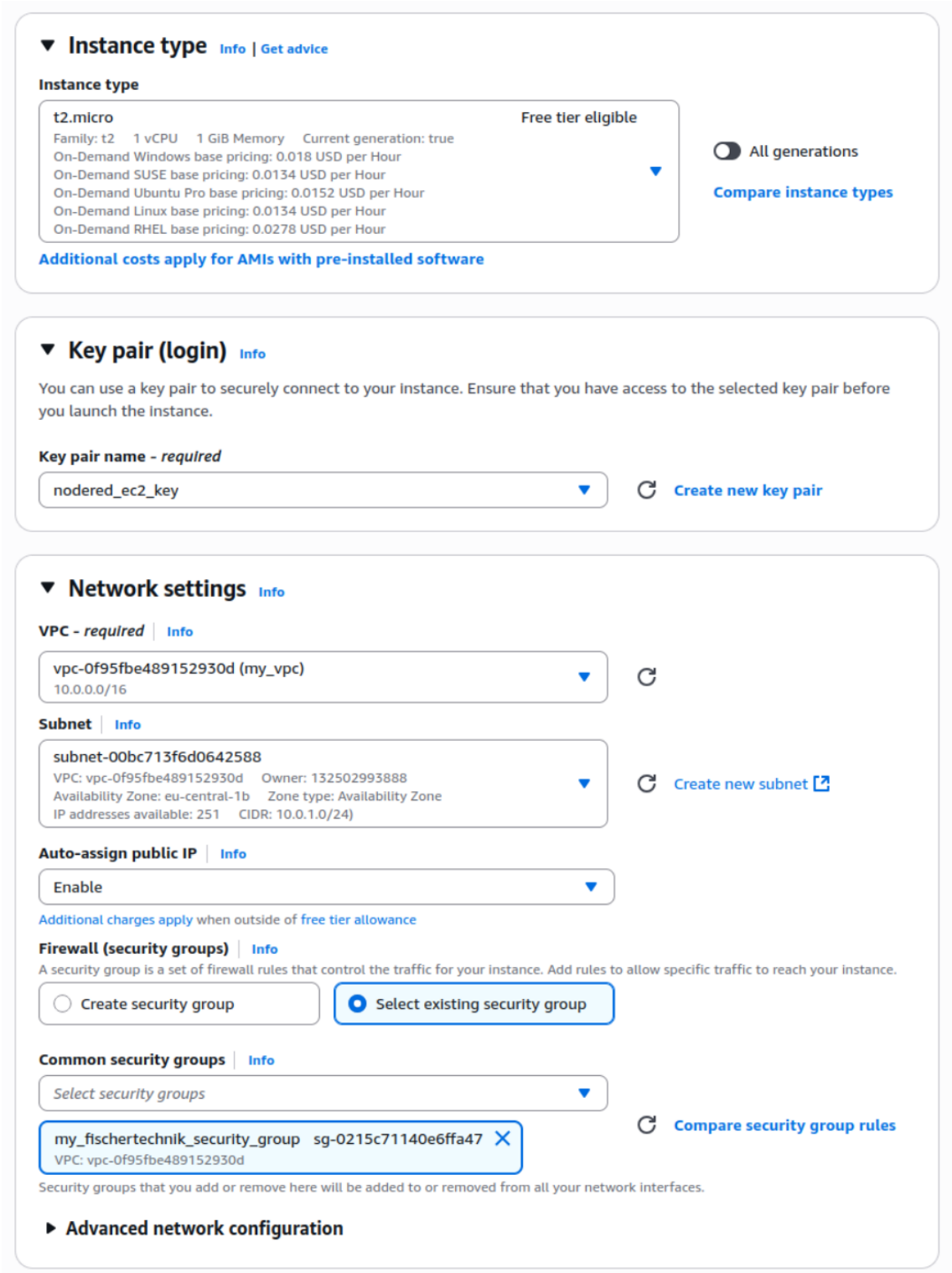

Figure 1.8: This screenshot shows how to set the network settings of the EC2 instance.

Due to a problem when creating EC2 instances, additional tags must be stored in the EC2 instance. Otherwise, no new EC2 instance can be launched. The tags are listed in the following screenshot.

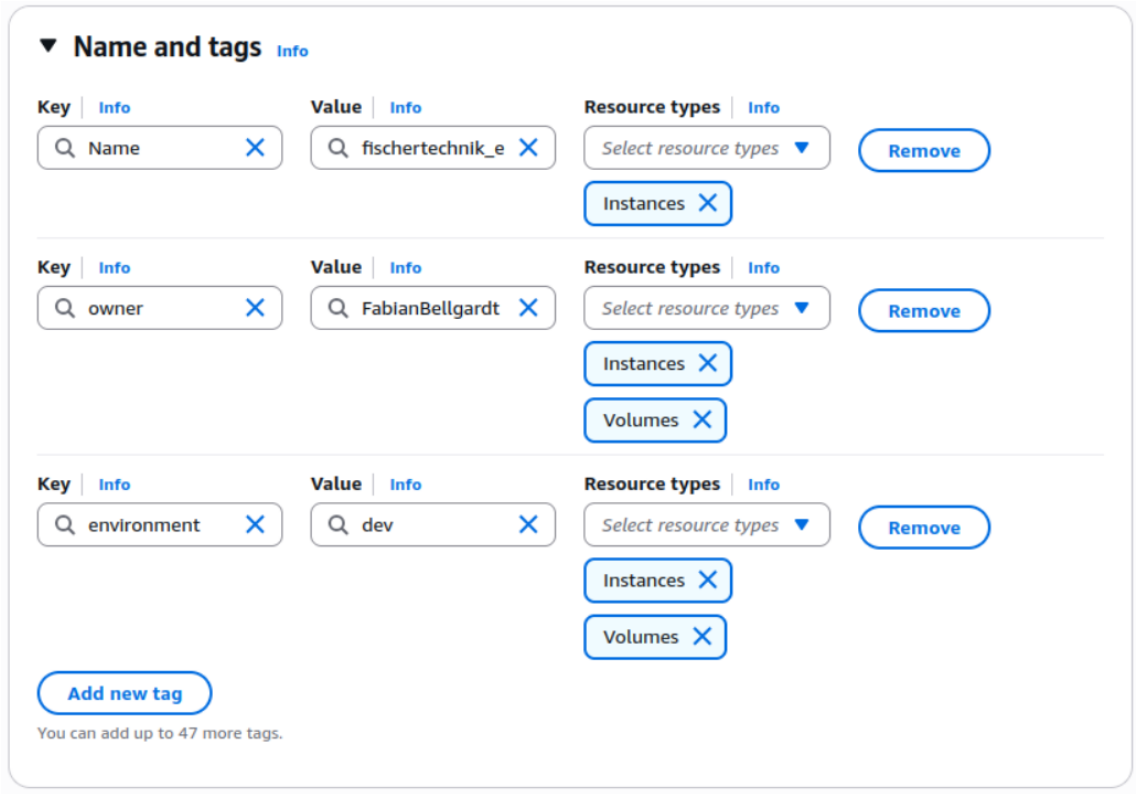

Figure 1.9: This screenshot shows how to add additional names and tags to the EC2 instance. They are needed to launch a new instance.

There is a field to create a key. This key is needed to connect to the EC2 Instance using SSH. The key can be given a specific name.

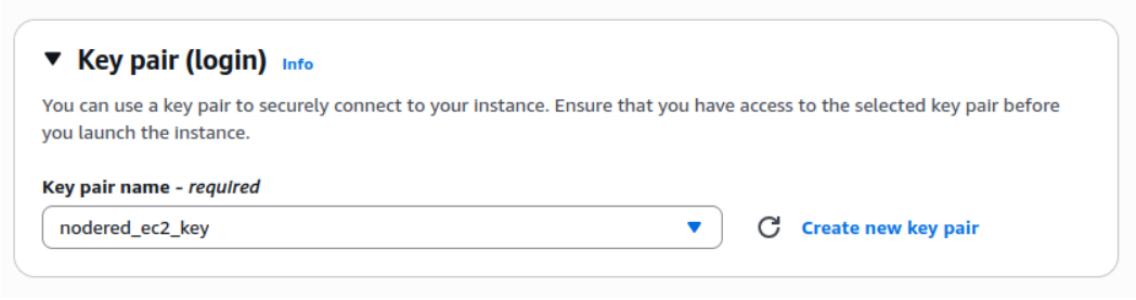

Figure 1.10: This screenshot shows how to download the key for SSH access.

The previous created routing table needs to be attached to a subnet. After this step, the

EC2 instance can be launched.

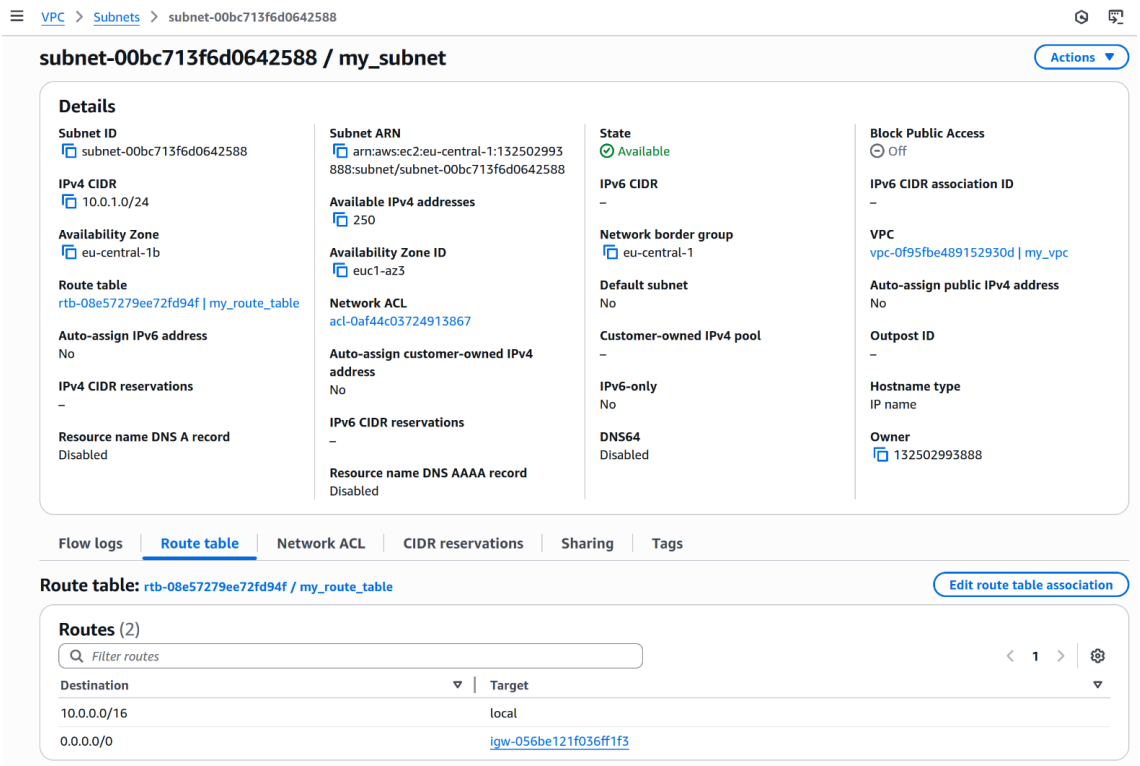

Figure 1.11: This screenshot shows how to connect the routing tables with a subnet.

1.2.1 Connect with Linux on EC2 Instance using SSH

Establishing SSH Connection to an EC2 Instance

To configure and manage the Linux-based EC2 instance hosting the Node-RED application, a secure connection is established using the SSH protocol. AWS provides a key pair (consisting of a public and private key) during instance creation. The private key (.pem file) is stored locally and used for authentication.

To start the connection, we have to move in the directory where the previous downloaded key is located. The connection is initiated via terminal using the following command:

It is important to note that the IP address undergoes a change following a restart of the instance. The following IP address is provided solely for illustrative purposes.

```
cd Downloads
ssh -i "nodered_ec2_key.pem" ec2-user@52.58.103.183
```

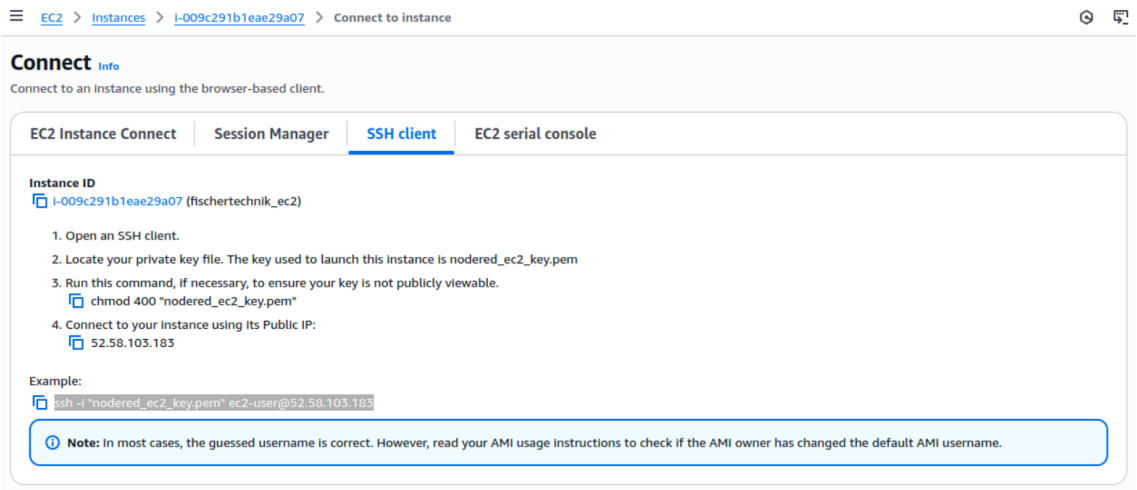

Figure 1.12: This screenshot shows how to establish a connection to the EC2 instance using SSH.

After you followed the previous step successful, the following text should appear.

Figure 1.13: This screenshot shows the Linux terminal feedback after a successful connection using SSH.

1.2.2 Install Docker and Node-RED using the SSH connection

In this example, the laptop was connected to the Wireless Fidelity (Wi-Fi): SDSU_Guest. To install Docker and Node-RED, use the following commands to install both using a established SSH connection to the Linux environment on the EC2 instance.
install Docker

```
whoami (answer: ec2-user)
sudo dnf update
sudo dnf install docker
sudo systemctl start docker
sudo systemctlenable docker
sudo systemctl status docker
```

```
docker ps (answer: access denied)
sudo usermod -aG docker $USER
newgrp docker
docker ps (List with running docker container)
```

install Node-RED

```
docker pull nodered/node-red
docker image ls
mkdir nodereddata
```

The following command creates the now Node-RED instance within a Docker container. The container is accessible at port 80. The directory is also specified in this command.

```
docker run -it -p 80:1880 -v /home/ec2-user/nodereddata:/data
--name mynodered nodered/node-red
```

start Node-RED

```
docker start mynodered
```

1.2.3 Connect to Node-RED in browser

After the EC2 instance got restarted, the Docker container with Node-RED on EC2 has to be also manually restarted using SSH!

The laptop has to be connected to the Wi-Fi: SDSU_Guest.

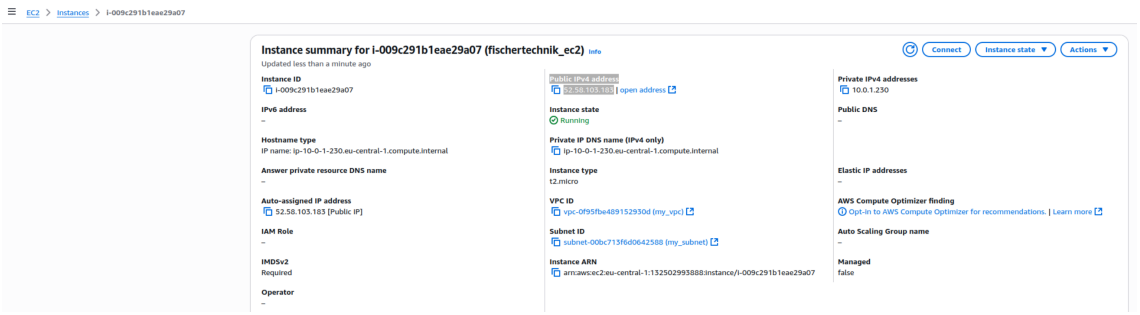

Figure 1.14: This screenshot shows where to find the URL to access the Node-RED environment. the Port :80 has to be added!

This example uses the address URL: http://52.58.103.183/ to access the Node-RED environment that runs on the Docker container. It is important to look up the current IP address.

Sometimes a reconnection to the Wi-Fi is needed. The Wi-Fi used in this example is called: SDSU_Guest. The IP Addresses used in the EC2 Instance bases on the IP Addresses used for the connection to the Wi-Fi used. Also it is important to use a http connection (Port 80!)

1.2.4 Add Password Security to Prohibit Access.

After the Node-RED environment got created, everyone can access is, using the public IP address. To prohibit this access, an authentication is added. The following steps show how the password security is established. The password was created using a salt 8 round by-crypt generator.

Name	admin
Password (clear)	#Nodered_fabian_25
Hash (bcrypt)	$2a$08$DWUuqOzAF3yG/FcynJmSYun PpFVDDbYAS5ANbxUdfx2A9vHO6ldG2

Table 1.1: Clear-text password and its corresponding bcrypt hash

Find and change the following file.

```
cd /home/ec2-user/nodereddata
sudo nano settings.js
```

```
/** To password protect the Node-RED editor and admin API, the following
 * property can be used. See https://nodered.org/docs/security.html for details.
 */
adminAuth: {
    type: "credentials",
    users: [{
        username: "admin",
        password: "$2a$08$DWUuqOzAF3yG/FcynJmSYunPpFVDDbYAS5ANbxUdfx2A9vHO6ldG2",
        permissions: "*"
    }]
},
```

Figure 1.15: This screenshot shows how to change the settings.js file to add password security

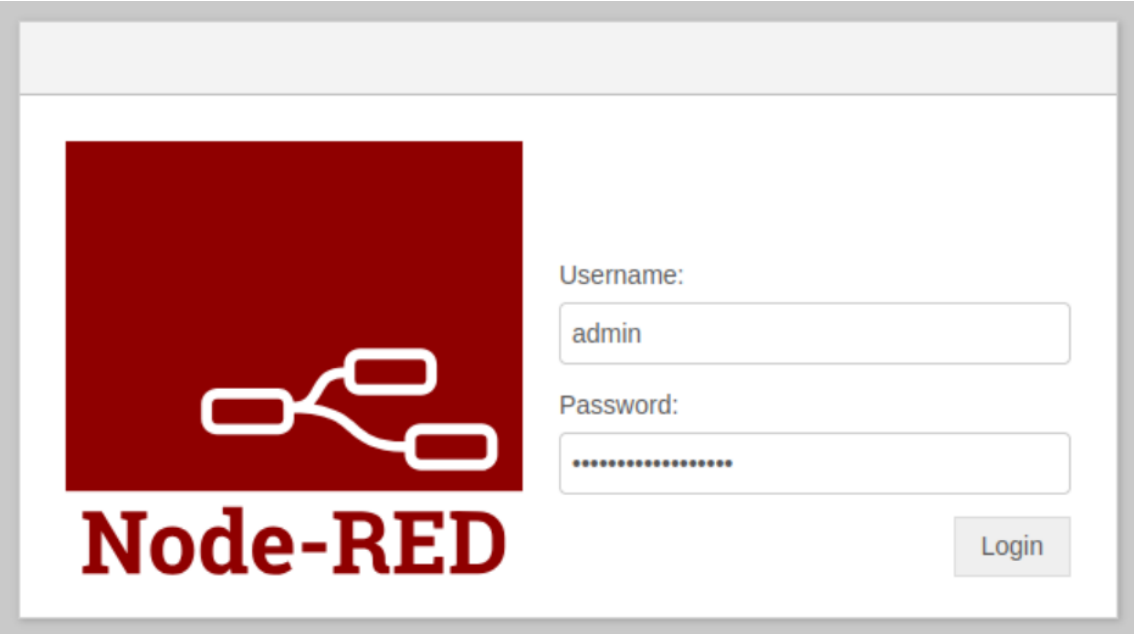

Figure 1.16: This screenshot shows what the new authentication looks like.

- name: admin
- password: #Nodered_fabian_25

1.3 Node-RED configuration

During the author's bachelor's thesis, a similar Node-RED dashboard and corresponding JavaScript Object Notation (JSON) configuration file were already developed. This earlier implementation enabled basic visualization and control of MQTT messages exchanged with the Fischertechnik factory model. This JSON file was reused and reconfigured to fulfil the needs of the new use case.

It is available in form of a GitHub-Repository. It can be downloaded using the following link [Bel25].

The JSON Files from this project needs an additional package. This package has to be installed manually. To install a new package click on the three strips on the right side, and click on the last point 'manage palette'.

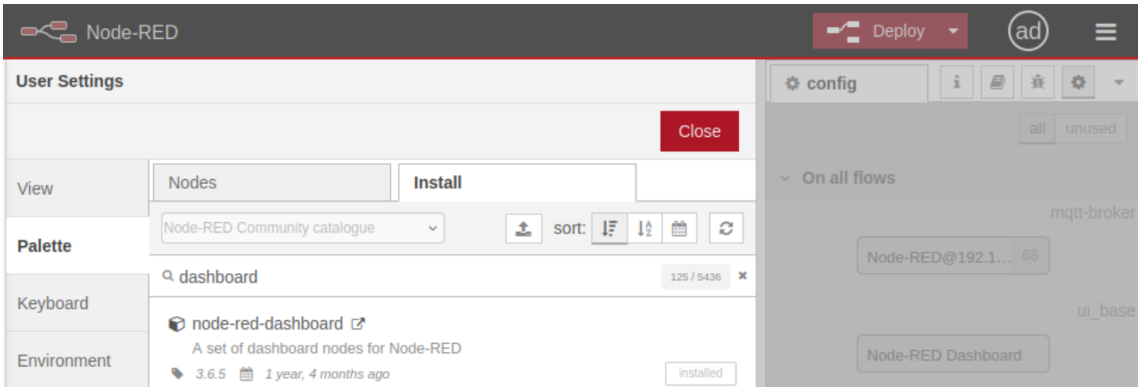

Figure 1.17: This screenshot shows where to install a new package.

1.4 Node-RED to Factory Communication via AWS IoT Core

To enable seamless communication between the cloud and the Fischertechnik Factory, a Node-RED instance deployed on an AWS EC2 instance is connected to a Node-RED instance running locally within the factory. The data exchange is facilitated using AWS IoT Core through a designated IoT Thing named `fischertechnik_thing_iso`.

This IoT Thing acts as a secure messaging broker between the two Node-RED environments. All MQTT messages routed through `fischertechnik_thing_iso` strictly use International Organization for Standardization (ISO) 8601 timestamps (e.g., `"2025-06-20T12:00:00Z"`), ensuring consistent and standardized temporal data representation across the distributed system.

Within the factory, the local Node-RED instance publishes sensor data and machine states to MQTT topics associated with `fischertechnik_thing_iso`. The cloud-based Node-RED instance subscribes to these topics via AWS IoT Core, enabling real-time processing, visualization, or further integration with other cloud services. Conversely, control commands or configurations can be sent from the cloud to the factory using the same communication channel, enabling bidirectional interaction in a vendor-independent and scalable architecture.

1.4.1 Creating a New IoT Thing in AWS

A new IoT Thing is needed so separate the traffic between the Node-RED on the EC2 instance and the Node-RED on the factory's Raspberry Pi. This is needed, since for this connection the messages have to use the ISO 8601 formate. The traffic between the factory and AWS IoT SiteWise uses the Epoch formate as a timestamp. During the creation, AWS offers you to download certificates. They are needed to establish a connection from the Node-RED Instances, using MQTT -in and MQTT -out nodes.

- Certificate: ...certificate.pem.crt

- Private key: ...private.pem.key
- CA Certificate: ...AmazonRootCA1.pem

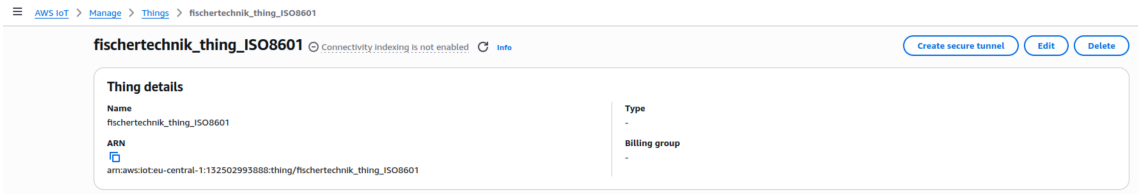

Figure 1.18: This screenshot shows where a new IoT Thing can be created.

1.4.2 Communication Between Node-RED Instances and the AWS IoT Thing fischertechnik_thing_iso

Both Node-RED instances communicate with each other via the AWS broker named fischertechnik:thing_iso. Each instance can publish messages to or subscribe to messages from this broker. The following two tables illustrate the configuration depending on the direction of message flow, whether sending or receiving. There are two different configurations:

- The local_mqtt_broker configuration is required to initially retrieve messages from the factory. This configuration is only used on the Node-RED environment running directly within the factory.
- The other configuration, named AWS: fischertechnik_thing_iso, is required on both Node-RED instances in order to send and receive data through the AWS broker fischertechnik_thing_iso. This setup ultimately enables communication between the two Node-RED instances.

1.4.3 Connection to the AWS Broker: AWS: fischertechnik_thing_iso

Table 1.2: Configuration of MQTT Nodes to connect to fischertechnik_thing_iso

Parameter	**Value**
Name	AWS: fischertechnik_thing_iso
Port	8883
Protocol	MQTT V3.1.1
Server	an70ci8l1maqu-ats.iot.eu-central-1.amazonaws.com
TLS name	tls_fischertechnik_thing_iso
Certificate	46e4b01528b89cfe2558ab827f15b2f2eaac1f35f de6b2b43ac1438261d67b0f-certificate.pem.crt
Private Key	46e4b01528b89cfe2558ab827f15b2f2eaac1f35 fde6b2b43ac1438261d67b0f-private.pem.key
CA Certificate	AmazonRootCA1(2).pem
Connect automatically	checked
Use TLS	checked
Use clean session	checked
Verify server certificate	checked
action	subscribe to single topic
topic	example: fischertechnik_thing_iso/f/i/stock
Output	auto detect (parsed JSON object, string or buffer)

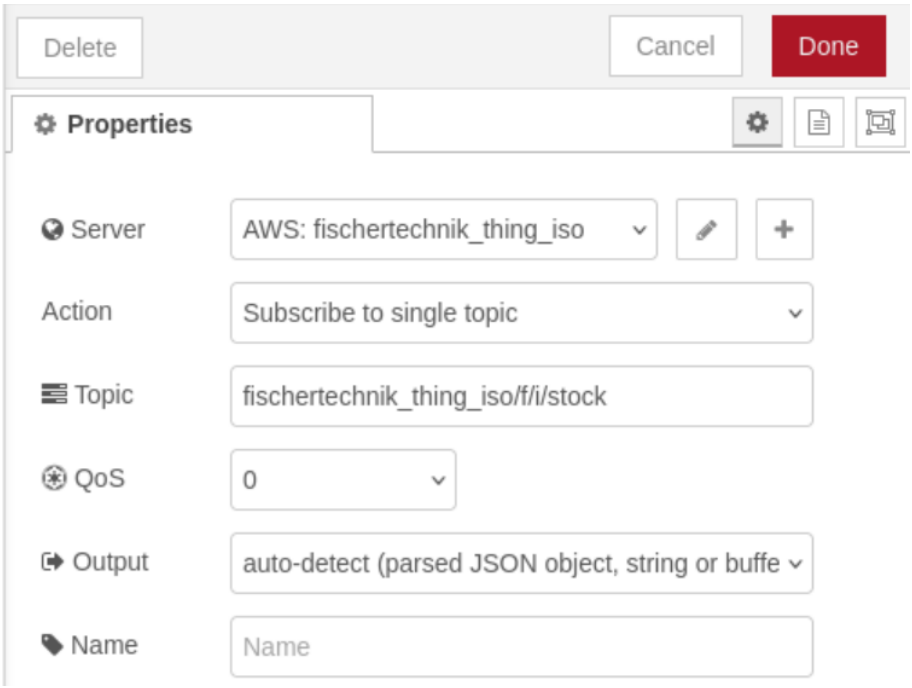

Figure 1.19: This screenshot shows the properties of an MQTT Node in Node-RED.

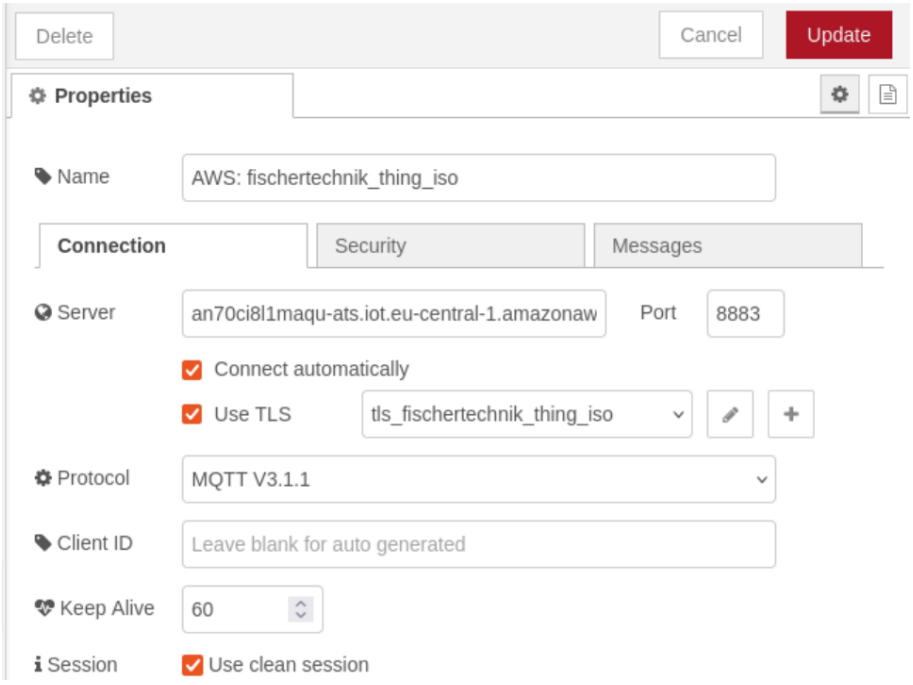

Figure 1.20: This screenshot shows the connection settings of the properties of a MQTT Node in Node-RED.

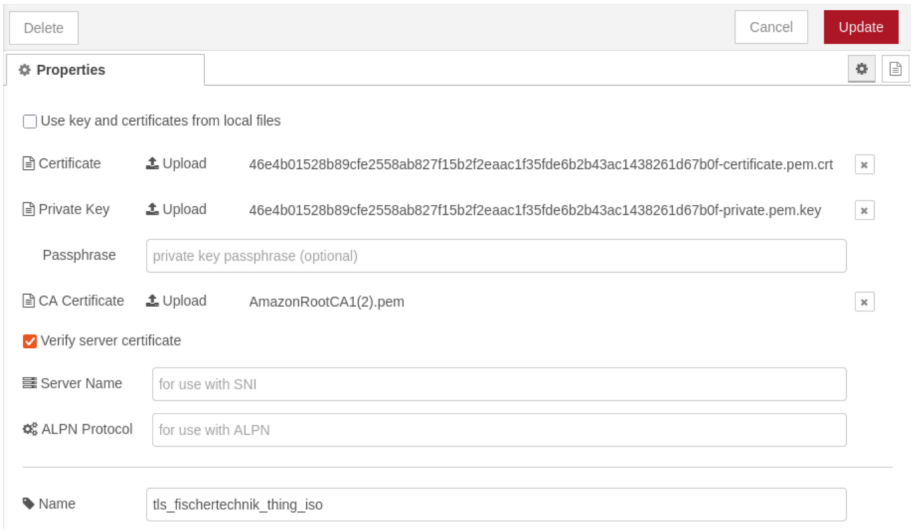

Figure 1.21: This screenshot shows the needed certificates to connect a MQTT node to an AWS Broker.

1.4.4 Connection to the Factory Broker: local_mqtt_broker

Table 1.3: Configuration of MQTT Nodes to connect to local_mqtt_broker

Parameter	**Value**
Name	local_mqtt_broker
Port	1883
Protocol	MQTT V3.1.1
Server	local_MQTT_Broker
Connect automatically	checked
Use TLS	unchecked
Use clean session	checked
action	subscribe to single topic
topic	example: f/i/stock
Output	a parsed JSON object

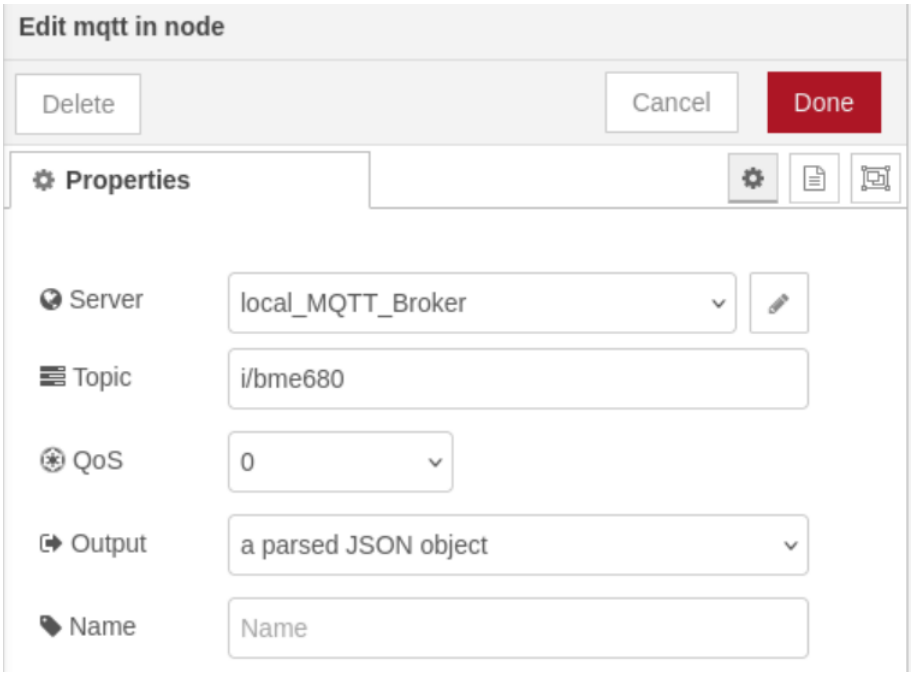

Figure 1.22: This screenshot shows how the MQTT node is configured that connects the factory to the IoT Core.

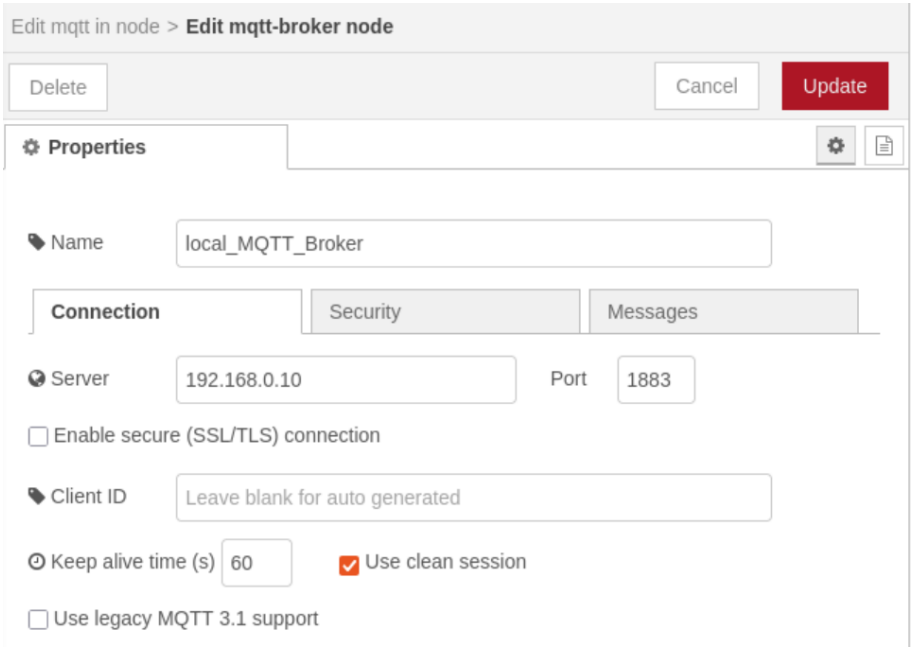

Figure 1.23: This screenshot shows how the MQTT broker node is configured with the needed IP address.

1.4.5 Node-RED on Factory: Node Setup

To access the Node-RED Environment on the factory's Raspberry Pi, connect to the factory's Wi-Fi:

- SSID: TP-Link-DACE-5G
- Password: 62021463

Node-RED can then be accessed using the url: http://192.168.0.5:1880. The following screenshot shows an Example of how the nodes are arranged. On the left side, the traffic of the factory gets subscribed, and this traffic gets published using the nodes on the right side. To test the traffic, it is recommended to use the AWS MQTT Test Client.

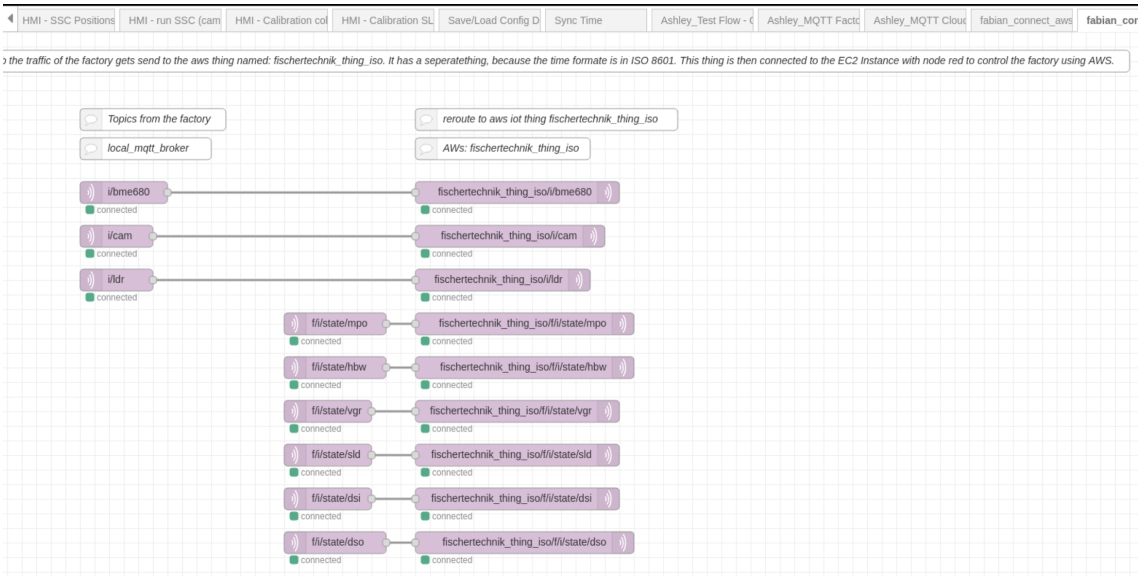

Figure 1.24: This screenshot shows how the nodes are connected to publish traffic from the factory to the IoT Core.

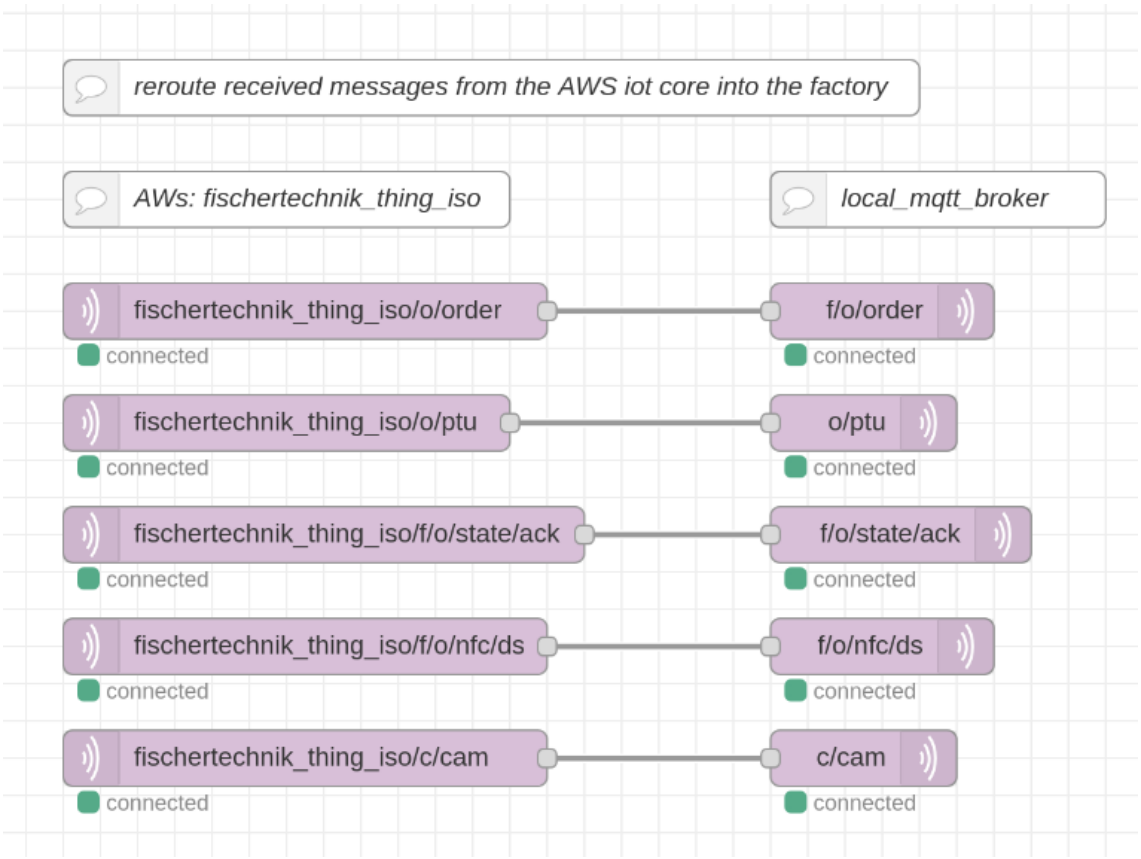

Figure 1.25: This screenshot shows how the nodes are connected to subscribe to traffic from the IoT Core and ingest them into the factory.

1.4.6 Node-RED on EC2: Node setup

The following two pictures show how the nodes in Node-RED on the EC2 instance in Node-RED are connected. The first examples shows how messages can be received from the AWS broker: fischertechnik_iot_thing. The second picture shows how the nodes can be connected to send orders back into the AWS broker: fischertechnik_iot_thing.

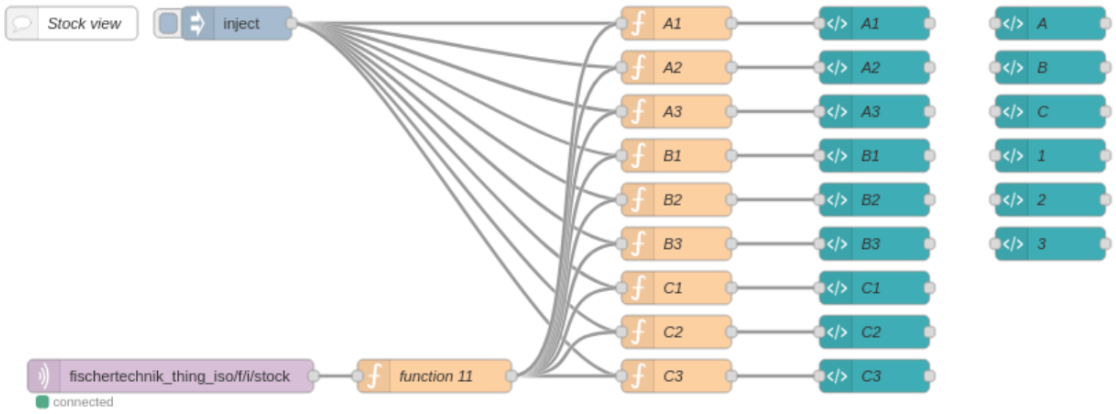

Figure 1.26: This screenshot shows how to subscribe to the AWS broker to receive messages from it.

Figure 1.27: This screenshot shows how to publish to the AWS broker to send messages to it.

Both connections can be tested using the MQTT test client on AWS. It is possible to read the messages that get published but also send messages from the test client to test the subscription.

1.4.7 The Dashboard

The final dashboard can be viewed and downloaded from GitHub at the following link [Bel25].

2 Step-by-Step Guide to Creating a Digital Twin in AWS

This chapter explains how the MQTT messages generated by the factory can be visualized within a 3D environment using the IoT TwinMaker service in the form of a digital twin. It outlines the required steps to create and connect instances across the relevant AWS services to achieve this functionality. Furthermore, a detailed step-by-step guide is provided to illustrate the complete setup process from message ingestion to 3D visualization.

2.1 AWS Iot Core Configuration

As a first step, an MQTT endpoint must be configured to enable the ingestion of messages originating from the factory environment into the cloud-based system. In AWS IoT Core, a Thing represents the digital identity of a physical device, such as a sensor or actuator. It enables secure communication between the device and the AWS cloud and can be enriched with metadata and security credentials. Each Thing can also have a Device Shadow, which maintains its current and desired state, allowing reliable synchronization even during connectivity issues. For this project a thing called 'fischertechnik_thing' got created.

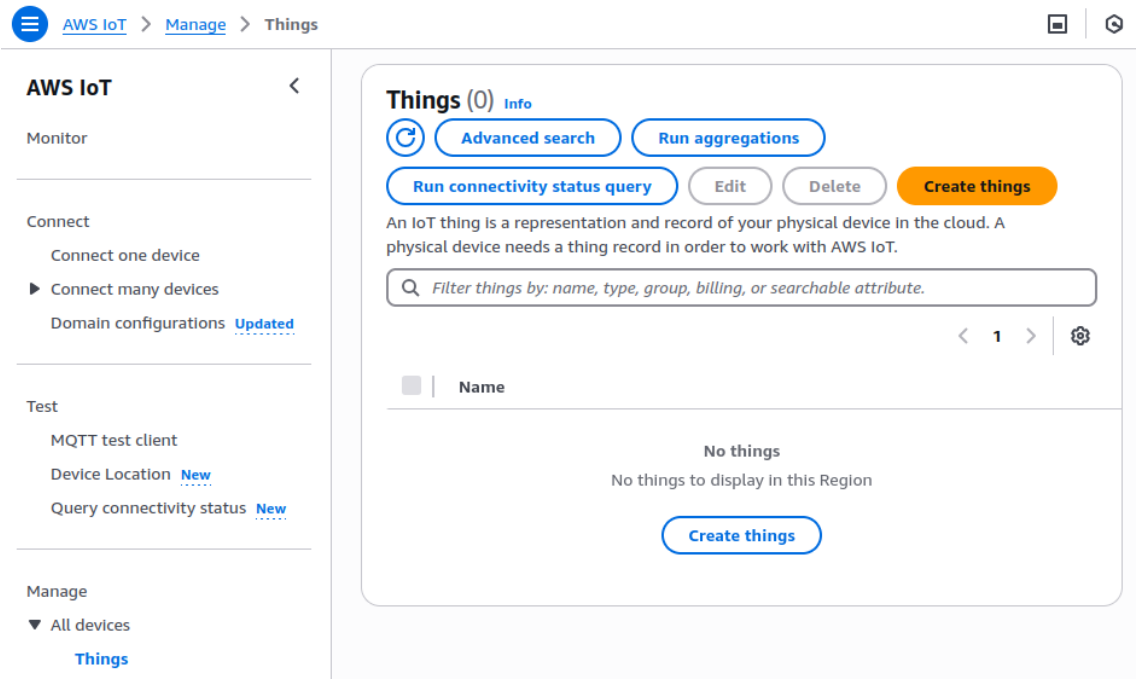

Figure 2.1: This picture shows where and how to create a thing in IoT Core.

For a secure communication between the Node Red Environment and AWS IoT Core, a secure connection has to established. For this, we will generate certificates in AWS.

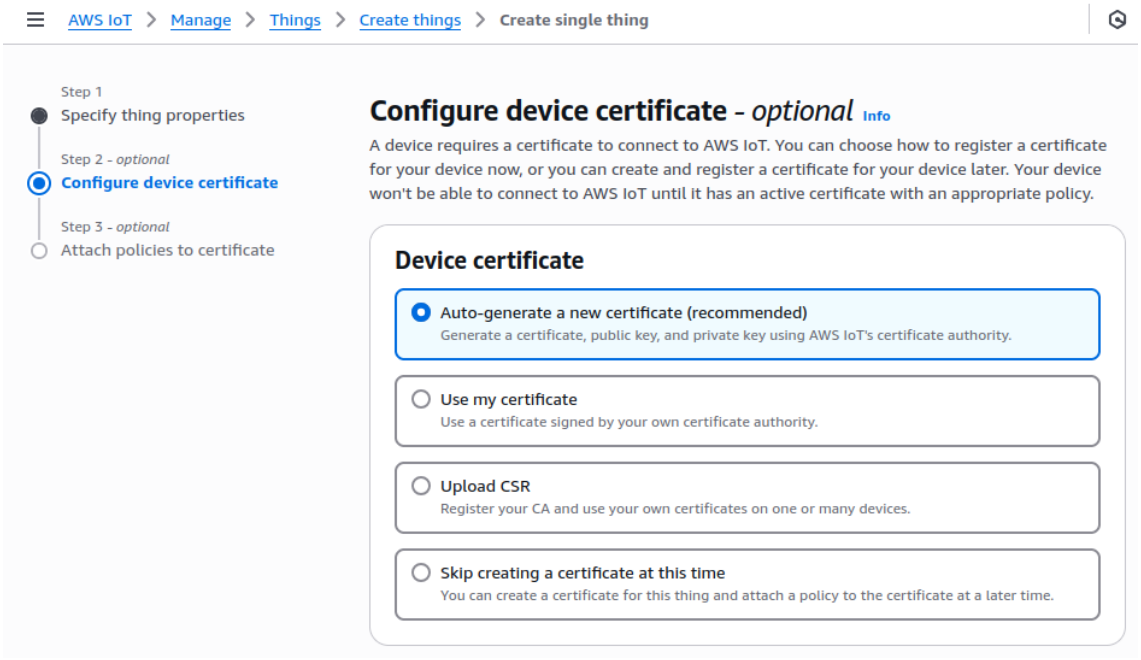

Figure 2.2: This picture shows how to create device certificates.

To restrict the rights, it is advisable to create a policy. To initially allow access to all resources, the placeholder '*' is used here.

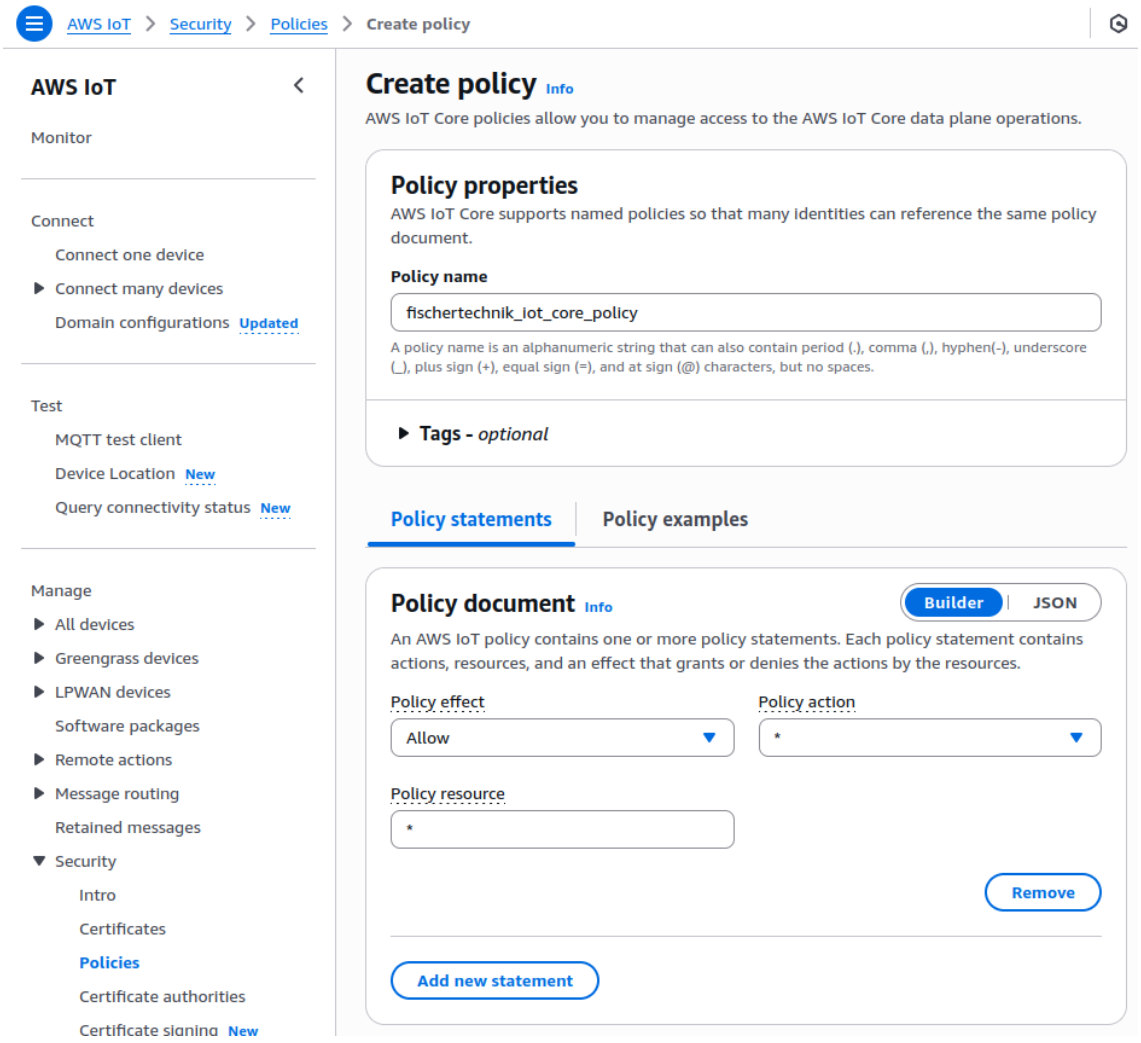

Figure 2.3: This picture shows how to attach policies.

After that, the certificates get created and can be downloaded.

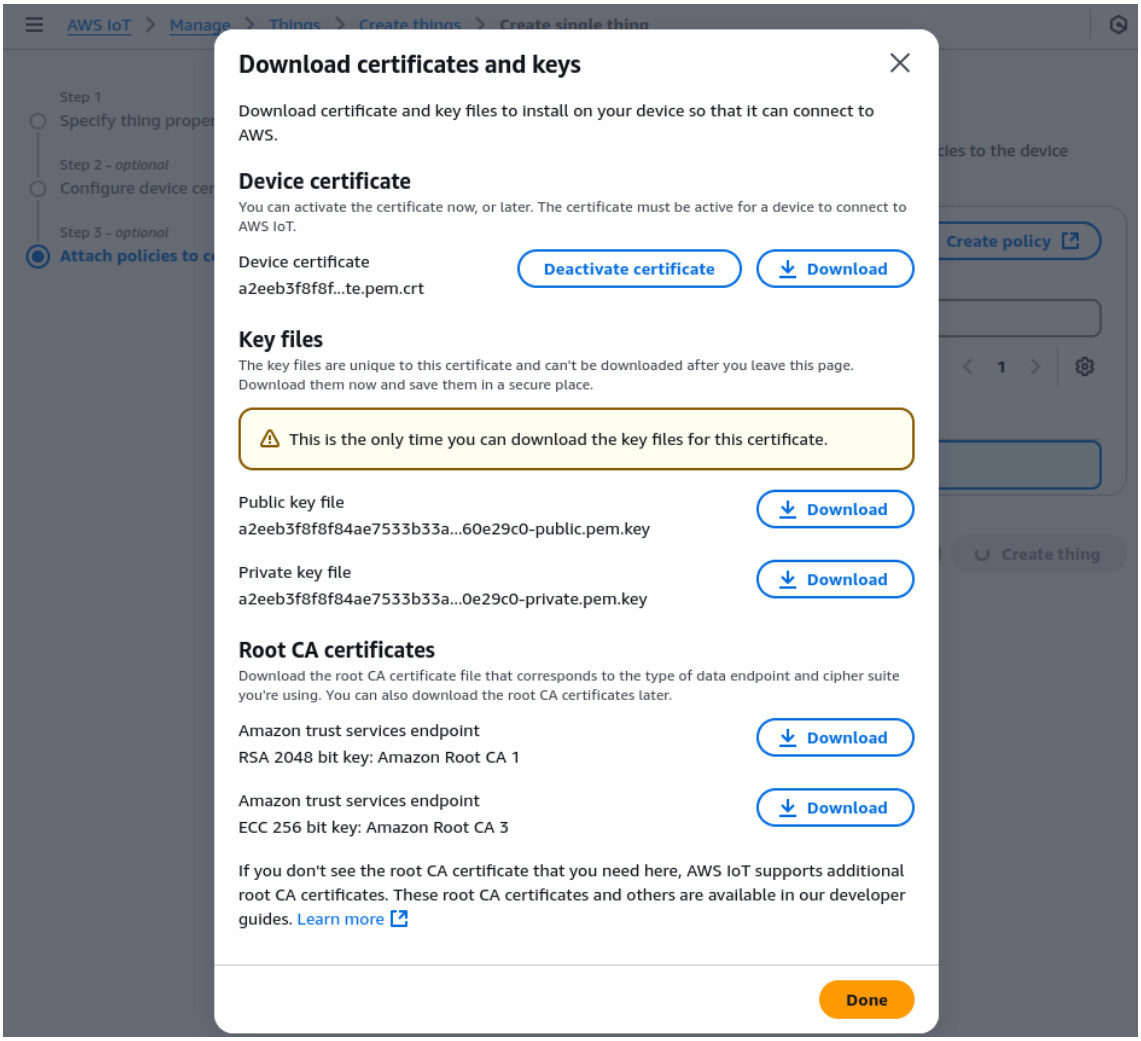

Figure 2.4: This picture shows how to download the credentials.

2.1.1 Setup MQTT Connection from Node-Red to IoT Core

To enable communication between AWS IoT Core and Node-RED, it is necessary to establish a connection using MQTT nodes within the Node-RED environment. These MQTT nodes act as interfaces for publishing data to and subscribing to messages from the configured MQTT broker, in this case, AWS IoT Core. In order to authenticate and authorize this connection securely, the device-specific security credentials that were previously downloaded during the Thing registration process in AWS must be uploaded into the configuration of the MQTT nodes. These credentials ensure that the communication is secure and enables the Node-RED instance to act as a trusted IoT device within the AWS IoT ecosystem. Once configured, the MQTT input and output nodes can be used to exchange messages with AWS IoT Core, enabling integration of Node-RED with other AWS services for further processing, storage, or visualization.

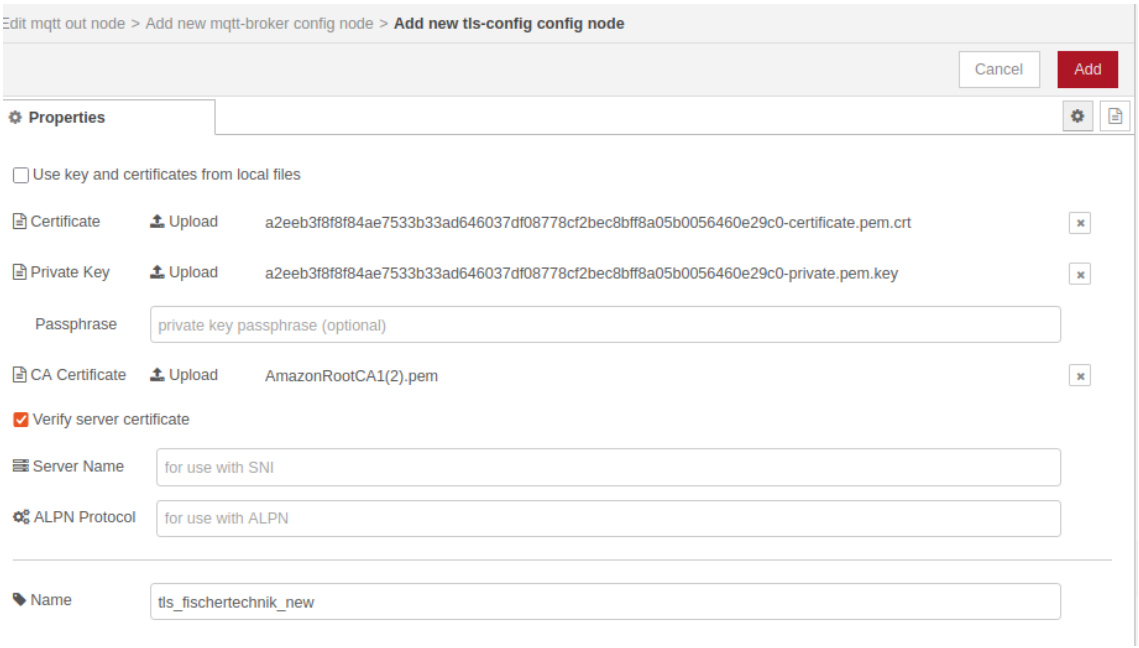

Figure 2.5: This picture shows how to upload the credentials.

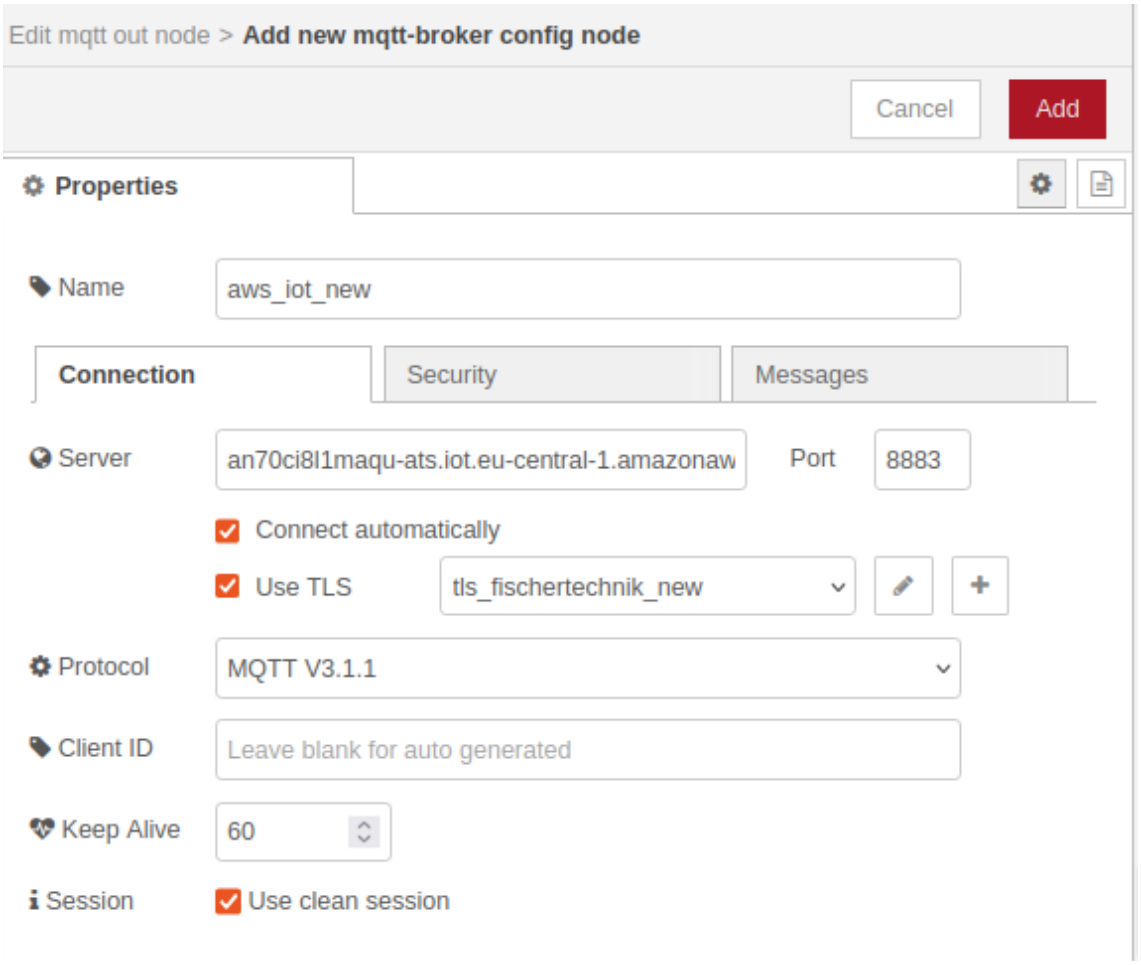

Figure 2.6: This picture shows how to configure the MQTT Node

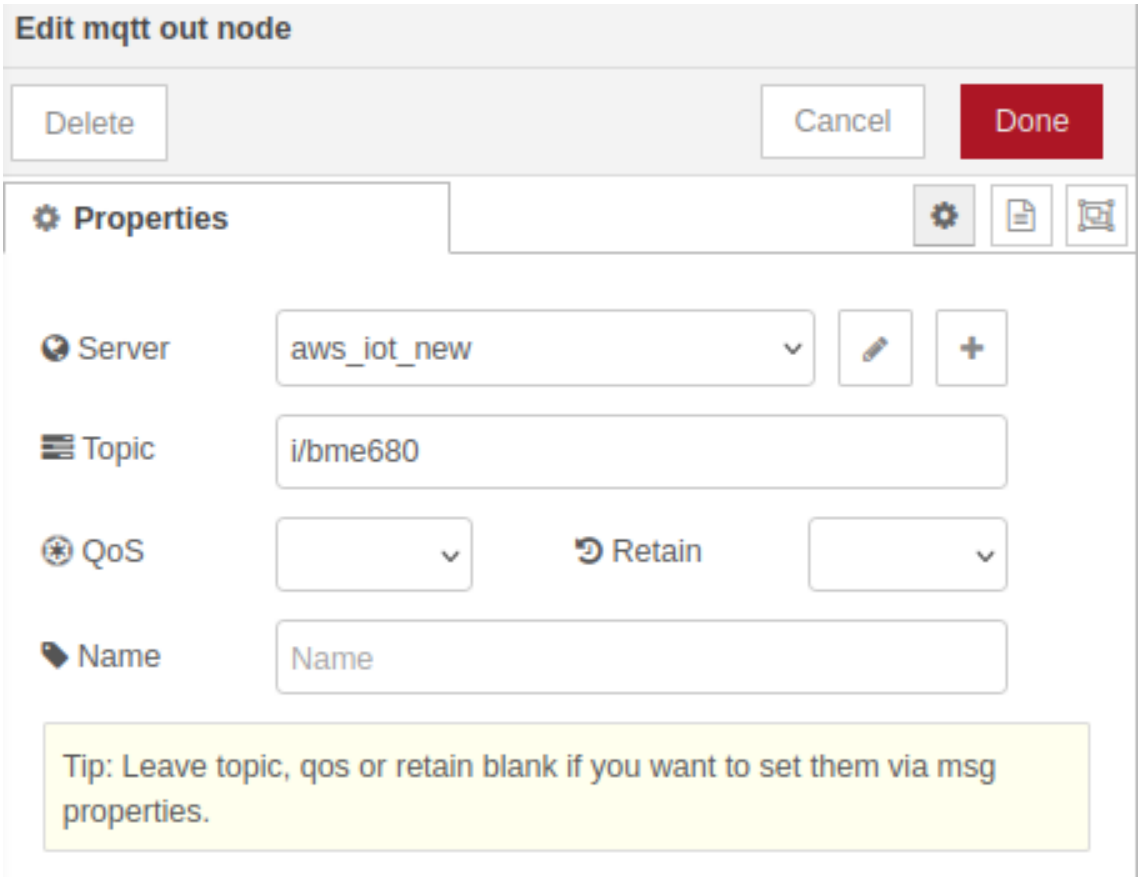

Figure 2.7: This picture shows how to set a specific Topic.

In addition to uploading the required security credentials, a new MQTT broker configuration should be created within Node-RED and assigned a descriptive name, such as 'tls_fischertechnik_new'. Since the communication with AWS IoT Core requires a secure connection, the default MQTT port 1883 must be changed to the Transport Layer Security (TLS) secured port 8883. Furthermore, the option "Use TLS" must be explicitly enabled within the MQTT node configuration to ensure encrypted communication. The server address for the connection corresponds to the unique AWS IoT Core endpoint associated with the previously created Thing. This endpoint can be retrieved by navigating to AWS IoT → Connect → Device → View Endpoint in the AWS Management Console. Once all parameters are correctly set, namely the server endpoint, port, TLS settings, and uploaded credentials—the MQTT node in Node-RED will attempt to establish a connection. If successful, the node will be visually marked with a green status dot, and the label "connected" will appear next to it, indicating a successful and active MQTT session.

2.1.2 Setup Test Input for MQTT Node

To validate the setup before integrating the actual Fischertechnik Factory, a simulated MQTT message from a Bosch Meteorological Environmental (BME) 680 environmental

sensor is generated using *Node-RED*. This approach allows testing of the full communication chain from message creation to transmission into AWS IoT Core via a secure TLS connection.

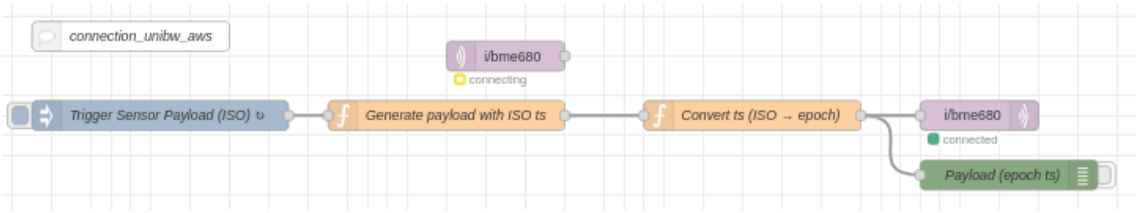

Figure 2.8: This picture shows how to ingest test messages into AWS using Node-RED.

- The flow starts with an inject node labeled Trigger Sensor Payload (ISO), which triggers every 10 seconds and initiates the simulation.
- This message is passed into a function node named Generate payload with ISO ts, where a JSON object representing sensor data is generated. The payload includes:
  - `ts`: Timestamp in ISO 8601 format
  - `t, rt, h, rh, p, iaq, aq, gr`: Simulated temperature, humidity, air pressure, and air quality parameters
- The second function node Convert ts (ISO → epoch) converts the ISO timestamp to an epoch value in seconds, as required by many AWS time series applications. This conversion is necessary. IoT SiteWise requires the epoch format!
- The message is then forwarded to an MQTT out node that publishes the payload to the topic /bme680 via a secure TLS connection to AWS IoT Core.

The MQTT broker configuration named aws_iot_new is set to connect to the AWS IoT Core endpoint (e.g., an70ci8l1maqu-ats.iot.eu-central-1.amazonaws.com) over port 8883, which is required for secure MQTT communication. The option Use TLS is enabled, and the TLS configuration tls_fischertechnik_new includes the previously downloaded certificates:

- Certificate file: *.certificate.pem.crt
- Private key: *.private.pem.key
- Root CA: AmazonRootCA1.pem

The endpoint used in this connection can be retrieved in the AWS Management Console under: AWS IoT Core → Settings → Device data endpoint. A successful connection is indicated in Node-RED by a green status dot and the label `connected` on the MQTT output node. In parallel, an MQTT in node is configured to listen to the same topic `i/bme680` from a local MQTT broker (for example, IP `192.168.0.10`, port `1883`) to support local testing and debugging. This setup demonstrates the interoperability between local and

cloud-based MQTT environments. Once the communication is verified using the simulated payload, the same flow can be connected directly to the real Fischertechnik Factory. Since the factory publishes sensor values under the same topic structure, the remaining steps (data transformation, forwarding, and integration with AWS services) remain valid without modification.

2.1.3 Test Connection

After the connection got implemented, it is recommended to test if it works. For that, AWS offers a MQTT test client in its AWS IoT Core. The following picture shows a subscription to the topic 'i/bme680'. This topic sends informations about the environment. It shows all MQTT messages that get received under this topic.

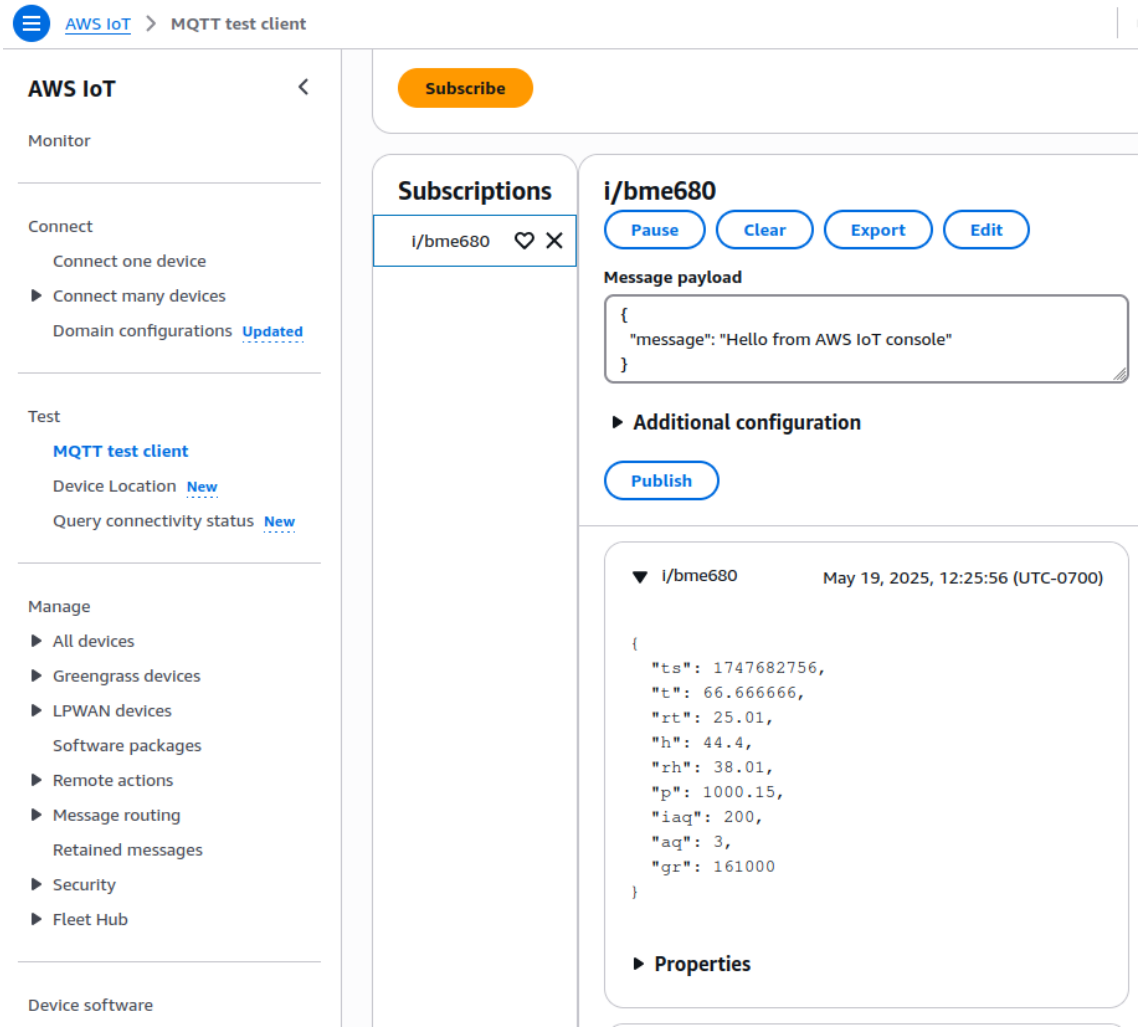

Figure 2.9: Test the setup with the MQTT Test Client in AWS.

2.2 IoT SiteWise

IoT SiteWise is a managed service designed to model, ingest, and organize data from industrial equipment in a structured and scalable way. It enables users to represent physical systems digitally through defined models and asset instances. One of the key advantages of IoT SiteWise is its ability to create hierarchical representations of a factory or system layout, which supports structured data management and analysis. Using IoT SiteWise as the data foundation for IoT TwinMaker is particularly beneficial. Since IoT TwinMaker builds digital twins based on dynamic, real-world data, IoT SiteWise provides a logical structure and consistent interface for accessing time-series values, calculated metrics, and live telemetry. This integration significantly reduces the complexity of digital twin development and ensures that 3D scenes in IoT TwinMaker reflect real-time operational conditions.

2.2.1 Models

An asset model in IoT SiteWise serves as a blueprint for a specific type of equipment or system. It defines the properties of the device, including static attributes. These models ensure consistency and reusability, as the same structure can be used for multiple identical or similar assets.

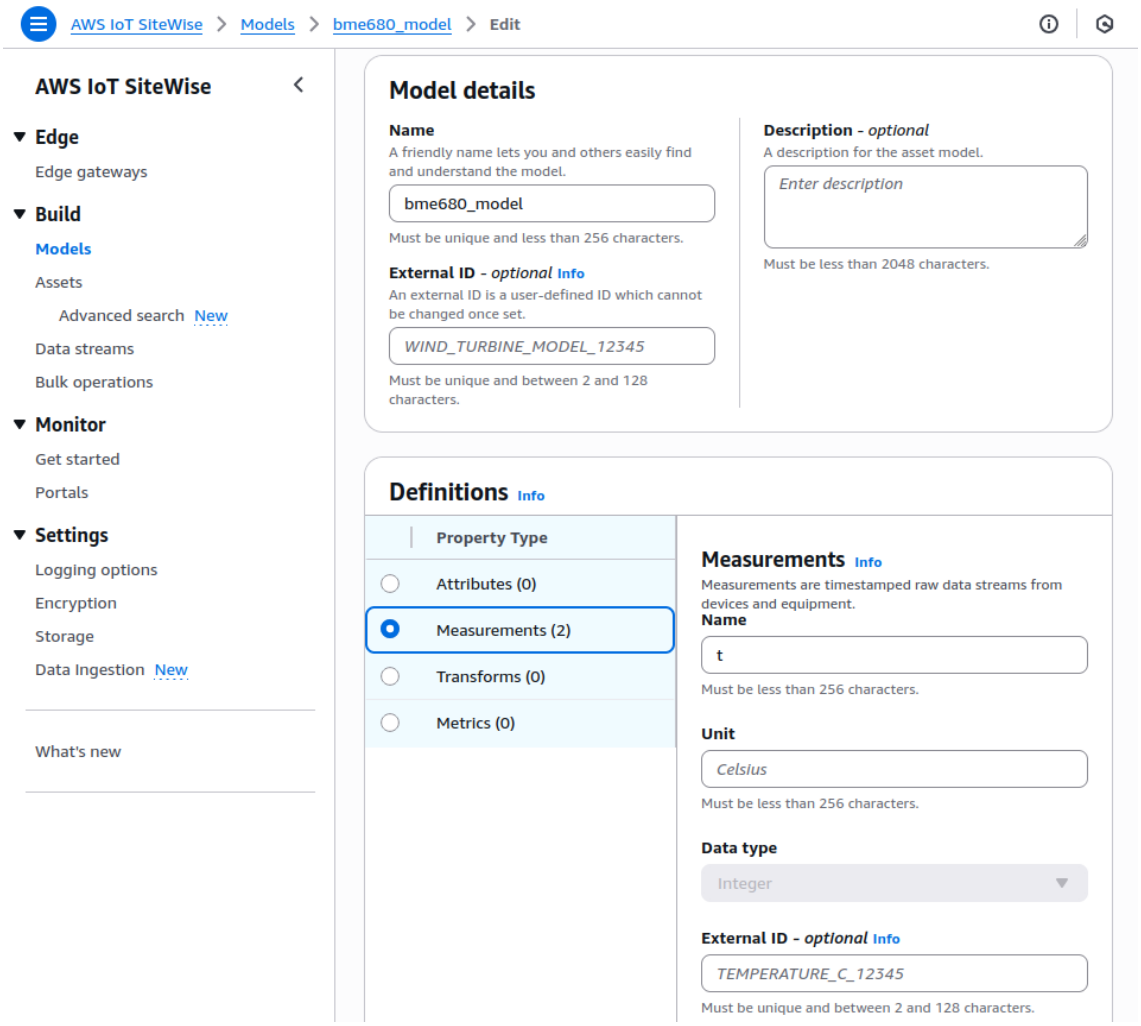

Figure 2.10: Create a Model for the BME 680 Message.

2.2.2 Assets

An asset is a concrete instance of an asset model and represents a specific physical device or system in the field. Each asset inherits the structure defined by its model and can be connected to real-time data sources, such as MQTT messages from AWS IoT Core. Assets are used to track the current state of devices and organize data within the IoT SiteWise asset hierarchy. This allows users to build scalable monitoring solutions and power digital twin applications where asset properties are visualized and updated in real time.

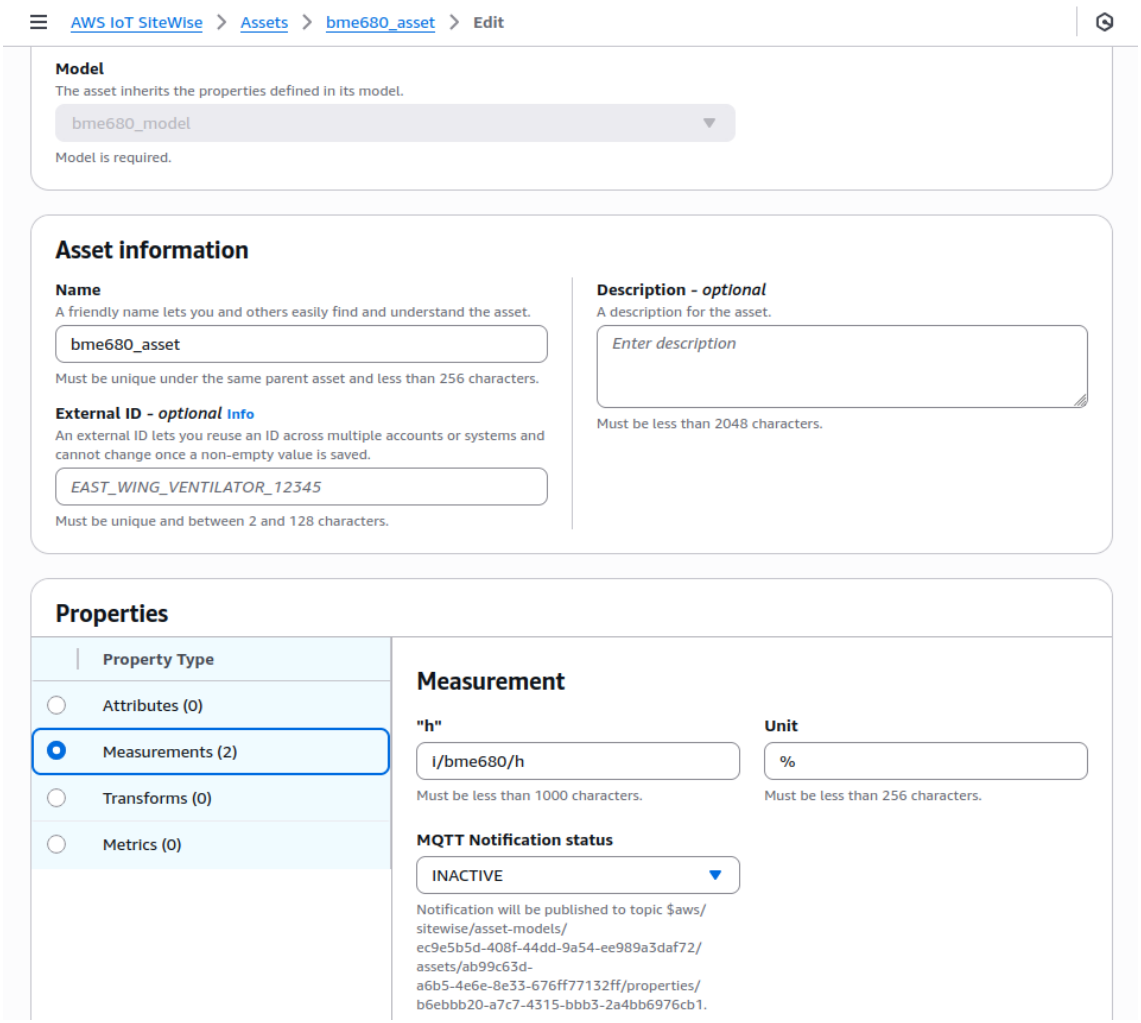

Figure 2.11: Create a Asset of a Model for the BME Message.

Asset alias has to have an unique name. For orientation reasons it is recommended to name it after the topic, but it is not necessary to name it similar to the topic.

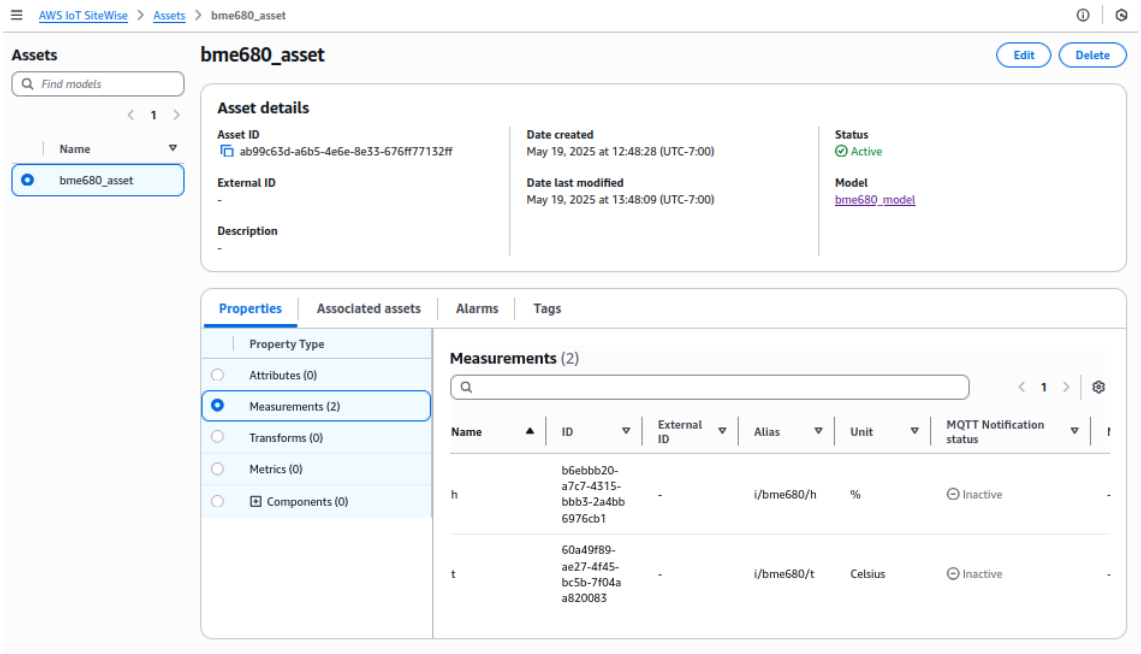

Figure 2.12: Set a specific alias for every asset.

2.3 IoT Rules Engine

The AWS IoT Rules Engine is a core component of AWS IoT Core that enables the processing and routing of incoming MQTT messages based on user-defined rules. These rules are written using a Structured Query Language (SQL)-like syntax that allows filtering, transforming, and selecting specific parts of the message payload or topic structure. The primary purpose of the Rules Engine is to enable seamless integration between IoT devices and other AWS services. For example, a rule can be configured to forward sensor data from an MQTT topic directly to services such as Amazon Timestream, AWS Lambda, Amazon Simple Storage Service (S3), or IoT SiteWise.

In this example, a new rule named 'core_to_sitewise_rule' ist created. This rule needs its own SQL statement to get the wished information out of the topic.

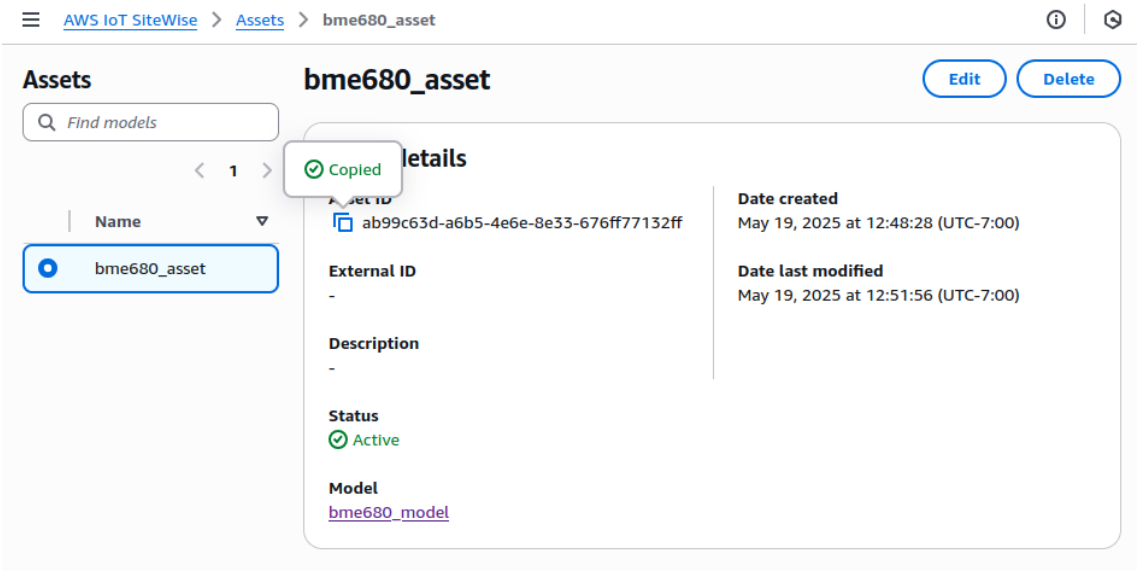

Figure 2.13: Screenshot about where to find the asset id.

The rule now wants to mach the received data out of the topic to a specific data field in an asset. For that, we have to implement the Asset ID of the Asset and the Property ID of a specific Value of the Asset to the right field in the IoT Rule. The Asset ID can be copied out of the Asset itself.

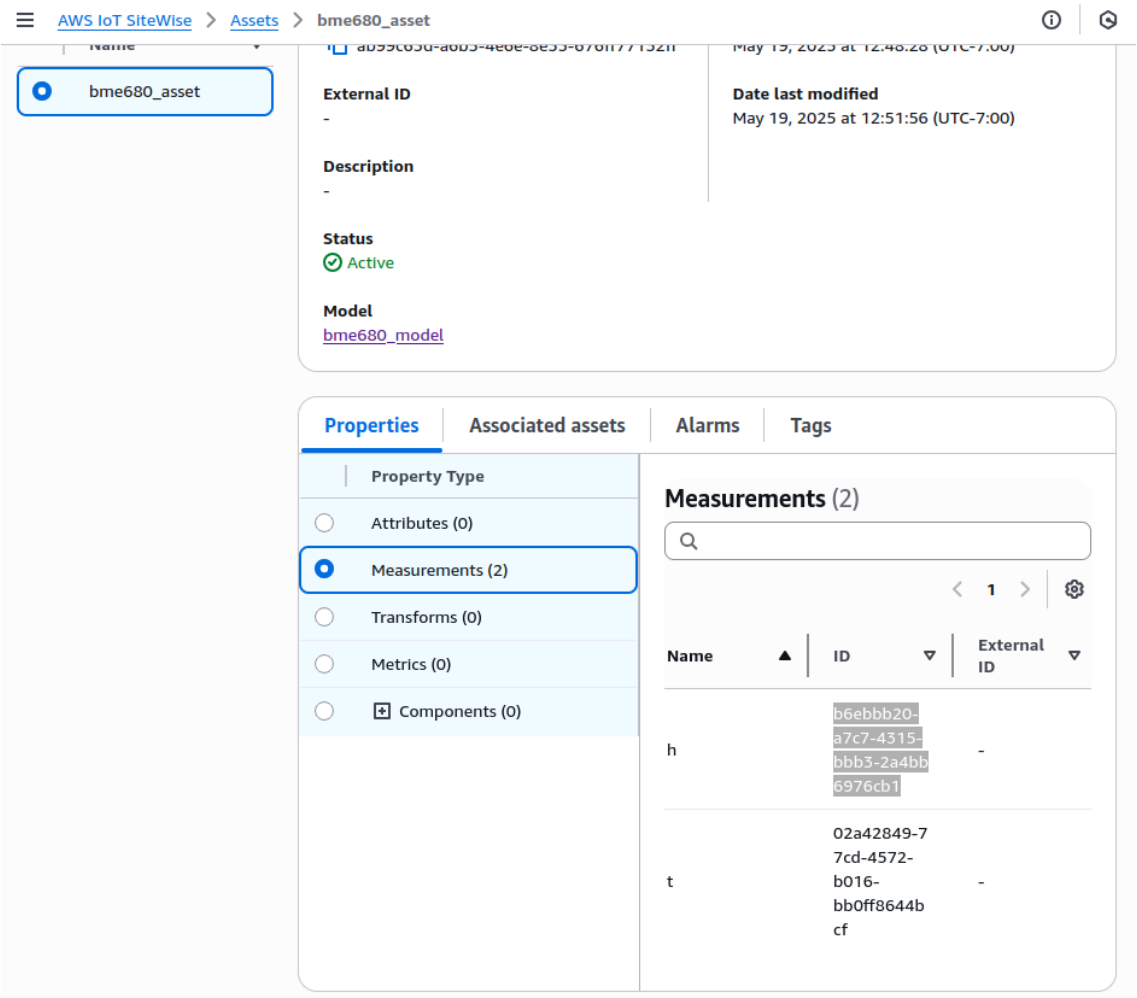

Figure 2.14: Screenshot about where to find the property id.

Within the rule, we now have to add rule actions. In those rule actions we choose 'By asset ID and property ID' and add both. Additionally we have to the rule the ability to read the timestamp of the message. For that, we use the value ${ts}. The data type should match the data type used in the creation of the measurement in the asset. Then we need also the name of the value ${h}. We have to create rule actions for every value we want to extract of the message into IoT SiteWise.

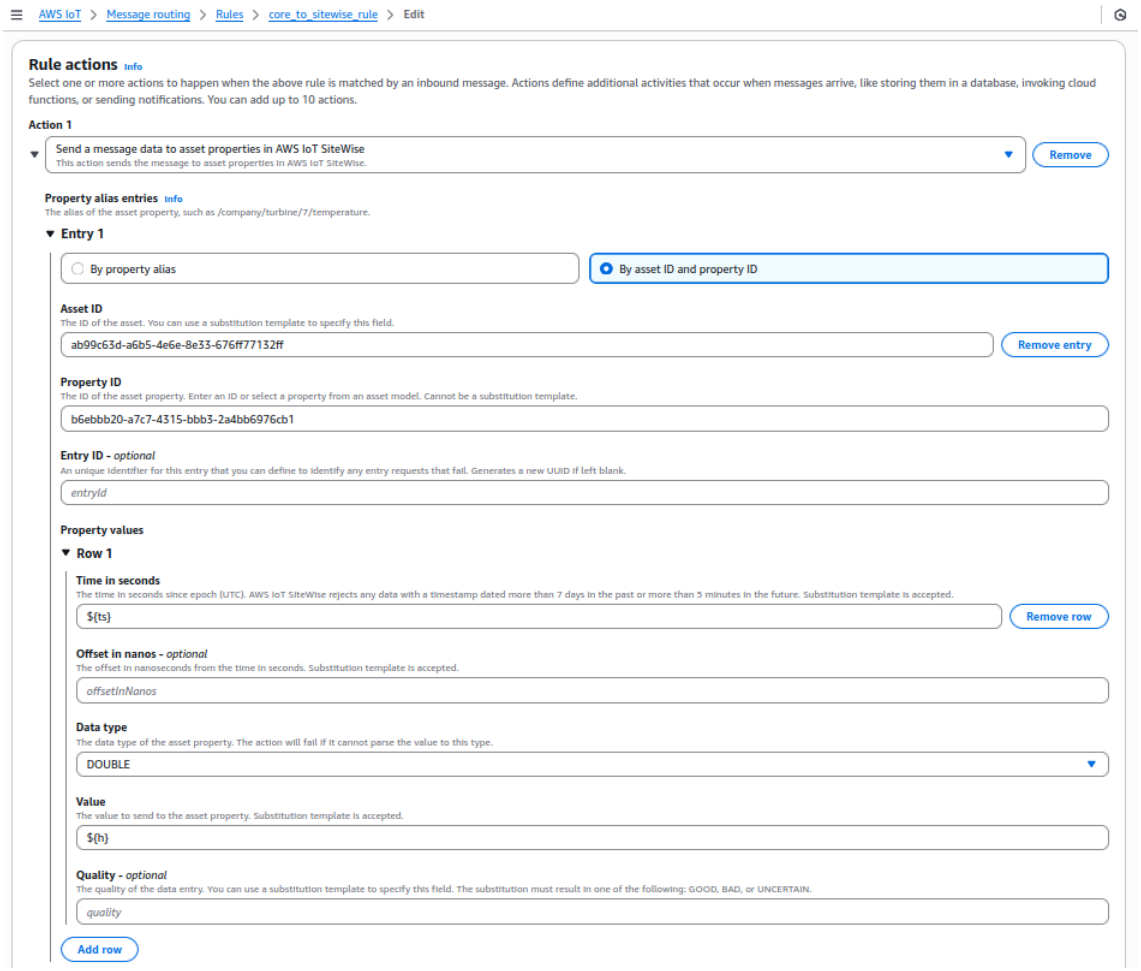

Figure 2.15: This screenshot shows an example of the configuration of a rule action.

2.3.1 IAM Role for Rule

The last thing that is needed is now to give the rule enough rights so that it is able to do everything it needs to do. For that we create a role with the right policies in AWS Identity and Access Management (IAM). The new role is named 'sitewise_ingest_role' First we created created the rule automatically, but then added additionally the policie AWSIoTSiteWiseFullAccess to this role.

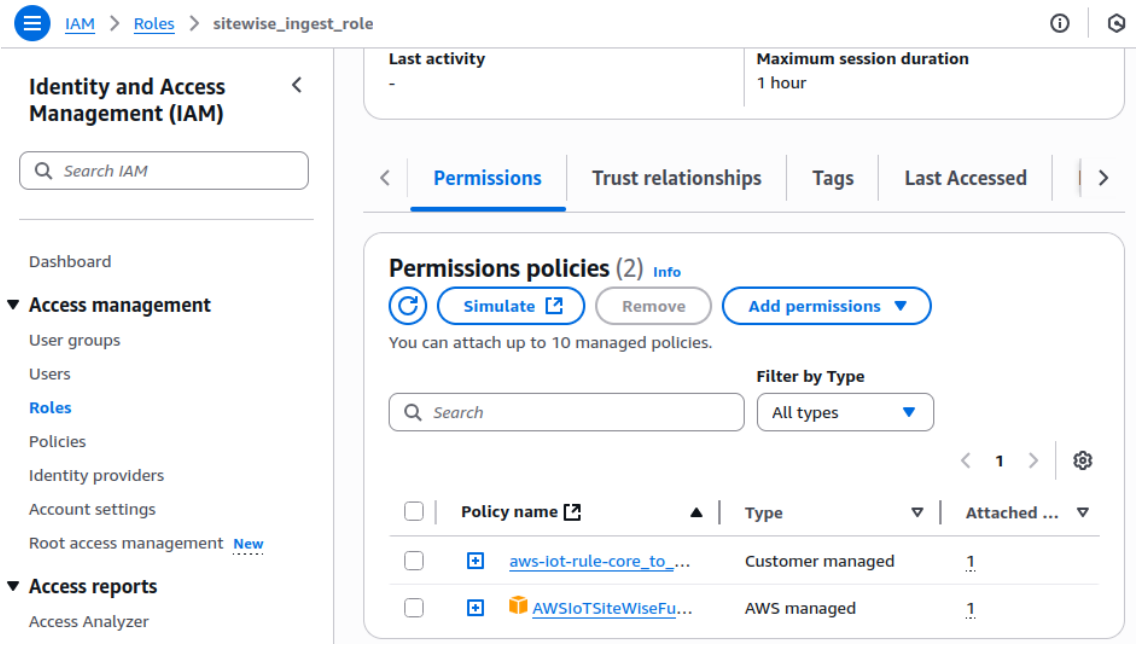

Figure 2.16: This screenshot shows an automatically created rule and the manually attached rule 'AWSIoTSiteWiseFullAccess' rule.

2.4 Create CloudWatch Action Rule for Debugging

Once a rule has been set up, the values in the associated asset should automatically update within 1–2 minutes. If the asset values are not updated as expected, it is necessary to debug the rule. For this purpose, AWS provides a dedicated service called AWS CloudWatch.

The easiest way to achieve this is by adding a new action to the rule named 'core_to_sitewise_rule'. Within this action, select 'CloudWatch log' as the action type. It is recommended to assign a descriptive name such as 'i/bme680_log_group' to the log group to make it easier to find later. After that, in the 'IAM role' field, click the 'Create new role' button. This will automatically generate a role with the necessary permissions.

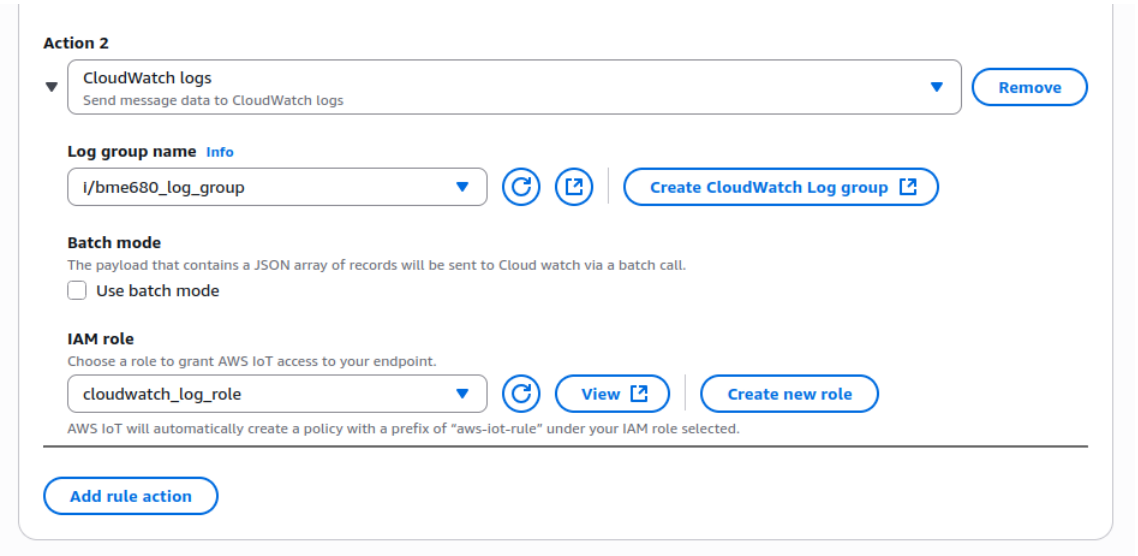

Figure 2.17: This screenshot shows how an action for debugging can be added to the rule.

Additionally, an error action can be added. This ensures that if the main action fails, the error details will be captured—for example, in a CloudWatch log group.

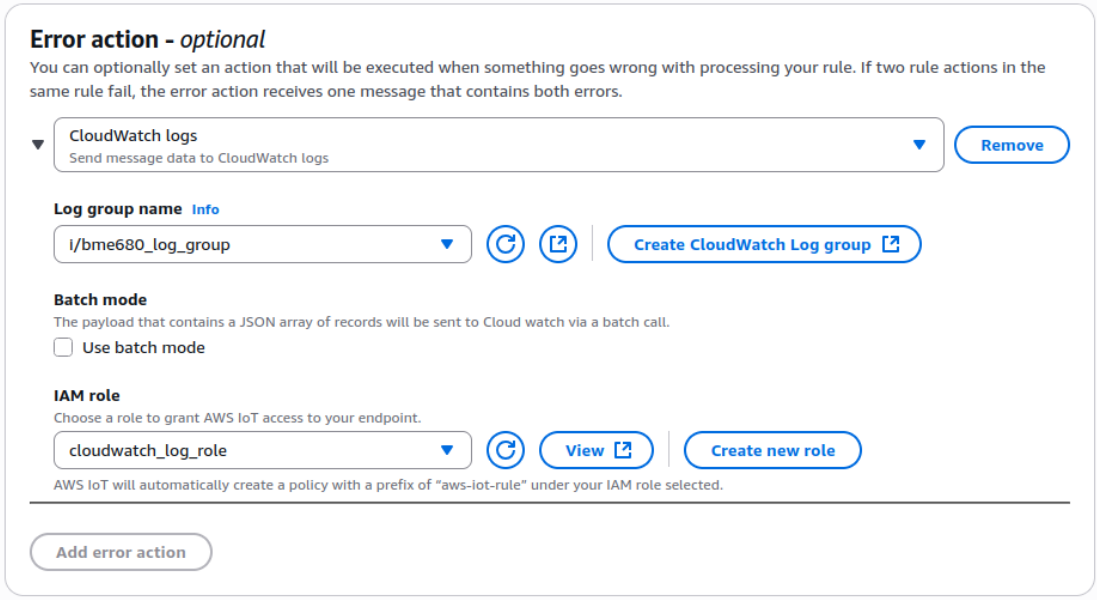

Figure 2.18: This screenshot shows how an error action can be added to the rule.

It is recommended to set the time to '1m' so that only current events are previewed. An error is visualized using a red 'failures' entry in the code. For example, the error shown here is due to an incorrect data type within an asset.

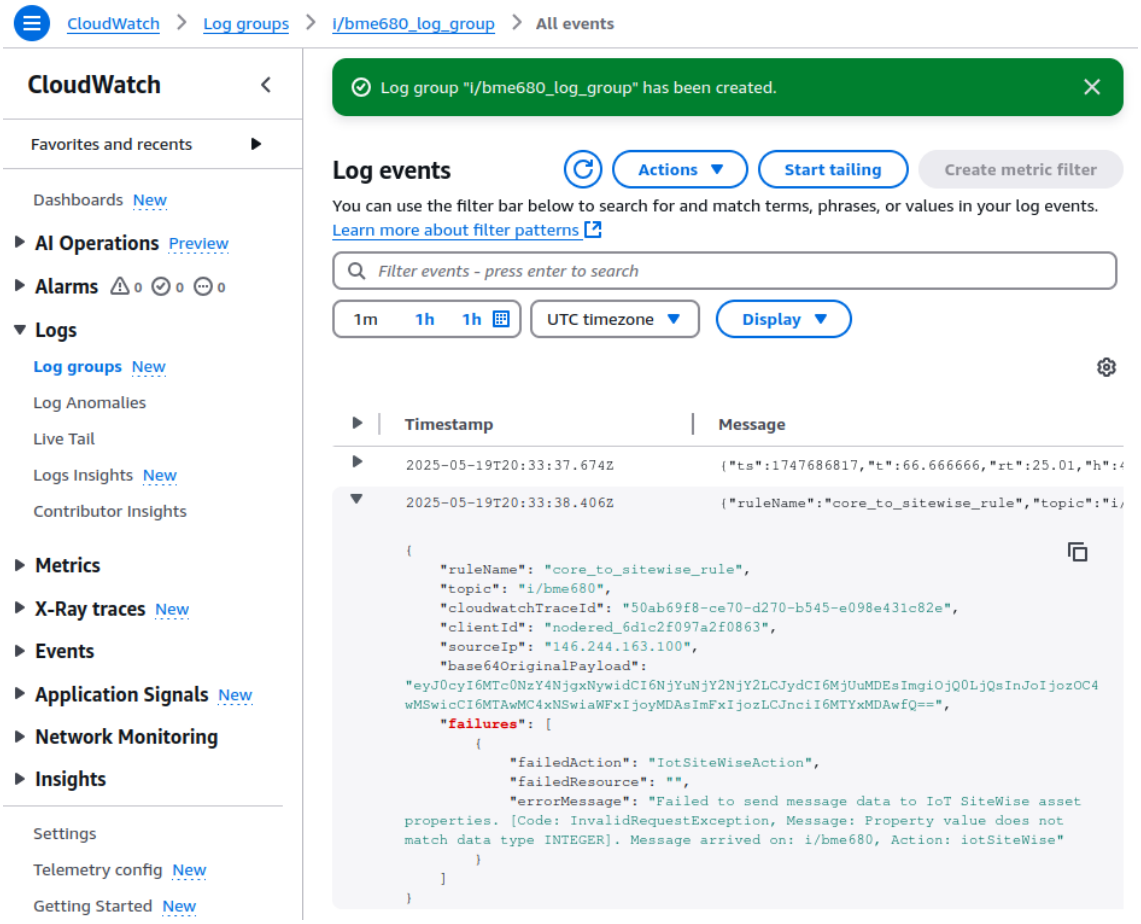

Figure 2.19: This screenshot shows how an error got tracked by the CloudWatch group.

2.5 IoT TwinMaker Role

IoT TwinMaker is a service for building digital twins of real-world systems. It allows visualizing and analyzing operational data in 3D scenes. IoT TwinMaker can retrieve data directly from IoT SiteWise to display real-time states and processes of assets.

The first step is to create an execution role to allow other services or accounts to perform actions within this account. In AWS IAM, create a new role and select 'Custom trust policy' as the trust relationship type.

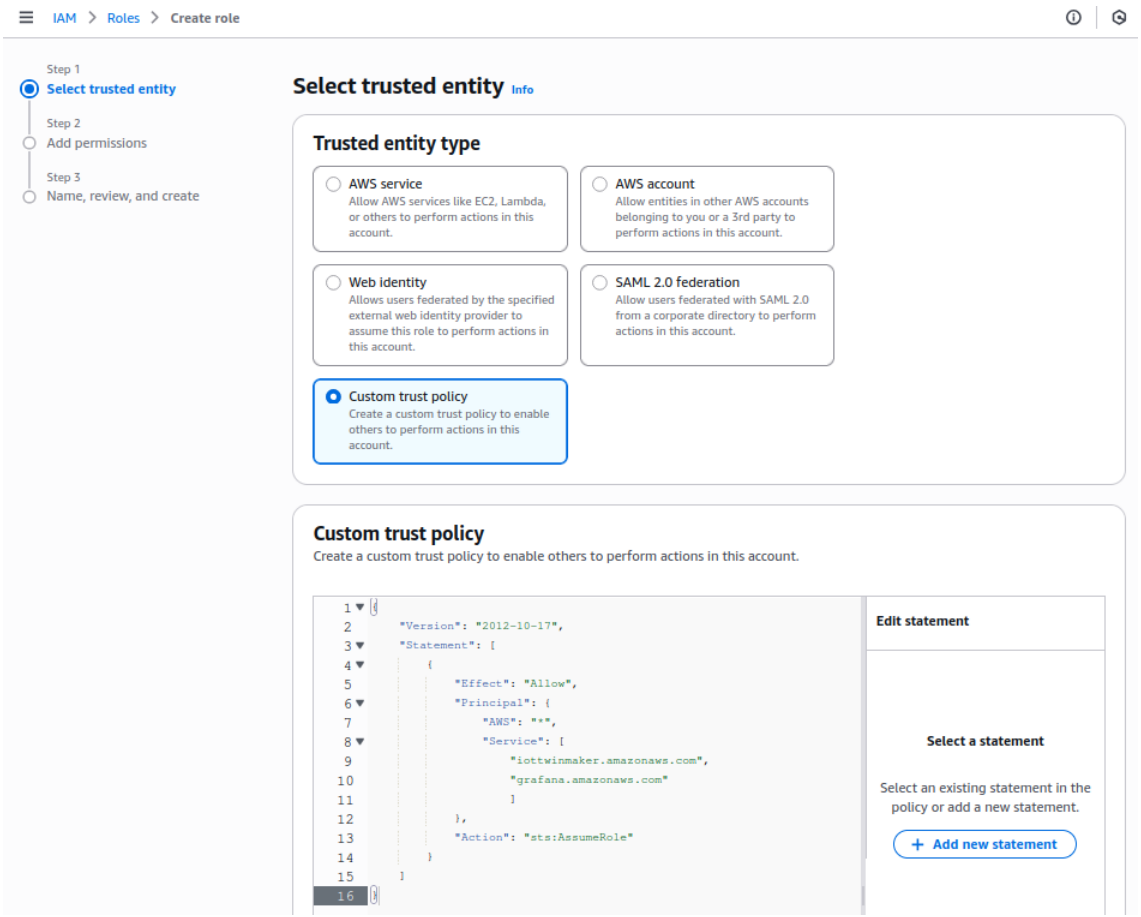

Figure 2.20: This screenshot shows how to create a new role specific for IoT TwinMaker.

This new role requires a specific inline policy that grants access to the following AWS services: IoT TwinMaker, IoT SiteWise, and Amazon S3.

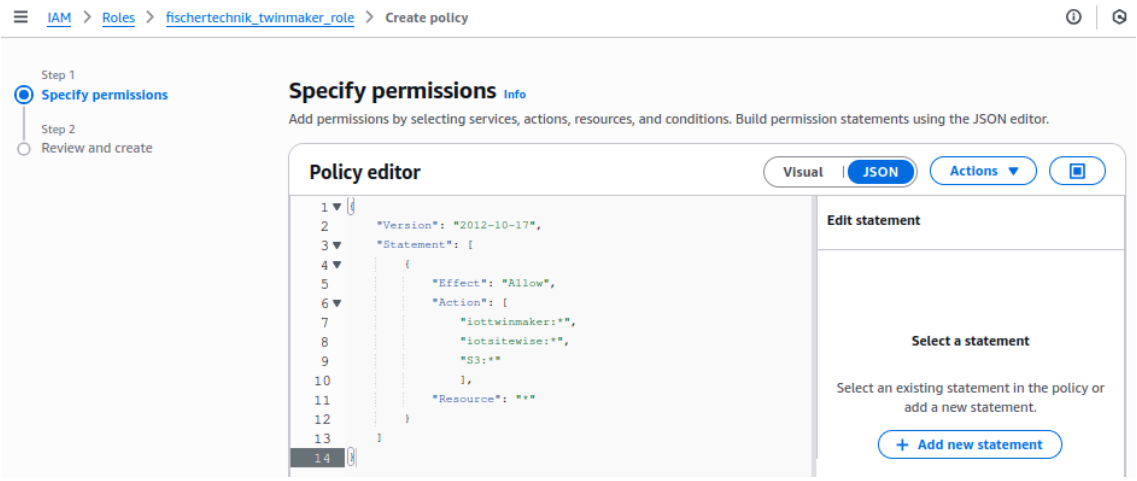

Figure 2.21: This screenshot shows how to add the needed permissions in JSON.

2.6 Create IoT TwinMaker Workspace

To visualize the data in a 3D environment, a workspace must be set up in IoT TwinMaker. Additionally, an S3 bucket is required, which can be created automatically during the setup process. IoT TwinMaker is used to create and visualize digital twins of physical systems in a 3D environment. The S3 bucket serves as a storage location for resources such as 3D models, scene configuration files, and metadata used within the IoT TwinMaker workspace.

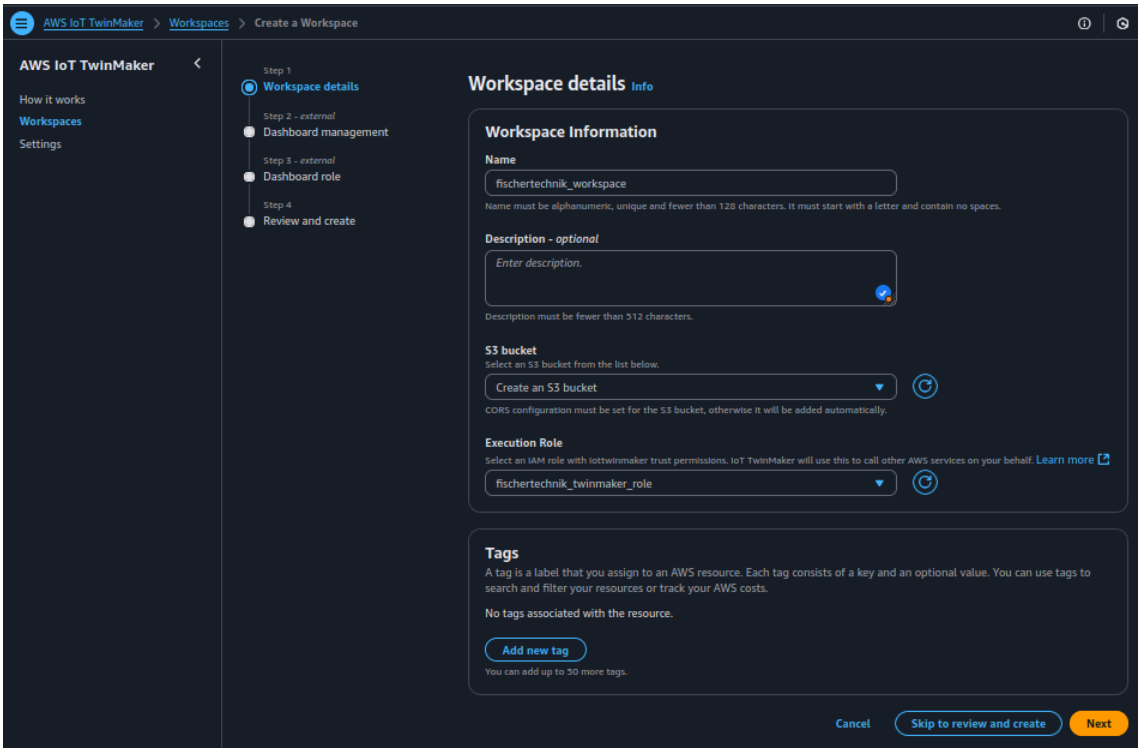

Figure 2.22: This screenshot shows where and how a IoT TwinMaker workspace can be created.

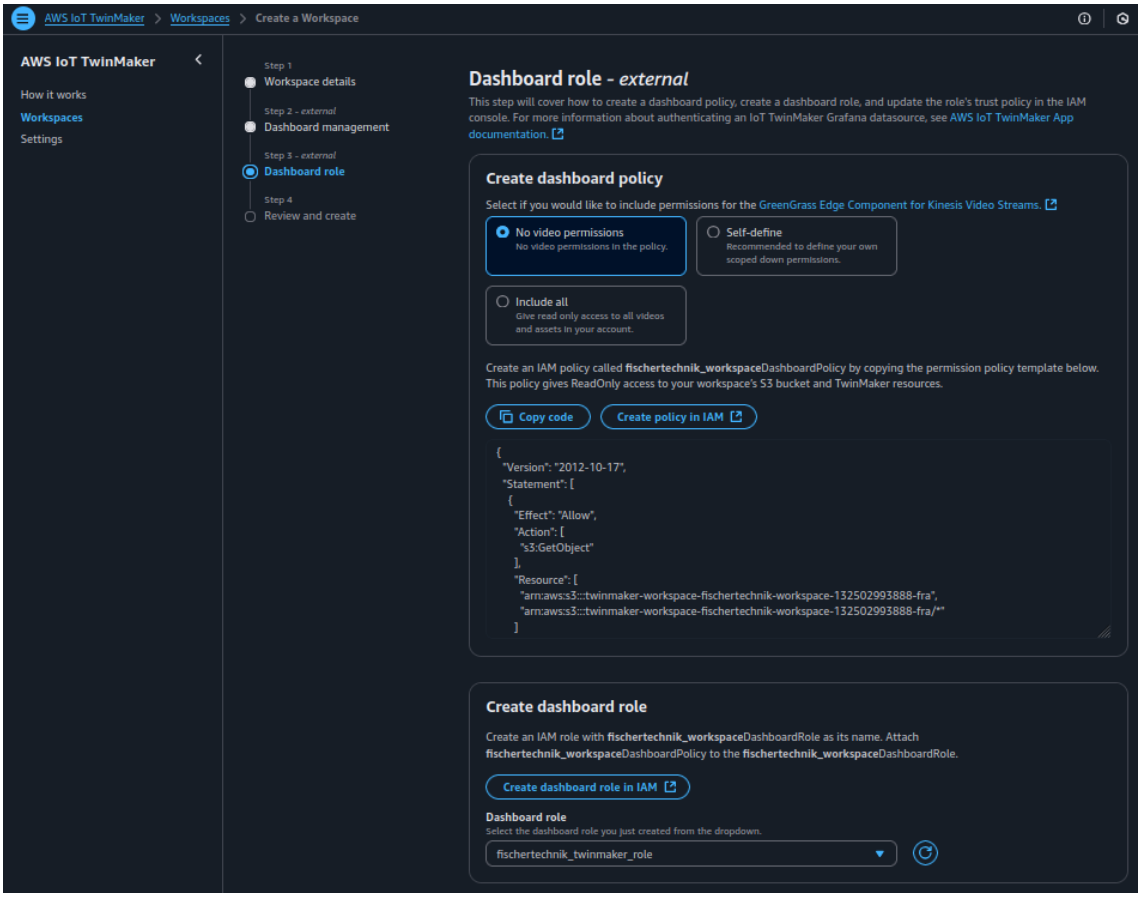

Figure 2.23: This screenshot shows how to configure the IoT TwinMaker dashboard role.

2.6.1 Create Entity in IoT TwinMaker Workspace

In IoT TwinMaker, entities represent physical or logical components of a system, such as machines, rooms, or sensors. They serve as the building blocks of the digital twin and can be linked to data, 3D models, and components to visualize real-time status and behaviour within a scene.

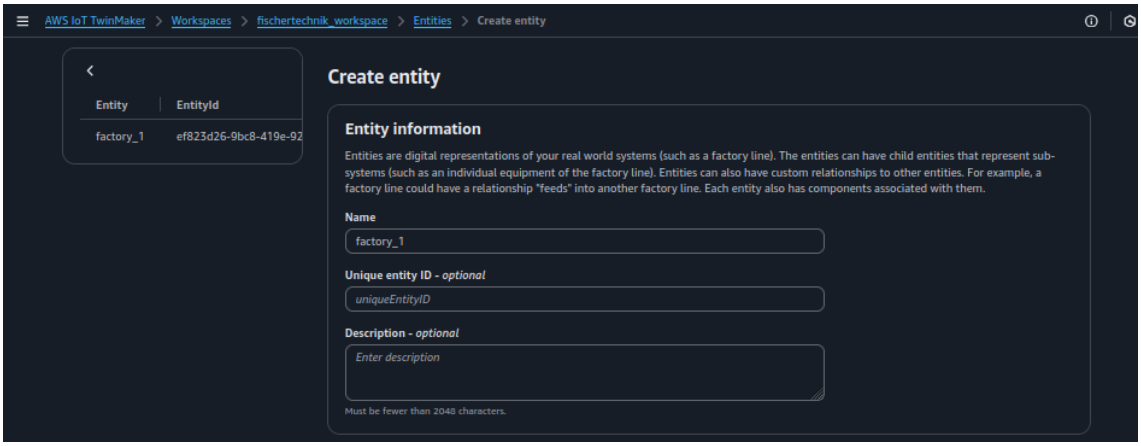

Figure 2.24: This screenshot shows how to create a Entity in IoT TwinMaker.

2.6.2 Create Component in IoT TwinMaker Workspace

Select the entity and add a component. The component should only be created once all prior steps have been successfully completed and the asset, along with the rule, contains all the required elements. This is essential, as the component is built based on this data. Every value of the assets is now automatically connected to an external storage.

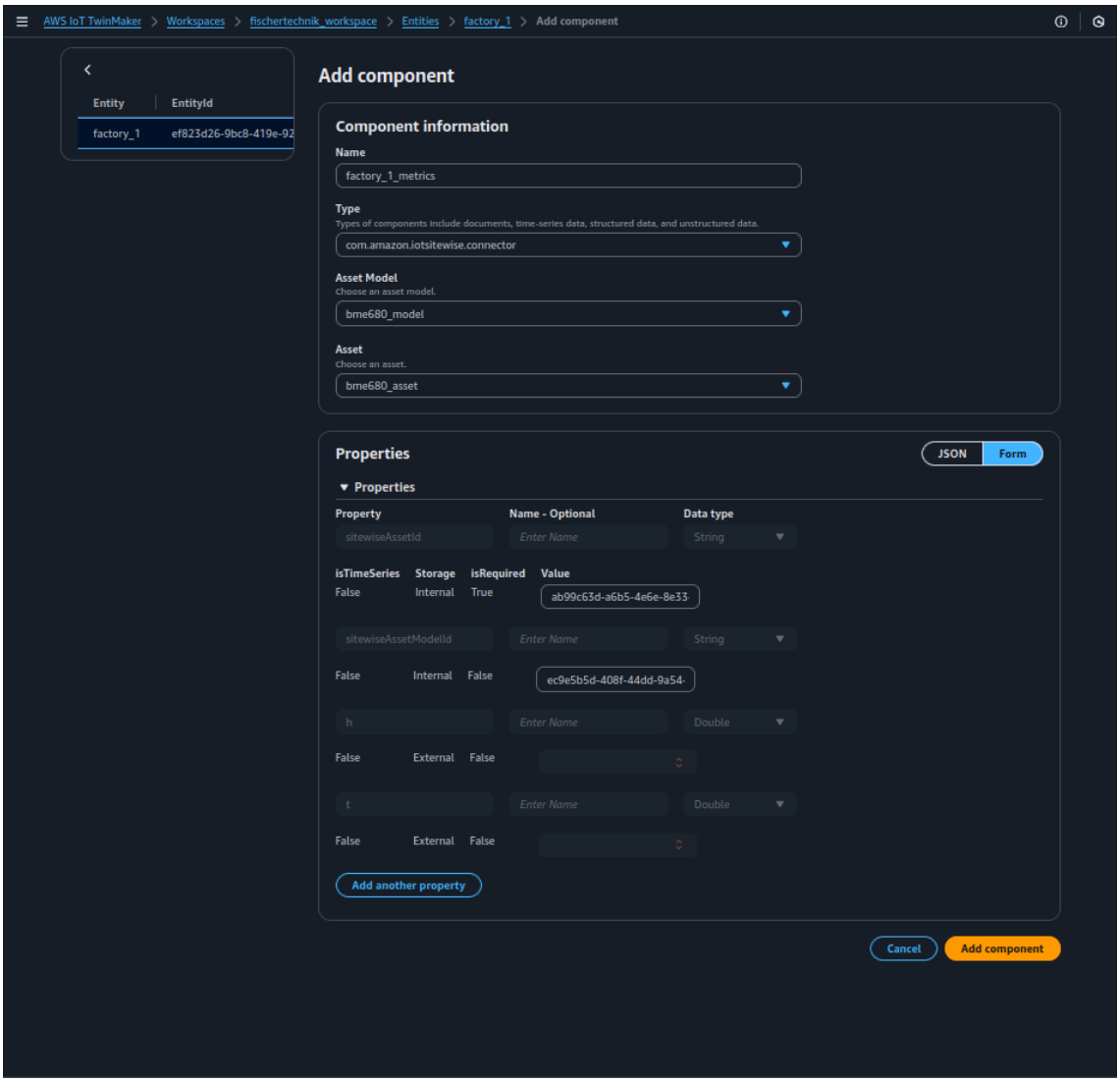

Figure 2.25: This screenshot shows how to create a Component in IoT TwinMaker.

After the component is created, it can be tested immediately.

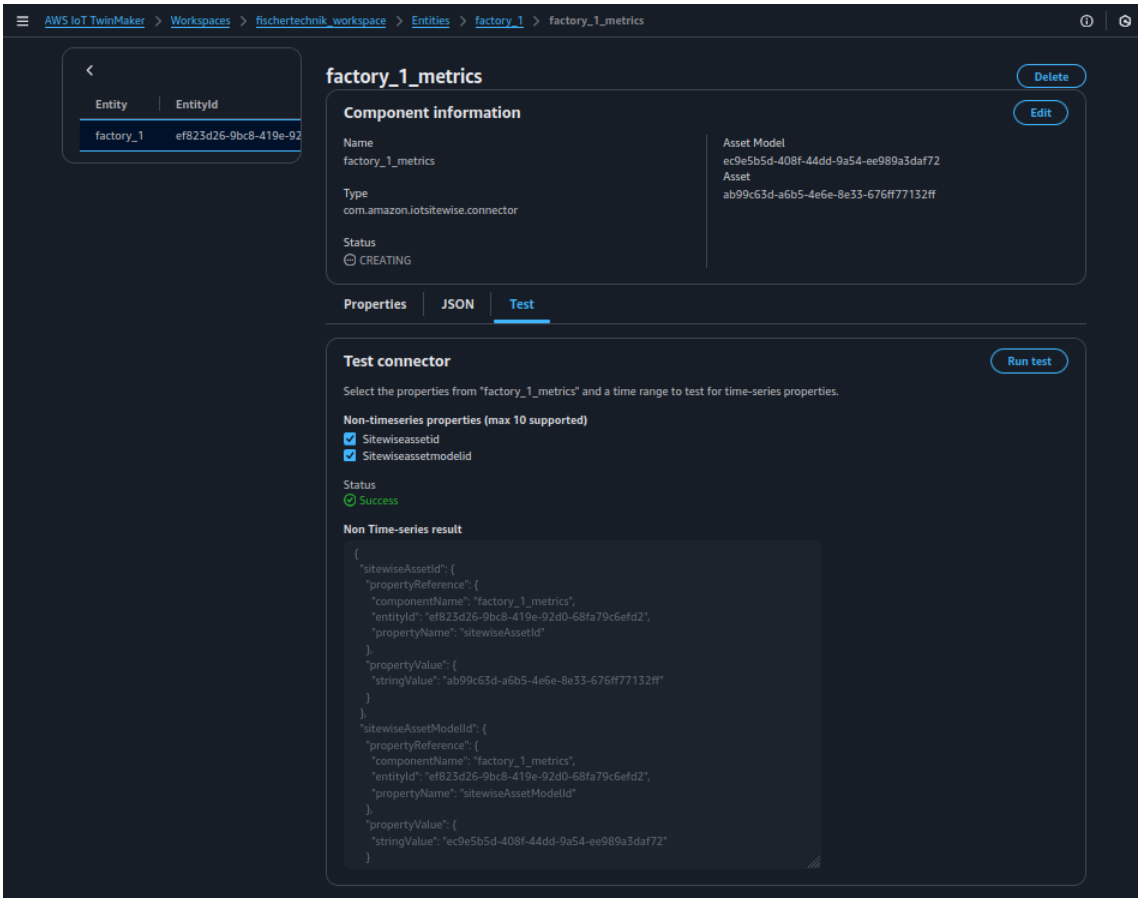

Figure 2.26: This screenshot shows how to test a previously created component in IoT TwinMaker.

2.6.3 Add a Resource

To visualize elements in a 3D environment within a IoT TwinMaker scene, it is necessary to add a 3D model. This model can be uploaded to the workspace. In this example, a factory model is used, provided in the .Graphics Library Transmission Format Binary (GLB) format. It can be downloaded in the GitHub [Bel25].

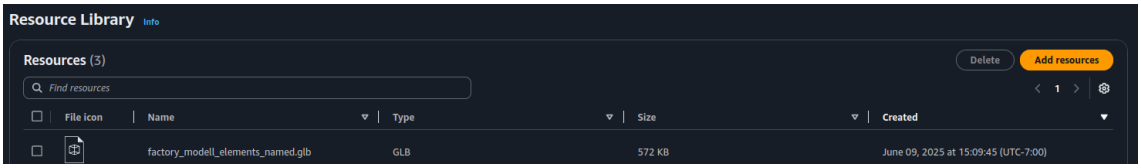

Figure 2.27: This screenshot shows how to upload a resource.

2.6.4 Create static scene

It is very important that the scene is created as a static scene! This does not mean that dynamic values cannot be visualized within the scene. A static scene is required in order to display it later with dynamic values in Amazon Managed Grafana as a panel.

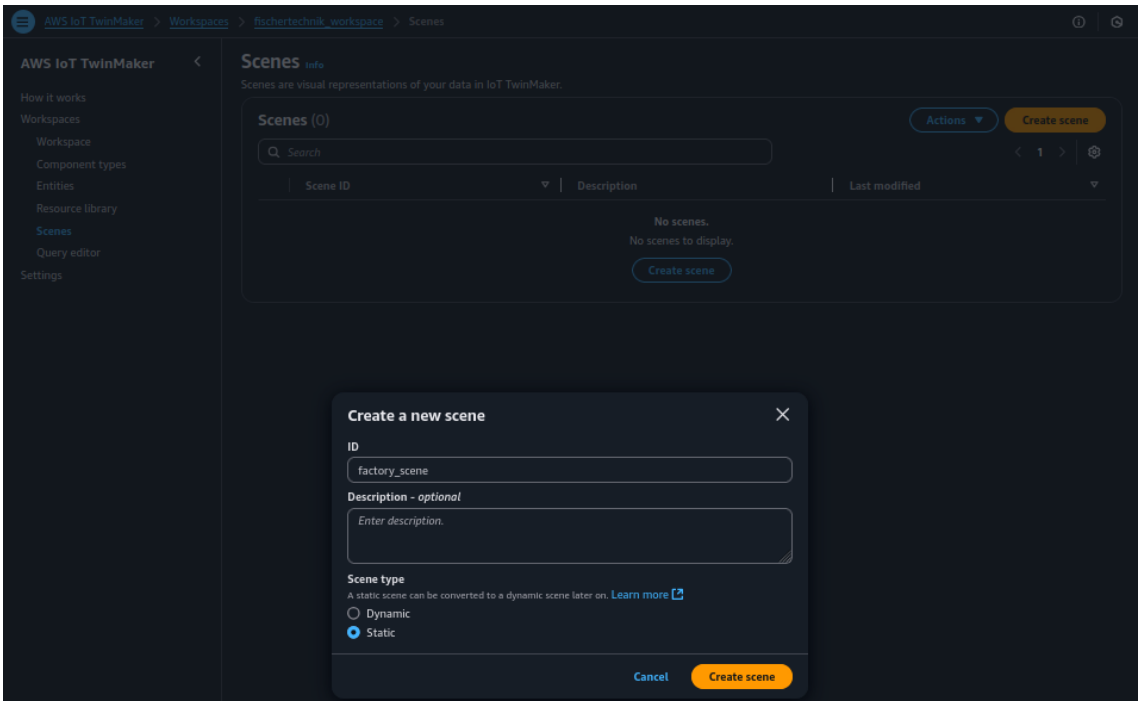

Figure 2.28: This screenshot shows how to create a static scene.

At this stage, the 3D model can be uploaded to the scene to make it visible within the environment. Select the '+' symbol to add a 3D model.

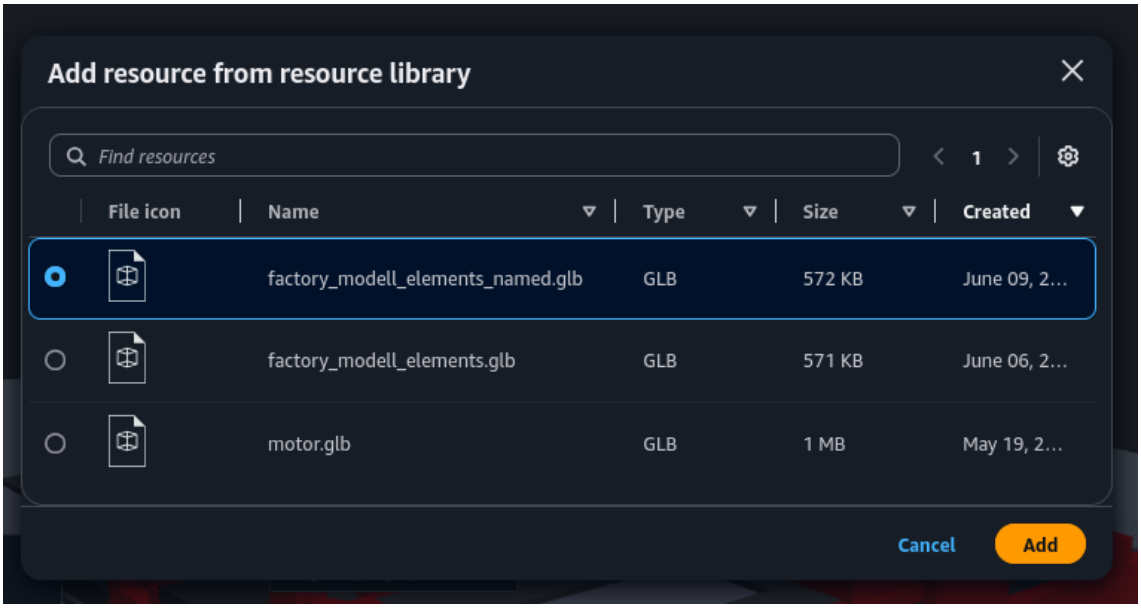

Figure 2.29: This screenshot shows how to upload a model into a static scene.

2.6.5 Create an annotation

Annotations are used to display dynamic values as labels or signs within a 3D environment. In this example a annotation is created. To populate the annotation with information to be displayed, first click on the three dots in the inspector under 'Annotation' and select the annotation. In the Markdown format section, you can now define how the data should be visualized. Next, you need to select the corresponding entity, the component name, and the property name. Finally, the binding name must be adjusted to match the variable referenced in the Markdown section. A data binding must be added for each variable and it must be connected to the correct property.

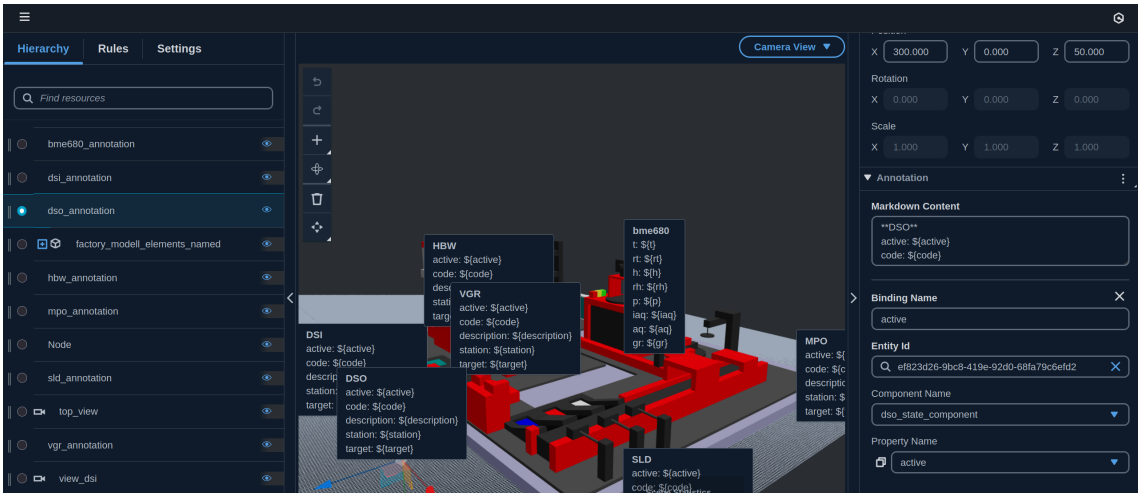

Figure 2.30: This screenshot shows how to create an annotation within the scene.

2.6.6 Change colour of object in 3D environment

A connected variable can be used to change the colour of a 3D object. This can be configured in the Rules section. In this case, a part of the 3D model appears in the colour red, blue or white, if the value of the variable is 'RED', 'BLUE' or 'WHITE'.

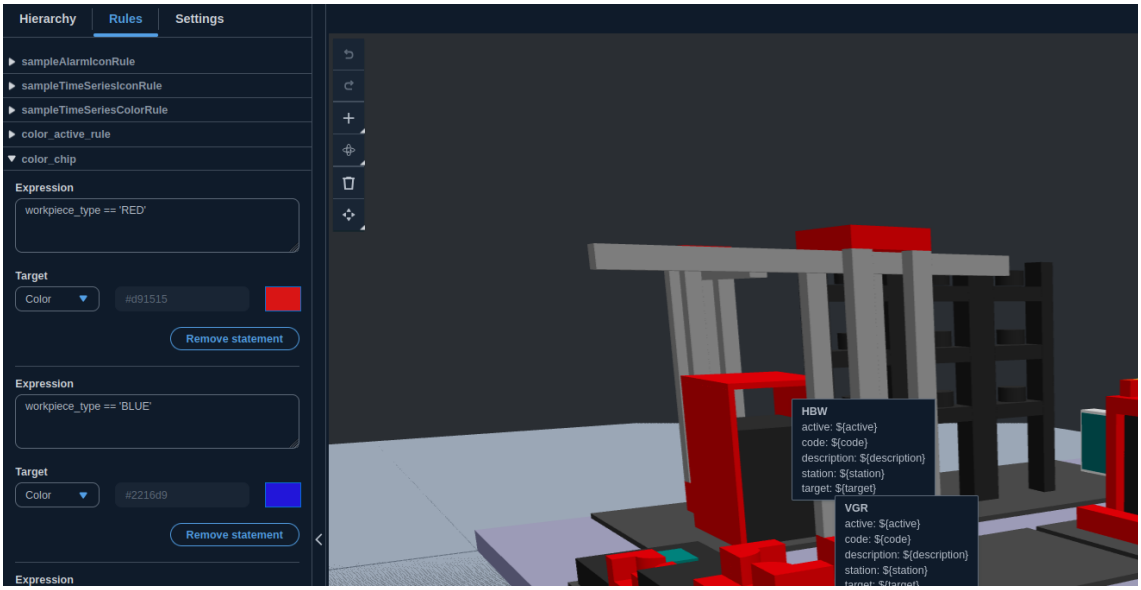

Figure 2.31: This screenshot shows how to add a colour shade in a scene.

Additionally, a model shader must be created using the '+' symbol. In this step, the corresponding variable must also be connected to the shader to enable dynamic colour changes based on its value. It is recommended to minimize the refresh rate as far as possible. It is recommended that the setting be adjusted to 5s to facilitate the visualization of the current data. To add a shader, click on the element that should change its colour. then click on the plus symbol next to the name. The element appears now in the top of the list. Here yo ucan select this element. While the element is selected, you can add a shader. On the right appears a Model shader. Here you can add an entity. If no Entity Identification (ID) is selectable, refresh the Website!.

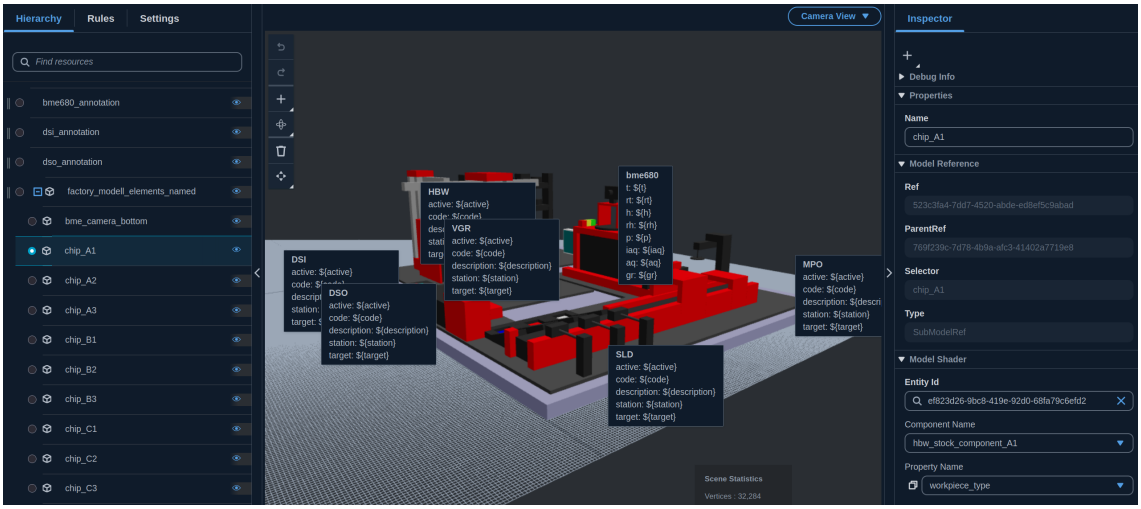

Figure 2.32: This screenshot shows how to create a model shader within the scene.

2.7 Amazon Managed Grafana

The Amazon Managed Grafana Workspace has to be created in the Version 8.4! This is not the latest version, but it is compatible with IoT TwinMaker and IoT SiteWise. Amazon Managed Grafana is an open-source analytics and visualization platform used to monitor and explore time-series data. It allows users to create interactive and customizable dashboards from various data sources like IoT SiteWise. In the context of IoT TwinMaker, it enables the integration of 3D scenes with live data panels to visualize the state of digital twin assets.

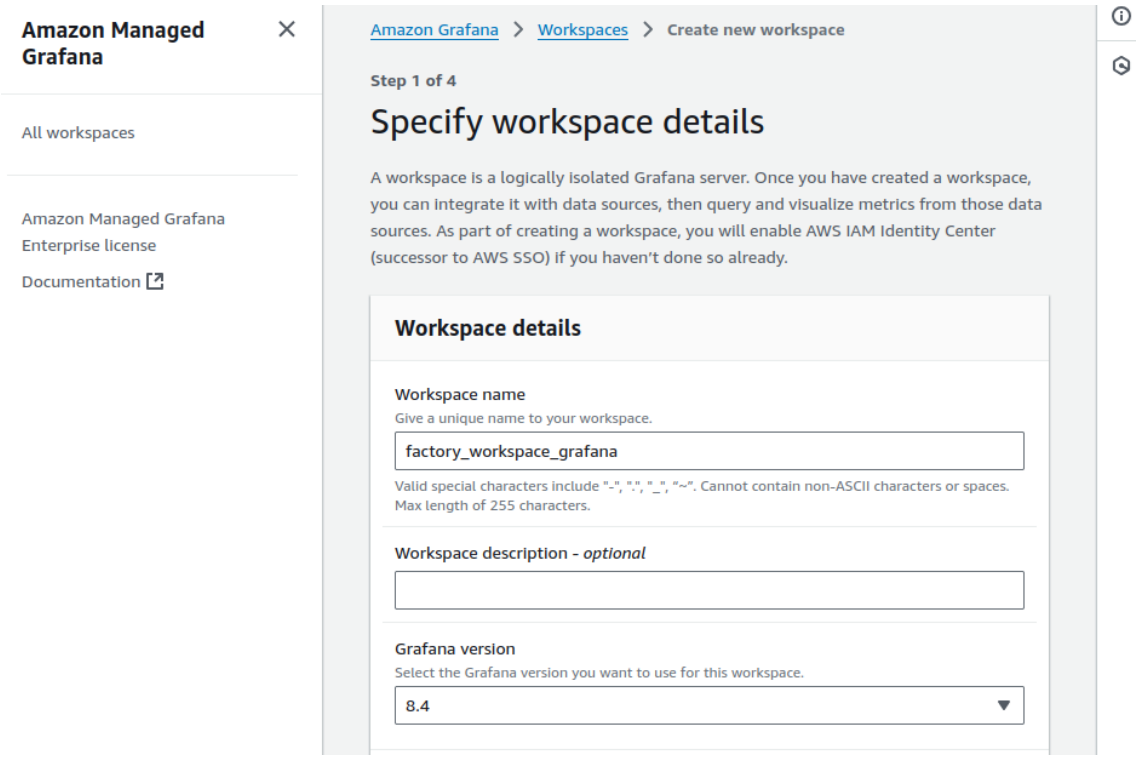

Figure 2.33: This screenshot shows how to create a Workspace in Amazon Managed Grafana.

The role that needs to be added is the previous used 'sitewise_ingest_role'. The total configuration can be seen in the following screenshot:

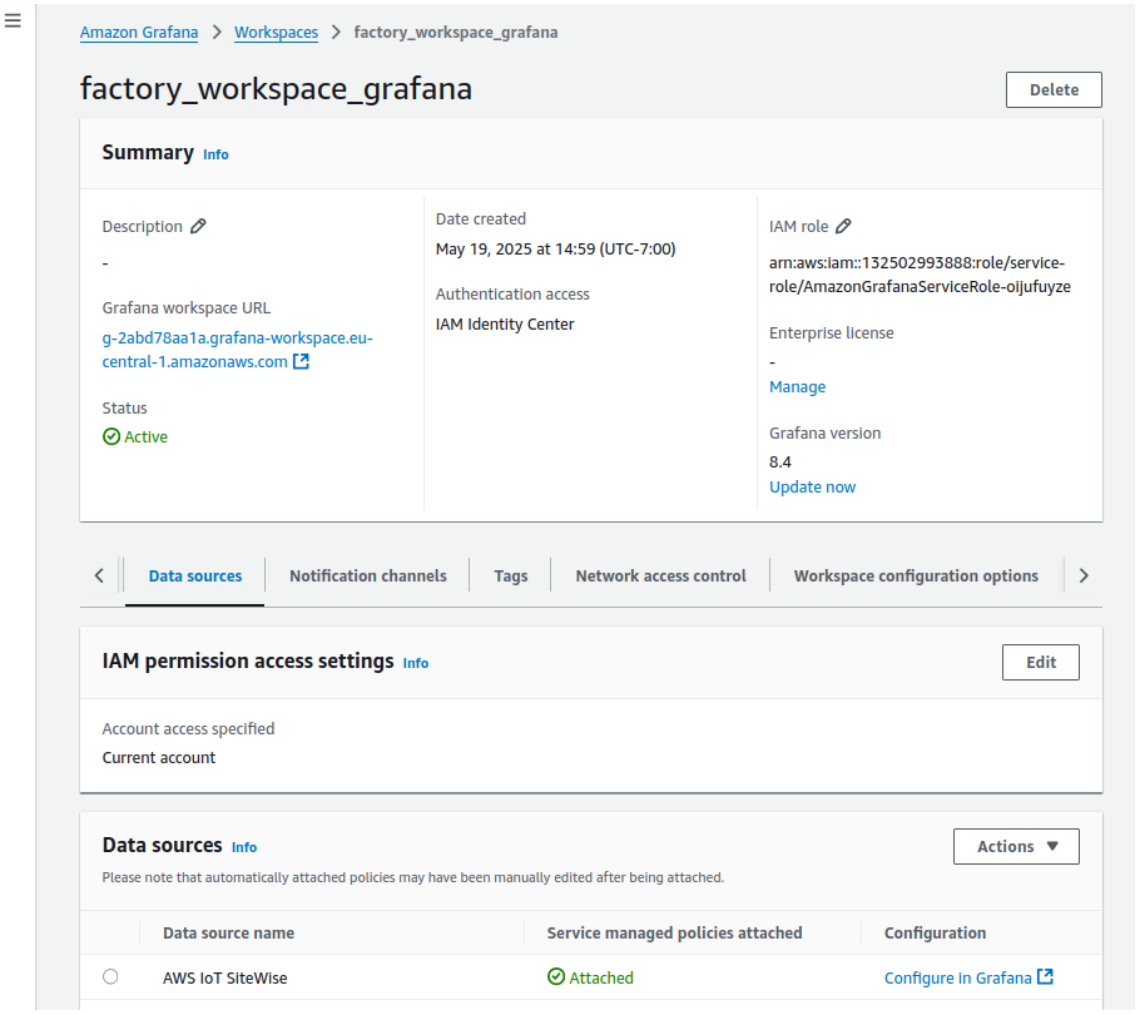

Figure 2.34: This screenshot gives an overview about the workspace configuration.

2.7.1 Create User for Final Dashboard

A user must be created in AWS IAM to securely access the Amazon Managed Grafana dashboard and retrieve data from services like IoT TwinMaker or IoT SiteWise. This ensures controlled access and proper permission management. It is very important to give this user (in this example) admin rights. This allows the user to create and adjust the dashboard.

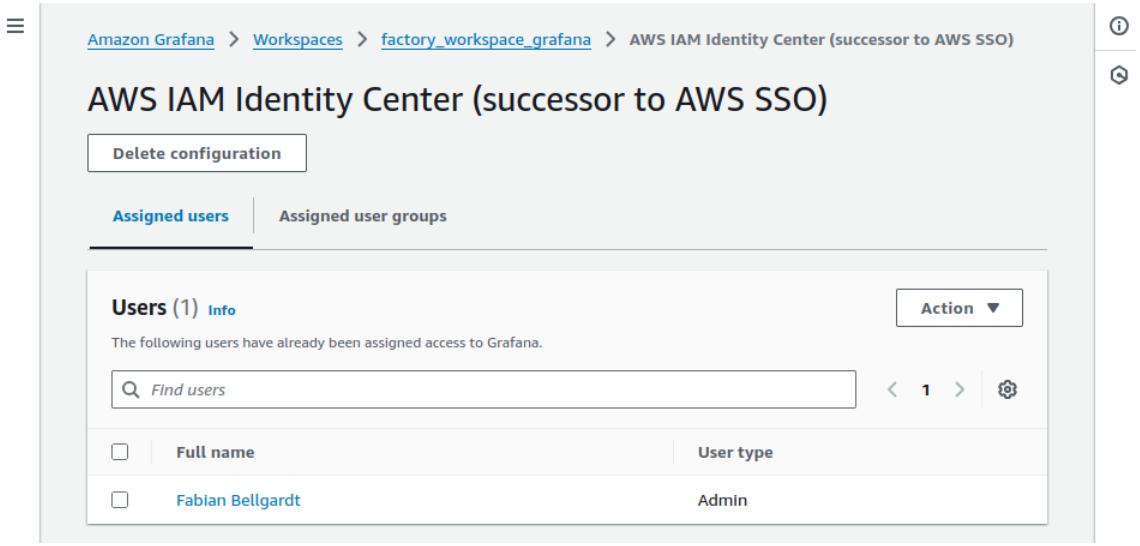

Figure 2.35: This screenshot shows how to create a user for the Amazon Managed Grafana dashboard.

2.7.2 Add Datasource

After the admin user logs in to Amazon Managed Grafana, a data source must be added. To add a data source, click on the gear symbol and search for IoT TwinMaker. The following screenshot shows the recommended configuration. An Amazon Resource Name (ARN) must be provided—specifically, the ARN of the role created for IoT TwinMaker. This ARN can be found in AWS IAM. It is also important to select the correct region, which should match the region where the previously created services are located. After saving the configuration once, the IoT TwinMaker workspace becomes selectable.

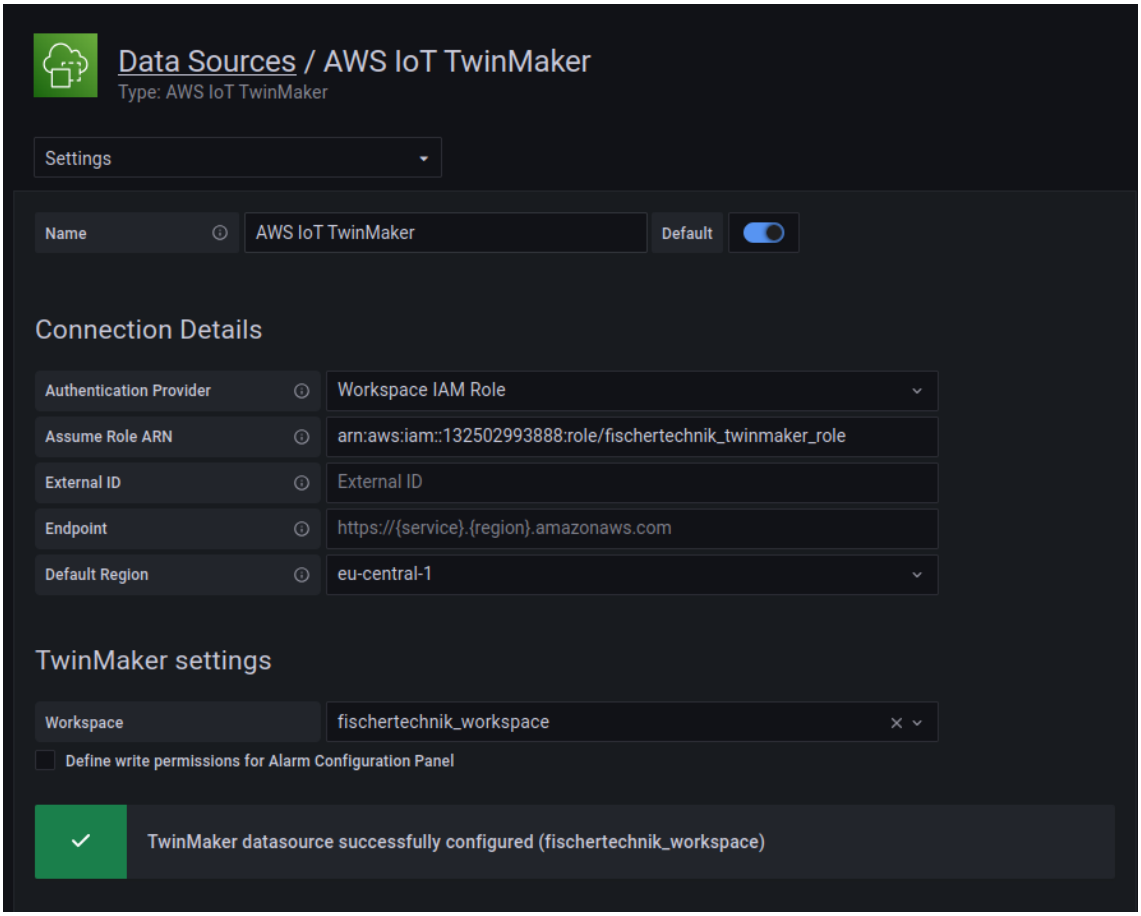

Figure 2.36: This screenshot shows how to add IoT TwinMaker as a data source in Amazon Managed Grafana.

2.7.3 Create a Dashboard in Amazon Managed Grafana

Amazon Managed Grafana offers different options of panels to add into a dashboard. Those panels can be sorted in rows. It is important to click the floppy disk icon to save the entire dashboard.

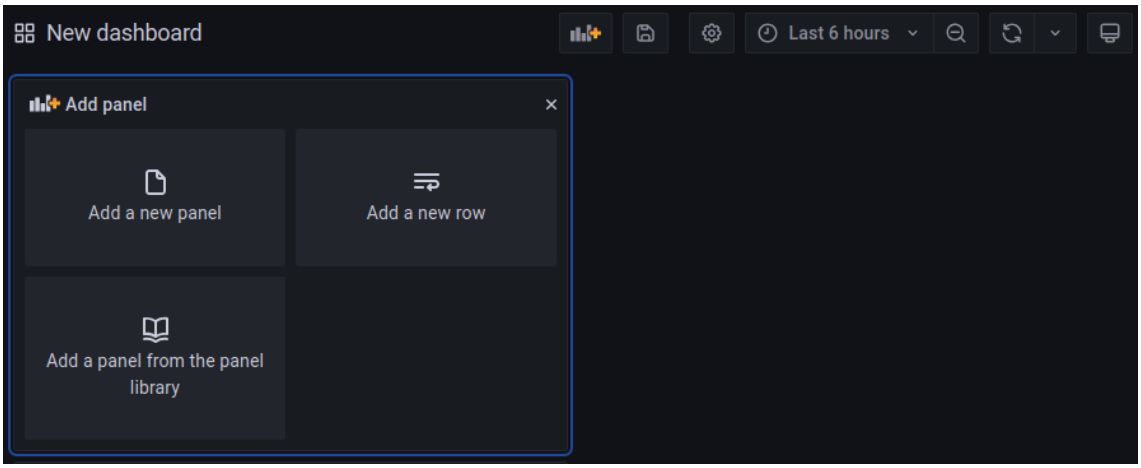

Figure 2.37: This screenshot shows how to add a dashboard to Amazon Managed Grafana.

In one of those panels, 'IoT TwinMaker Scene Viewer' can be chosen. Here we have to add the query type, entity, component name and the selected properties.

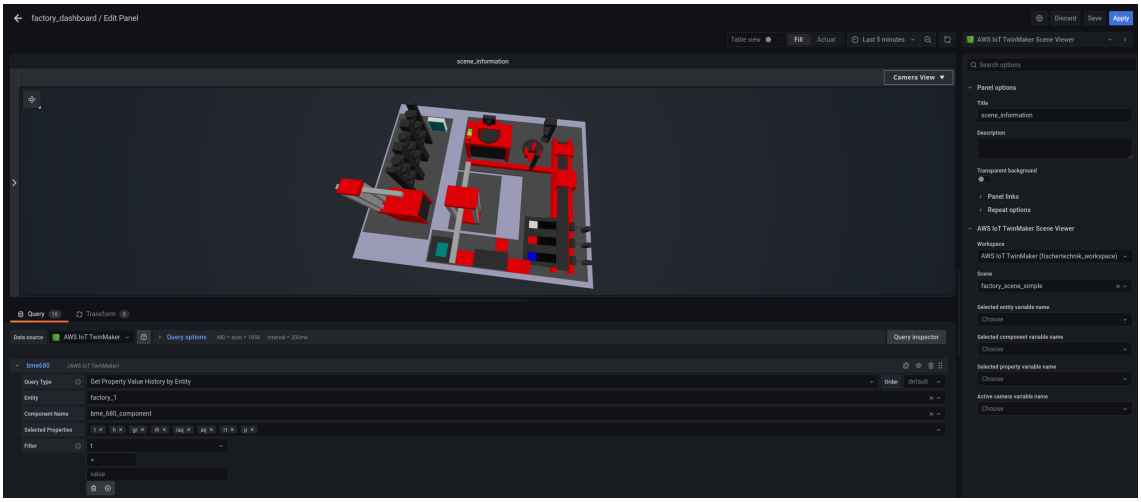

Figure 2.38: This screenshot shows how to connect data to a panel. This panel has to be updated for every annotation and data used in the IoT TwinMaker scene.

The following screenshot shows the final dashboard. This dashboard is now able to show dynamically informations in a 3D environment. This is the fundament for a digital twin and the visualisation of the Fischertechnik factory.

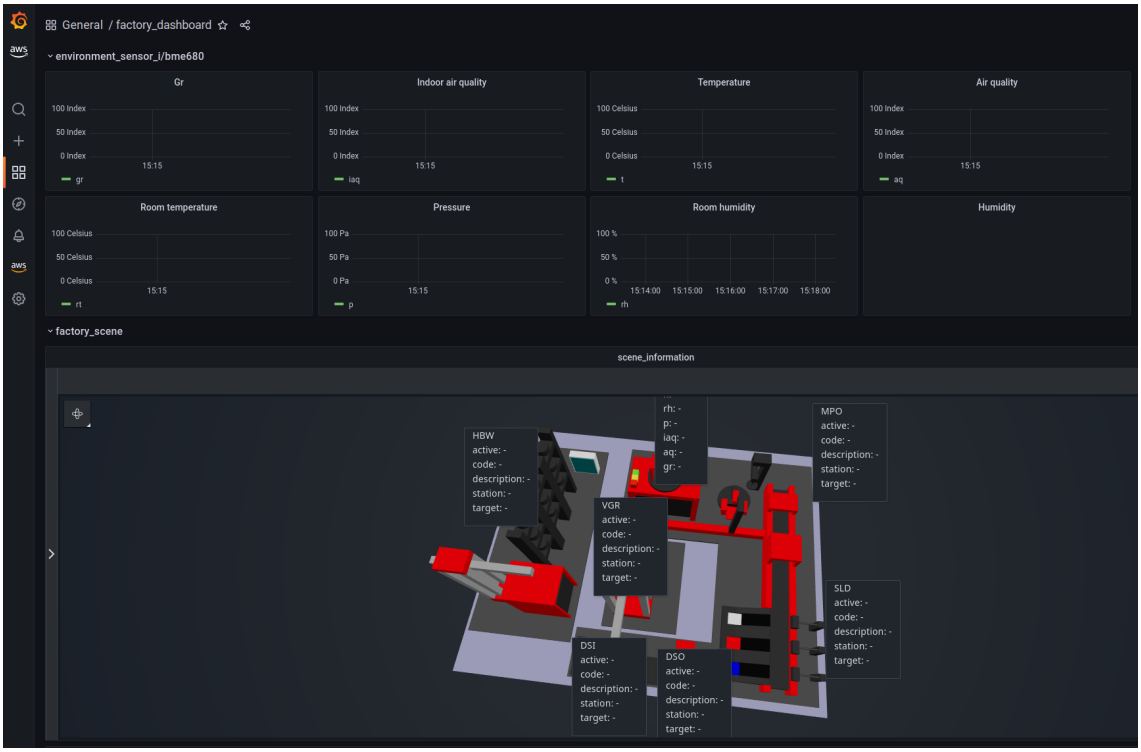

Figure 2.39: This screenshot shows the final Amazon Managed Grafana dashboard with dynamic informations.

A Further Resources

For further sources, it is recommended to visit the following GitHub link. There you will find, among other things, the Node-RED JSON files for the EC2 instance and the Node-RED instance at the factory [Bel25].

List of Figures

List of Tables

Shortcuts

AWS Amazon Web Services

IoT Internet of Things

EC2 Elastic Compute Cloud

MQTT Message Queuing Telemetry Transport

3D Three-Dimensional

SQL Structured Query Language

TLS Transport Layer Security

ISO International Organization for Standardization

VPC Virtual Private Cloud

NAT Network Address Translation

IP Internet Protocol

HTTP Hypertext Transfer Protocol

OS Operating System

SSH Secure Shell

JSON JavaScript Object Notation

IAM Identity and Access Management

ARN Amazon Resource Name

Wi-Fi Wireless Fidelity

BME Bosch Meteorological Environmental

S3 Simple Storage Service

GLB Graphics Library Transmission Format Binary

ID Identification

URL Uniform Resource Locator

Bibliography

[Bel25] Fabian Bellgardt. *Fischertechnik Digital Twin and Dashboard Implementation Using AWS*. 2025. URL: https://github.com/FabianBellgardt/Fischertechnik-Digital-Twin-and-Dashboard-Implementation-Using-AWS.git.