FINAL REPORT

UNIVERSITI TEKNOLOGI PETRONAS

MAY 2025 SEMESTER

RBB2013: DIGITAL TWIN

LECTURER: TS. DR HO TATT WEI

GROUP MEMBERS:

NO.	NAME	STUDENT ID
1	VEEBHAKER KALAI SELVAN	22007032
2	MUHAMMAD KHAIRY ANWAR BIN MOHD FARIZ	22006361
3	SITI NUR FATIHAH BINTI IMRAN	22007814
4	MOHD IZZUDIN BIN ABDUL RAHIM	22003651

DEMOSTRATION VIDEO LINK:

1	https://drive.google.com/file/d/1bQC952jhbV_eV67uTmyUNDJ72izoJjSf/view?usp=sharing

Table of Contents

Table of Contents

CLO1 Dataset Identification

Dataset Name: Logistics Risk and Delay Dataset

Source: Logistics and Supply Chain Dataset – Kaggle

Purpose: To simulate and monitor real-time logistics operations using a Digital Twin (DT) model integrated with AWS services, particularly AWS IoT TwinMaker.

Why This Dataset Was Chosen

This dataset was selected because it contains time-stamped, real-world logistics data that can be directly streamed into AWS IoT TwinMaker. It includes sensor-like inputs (e.g., GPS, fuel rate, driver behavior), contextual features (e.g., traffic level, weather), and target variables (risk_classification, delivery_time_deviation) that support predictive modelling. The format is clean, flat, and scalable for both batch processing and real-time telemetry, making it ideal for building a Digital Twin.

Digital Twin Problem Context

The Digital Twin system aims to represent, monitor, and predict the behaviour of logistics vehicles under real-world conditions. The DT processes time-stamped sensor data to:

Predict risk classification (Low, Moderate, High)

These outcomes enable logistics operators to take pre-emptive actions, such as rerouting vehicles, prioritizing critical deliveries, and optimizing driver assignments in real time

Justification of Dataset Suitability

1. IoT-Compatible Structure and Temporal Data

- Field: timestamp
- Purpose: Enables chronological simulation of vehicle events
- Benefit: Essential for streaming to AWS IoT Core and replaying sensor flows in TwinMaker

2. Sensor-Mapped Features Reflect Real-World Data

The dataset includes multiple fields that represent data typically collected from IoT devices:

Dataset Field	IoT Device Equivalent	Use in DT Model
vehicle_gps_latitude vehicle_gps_longitude	GPS Tracker	Real-time location and movement monitoring
fuel_consumption_rate	Telematics Unit	Operational cost and efficiency modeling
fatigue_monitoring_score	In-Cabin Sensor	Driver safety and alertness prediction
driver_behavior_score	Behavior Scoring System	Risk factor input for delivery reliability
traffic_congestion_level	External Traffic API	Predictive delay indicator
weather_condition_severity	Environmental Sensor/API	Influence on delivery risk and timing
cargo_condition_status	Cargo IoT Sensor	Condition monitoring of sensitive shipments
iot_temperature	Environmental Sensor	Contextual factor for cargo and vehicle state

These features allow for direct integration with AWS IoT Core and TwinMaker as entity properties and telemetry streams.

3. Predictive Relevance of Features

- Target Variables:
    - risk_classification: used for classification of delivery risks
- Supporting Features:
    - Fields like driver_behavior_score, route_risk_level, and disruption_likelihood_score strongly influence prediction outcomes
    - These are critical for training machine learning models in SageMaker and updating TwinMaker with predicted values

This makes the dataset ideal for ADigital Twins, where prediction feeds real-time decision-making.

4. Structured, Lightweight Format for Cloud Integration

- CSV format, flat structure (no nested objects)
- Easily parsed by AWS Lambda, streamed via IoT Core, and consumed by TwinMaker without transformation
- Scalable for batch and real-time ingestion

5. Supports Both Static and Dynamic Digital Twin Views

- Static Data Examples: VehicleID
- Dynamic Data Examples: fuel rate, fatigue score, cargo condition (streamed as telemetry)
- Allows full DT lifecycle: initialization → monitoring → prediction → optimization

Conclusion

The Logistics Risk and Delay Dataset is exceptionally well-suited for building a Digital Twin in AWS. It:

- Contains temporally structured, sensor-analogous inputs that match real-world logistics data flows
- Enables real-time and predictive modeling of logistics outcomes through fields tailored for classification and regression
- Is directly mappable to TwinMaker components and properties for visualization and analysis
- Facilitates seamless integration with AWS services including IoT Core, Lambda, SageMaker, and Timestream

Specifically, the data elements chosen are structured to stream temporally as IoT signals through AWS IoT Core, enabling dynamic updates to the digital twin and real-time synchronization between physical operations and their virtual counterparts.

This dataset doesn't just support a Digital Twin, it unlocks predictive intelligence and operational agility within it.

CLO1 Dataset relevance to problem

Dataset Name: Logistics Risk and Delay Dataset

Source: Logistics and Supply Chain Dataset – Kaggle

Use Case: Digital Twin of a logistics delivery operation to simulate, predict, and respond to real-time conditions.

Problem Statement and Digital Twin Outcome

The core problem addressed by this Digital Twin (DT) model is the real-time prediction of delivery delays and their associated costs in a dynamic logistics environment. In the modern world, logistics providers often face unexpected disruptions due to traffic congestion, driver fatigue, fuel inefficiency, and weather conditions, all of which contribute to late deliveries and increased operational expenses.The primary outcomes of this Digital Twin are : Predictive classification of delivery risk level (Low, Moderate, High)

Qualitative Justification of Dataset to Support Digital Twin (DT) Outcomes

The selected **"Logistics Risk and Delay"** dataset, adapted from Kaggle, is well-suited to simulate the types of real-time and periodic inputs expected in a logistics-based digital twin (DT) system. Although the original dataset is structured with **hourly timestamps**, it was modified to simulate **10-second intervals** for the purpose of demonstrating real-time streaming in the DT prototype using AWS SiteWise and TwinMaker.

The features were reorganized into three categories based on realistic sensor behaviour: **Fixed**, **Realtime**, and **Periodic**. This supports accurate simulation of IoT telemetry and predictive insights in the DT environment.

Fixed Features (Sent once per vehicle/trip setup)

Feature	**Sensor Type / Source**	**Purpose in DT Model**
lead_time_days	ERP / logistics planning	Provides baseline delivery expectations
supplier_reliability_score	Supplier analytics system	Historical vendor performance metric
port_congestion_level	Port data / API feeds	Indicates likelihood of port-related delays
weather_condition_severity	Weather APIs	Affects driving conditions and risk
route_risk_level	Historical route analysis	Impacts probability of disruption
handling_equipment_availability	Warehouse/port system	Operational readiness signal
historical_demand	ERP historical records	Helps determine urgency or priority
driver_behavior_score	Vehicle telematics	Influences risk and efficiency assessment
warehouse_inventory_level	Warehouse IoT system	Determines cargo availability and scheduling flexibility
loading_unloading_time	Port or dock sensors	Affects trip duration and scheduling

Realtime Features (Sent every 10 seconds)

Feature	Sensor Type / Source	Purpose in DT Model
vehicle_gps_latitude	GPS module	Provides real-time geolocation for tracking
vehicle_gps_longitude	GPS module	Used for dynamic route visualization
fuel_consumption_rate	Vehicle telematics	Indicates vehicle efficiency and operational costs
traffic_congestion_level	External traffic APIs	Monitors expected versus actual transit delays
iot_temperature	Cargo temperature sensor	Important for perishable or sensitive goods
cargo_condition_status	Cargo monitoring device	Detects spoilage, vibration, or damage during transit
trip_duration	System time	Measures elapsed travel time

Periodic Features (Sent every 60 seconds)

Feature	Sensor Type / Source	Purpose in DT Model
fatigue_monitoring_score	In-cabin driver monitoring	Used to assess safety and potential for delay
disruption_likelihood_score	ML risk predictor	Aggregated early warning based on multiple signals
eta_variation_hours	ETA analytics	Indicates deviation from planned arrival time
order_fulfillment_status	Supply chain systems	Tracks cargo progress through supply chain
shipping_costs	ERP / finance system	Monitors fluctuating delivery cost
customs_clearance_time	Port/customs system	Can cause bottlenecks in international logistics
delay_probability	Predictive model	Used to alert potential delays based on conditions
delivery_time_deviation	Historical outcome data	Ground truth label for regression models

prediction	ML classifier	Final output: “Low Risk”, “Moderate Risk”, or “High Risk”

Note: Although prediction is a string class label, it is stored in SiteWise as a numerical (double) for integration compatibility.

Timestamp Conversion for Streaming Demo

- **Original dataset**: Hourly records
- **Simulated format**: 10-second intervals
- Realtime features are sent every 10 seconds
- Periodic features (including prediction) are sent every 60 seconds
- This approach allows faster feedback loops and better visualization in the digital twin dashboard

Why This Dataset is Perfect for Building an AI-Powered Digital Twin: IoT-Compatibility, Multi-Domain Inputs, and Machine Learning Support

This dataset is highly suitable for a Digital Twin (DT) framework due to its ability to seamlessly integrate with AWS IoT Core for real-time streaming and machine learning (ML) modelling. Here’s why:

- **IoT-Compatibility:**
  - Each row represents a time-stamped, sensor-driven event.
  - Can be streamed via AWS IoT Core in MQTT format, making it ideal for IoT systems.
  - Structure is tabular and time-stamped, which supports integration with both AWS IoT Core and SageMaker pipelines.
  - Can be easily streamed row-by-row as temporal IoT events.

- **Supports Machine Learning (ML) Modelling:**
  - Includes categorical (e.g., risk_classification) labels.
  - Suitable for supervised learning tasks such as classification (e.g., risk prediction)
- Multi-Domain Inputs for Real-World Simulation:

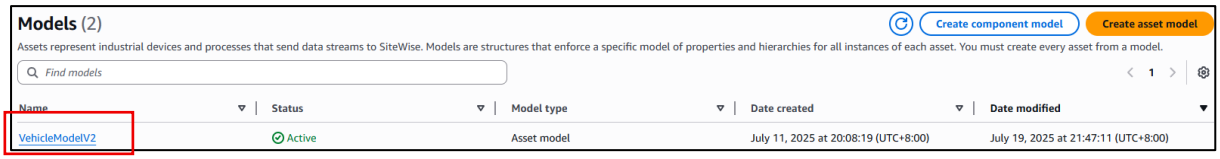
    - Environmental Factors: Includes weather, traffic conditions.
    - Driver Performance: Captures data like fatigue and behaviour.
    - Vehicle Metrics: Contains information on fuel rate, location, and other vehicle characteristics.
    - Cargo Information: Includes data on temperature and condition of cargo.
- Ideal for Building Digital Twins:
    - Supports complex real-world simulations with data from multiple domains.
    - Helps simulate and optimize logistics systems in real-time.
    - Enables AI-powered decision-making based on real-time data streaming and analysis.
- This dataset's ability to support both real-time streaming via AWS IoT Core and machine learning modelling, combined with its multi-domain, time-stamped structure, makes it not just suitable but ideal for building an AI-powered Digital Twin.

Summary

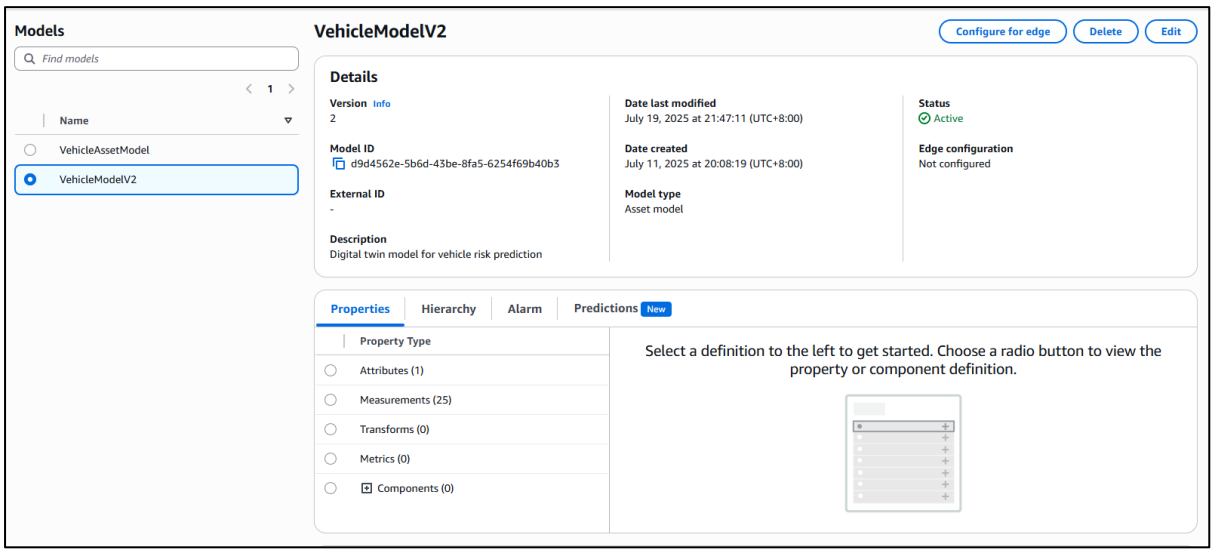
The chosen dataset enables an accurate, insightful, and deployable Digital Twin that predicts delivery delays and associated risks/costs. It provides all necessary data elements to simulate sensor signals, model outcomes using AI, and integrate with AWS cloud tools aligning perfectly with the DT objectives of monitoring, prediction, and real-time decision-making.

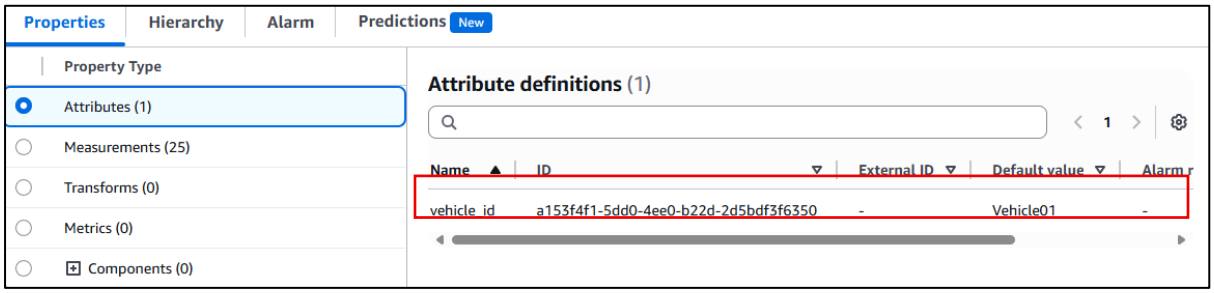
CLO1 Dataset Aggregation with AWS IOTCore

For the data connection first, we are using sitewise.



A model is created called VehicleModelV2





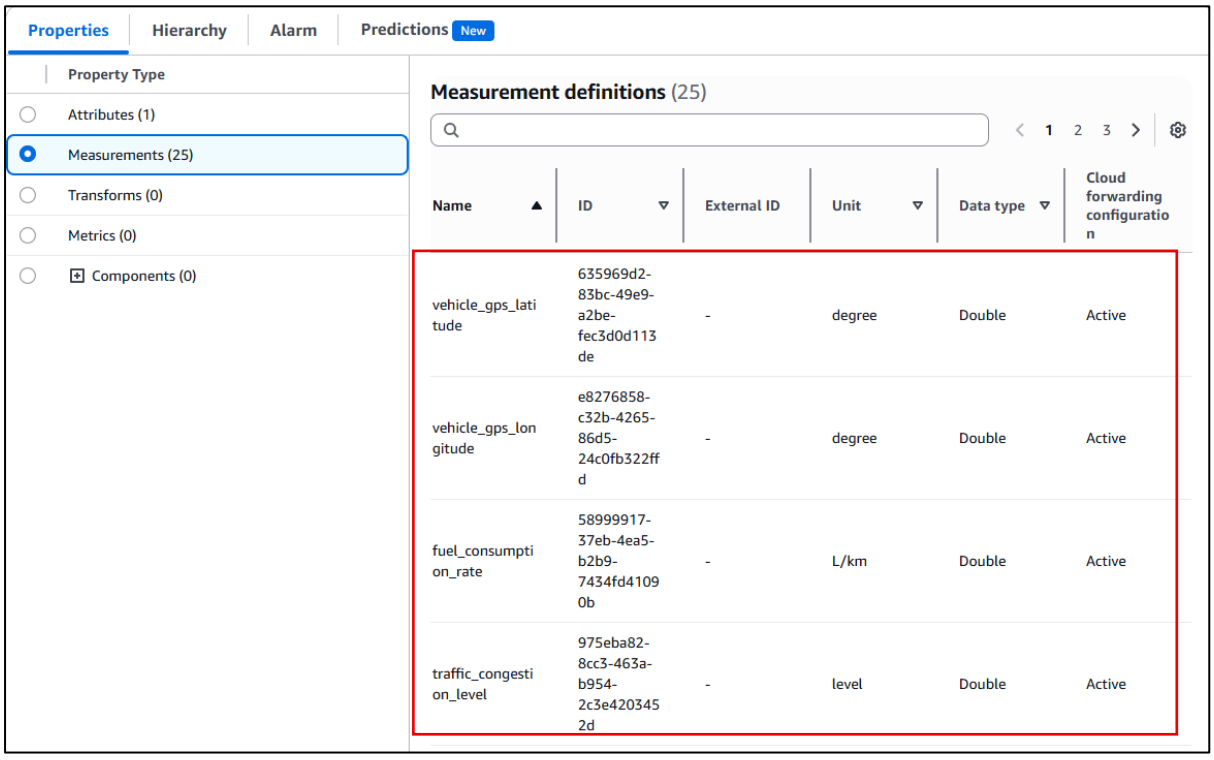
An attribute called vehicle_id is created

The vehicle_id attribute is a **unique identifier** assigned to each individual vehicle represented in the digital twin system. This attribute plays a fundamental role in organizing, tracking, and managing vehicle-specific data across various components of the system, whether it be for telemetry ingestion, historical analysis, or predictive modelling.

Purpose and Role

1. **Entity Identification:**
   a. In a logistics or IoT-based digital twin environment, multiple vehicles may be operating simultaneously.
   b. The vehicle_id allows the system to uniquely distinguish between different vehicles.
   c. For example, vehicle_id = "Vehicle01" could refer to a particular truck operating on a specific route with its own operational profile.

2. **Data Segregation:**
   a. Data such as GPS coordinates, fuel consumption, cargo condition, etc., are recorded over time for each vehicle.
   b. The vehicle_id helps associate each row of data (or SiteWise asset data point) with the correct vehicle.
   c. This ensures that predictions or analytics (like risk classification) are computed on the correct context.



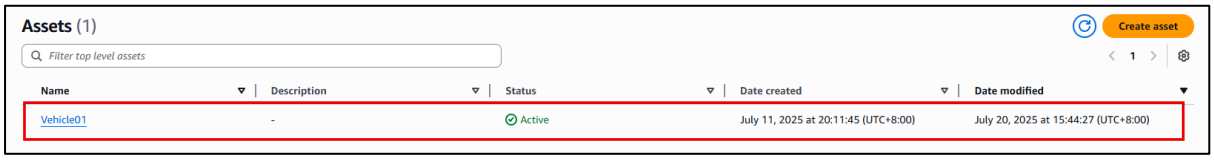
All the other measurements which are the data attributes are created including the main thing

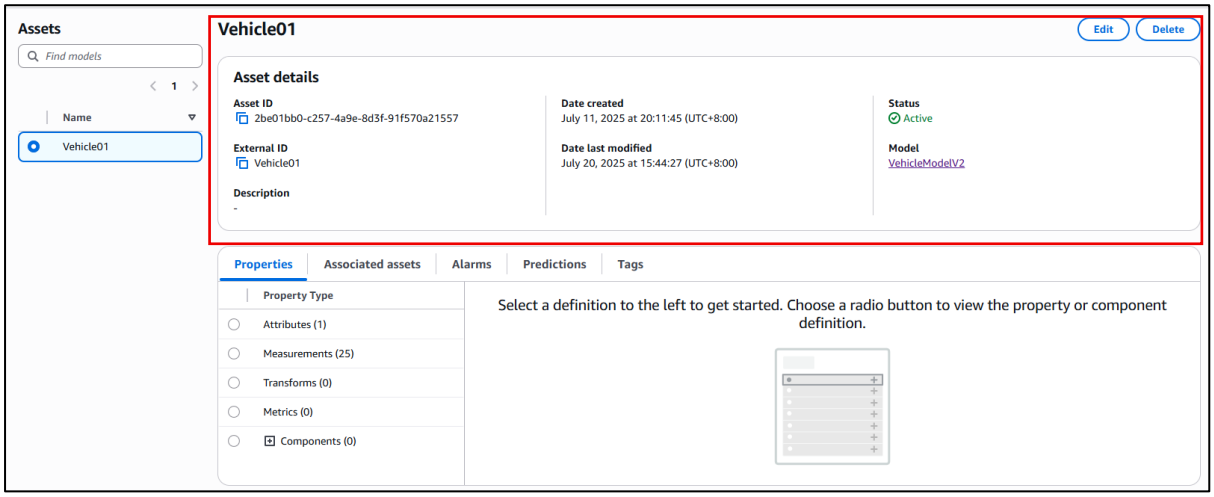
we want to get which is the risk prediction. Each **measurement** represents a specific data attribute that reflects a certain aspect of the vehicle's operation, logistics, or environment.

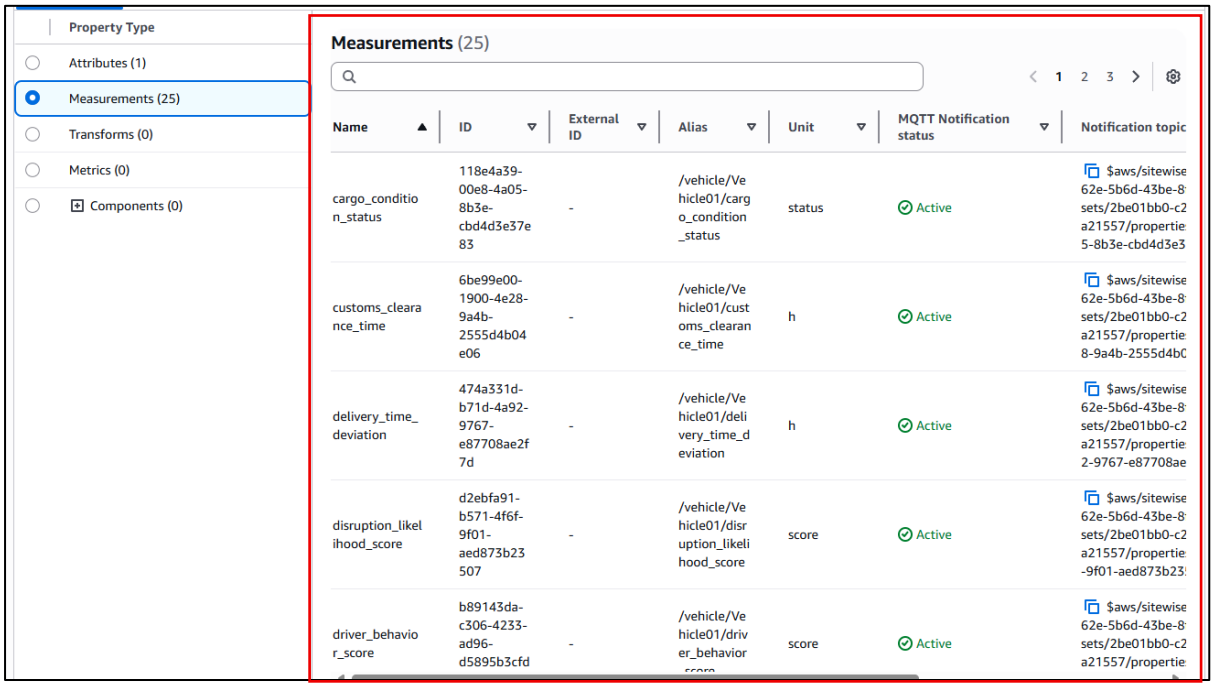
prediction	71f347f2-e4f0-4c6b-a832-3cbf87472b1d	-	-	Double	Active

Example of measurement(outcome)

Then we create our first Asset Vehicle01

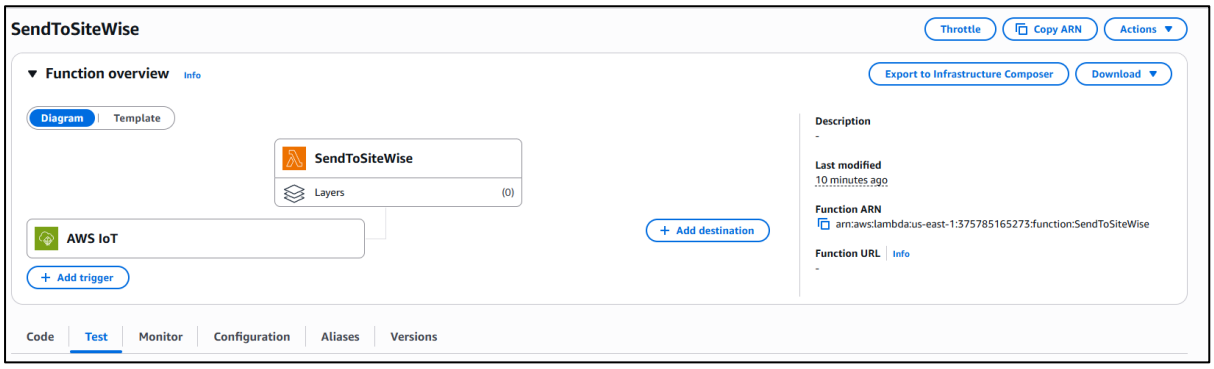






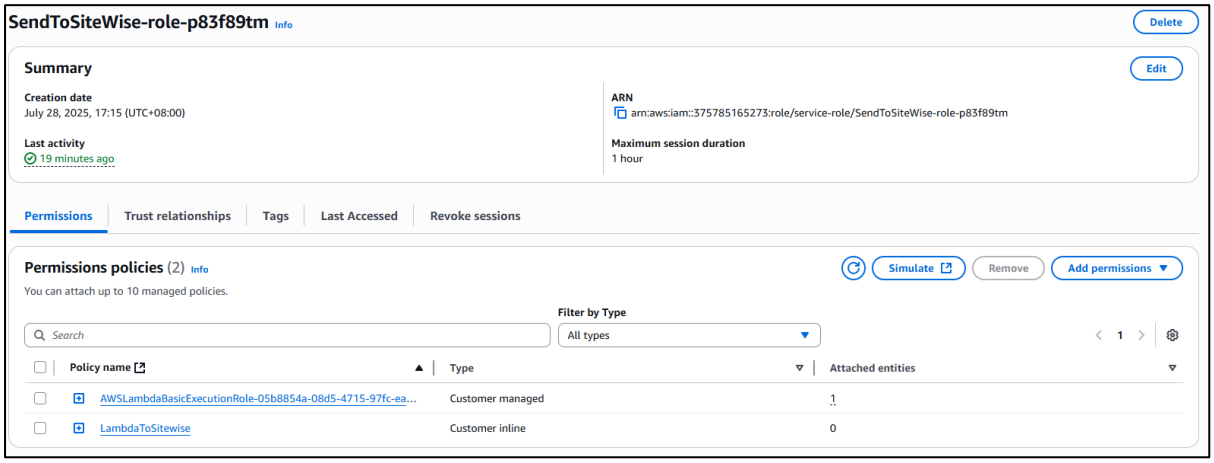
For each measurement the mqtt notification status should be turned active and for the Alias it

must follow a certain format. The alias is used as a connector for the data being sent. For example, when we send data from google collab, we must put the correct alias so that the data sent from google collab will be received by sitewise. If the alias put in is wrong, then the data will be sent to a random place.

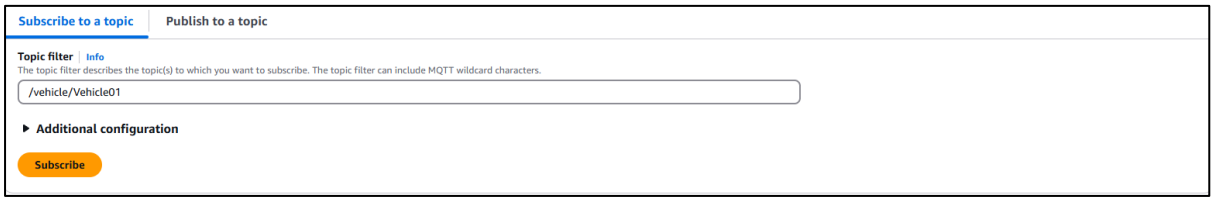


Create a lambda function.

A role at the IAM will be created. Add the policies required to do the testing. In this case we are testing the data flow from mqtt to sitewise which is triggered by lambda.



To this first test, we will send the longitude, latitude and route risk level (Not prediction risk). So first go to the mqtt test client.



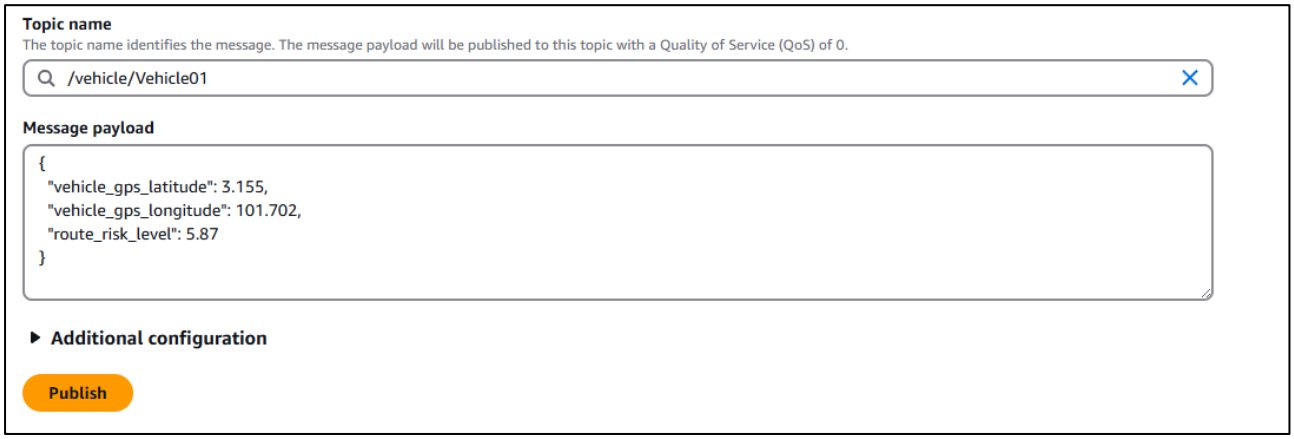
Subscribe to this following your asset created. In my case its Vehicle01.

Ok so now let's just do a simple test.

Below is the lambda code being used:

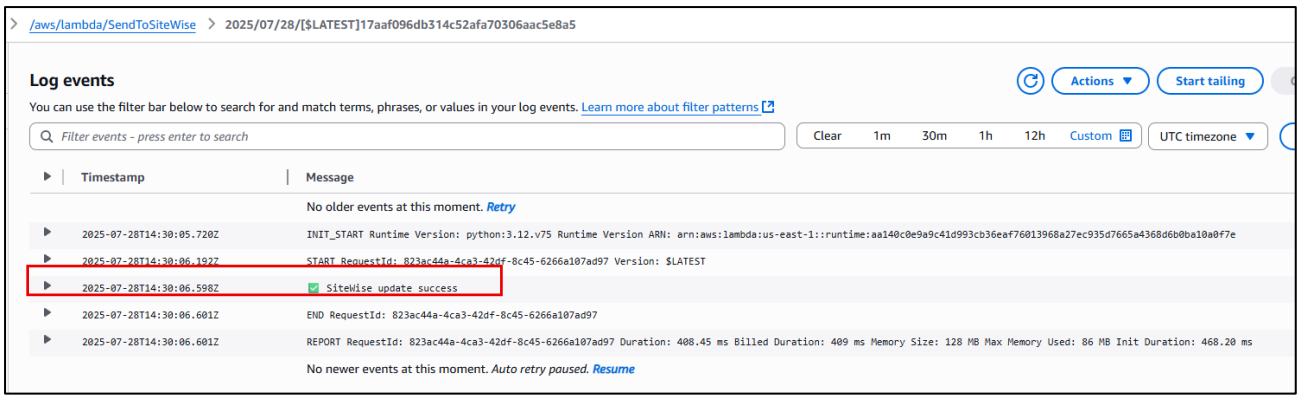
```
import boto3
import time
import json
sitewise = boto3.client('iotsitewise')

alias_map = {
   "vehicle_gps_latitude": "/vehicle/Vehicle01/vehicle_gps_latitude",
   "vehicle_gps_longitude": "/vehicle/Vehicle01/vehicle_gps_longitude",
   "route_risk_level": "/vehicle/Vehicle01/route_risk_level"
}

def lambda_handler(event, context):
   try:
      payload = event
      timestamp = int(time.time())
      entries = []

      for key, value in payload.items():
         alias = alias_map.get(key)
         if alias is None:
            continue
         clean_entry_id = f"{key}_{timestamp}".replace("/", "_")
         entries.append({
            'entryId': clean_entry_id,
            'propertyAlias': alias,
            'propertyValues': [{
               'value': {'doubleValue': float(value)},
               'timestamp': {
                  'timeInSeconds': timestamp,
                  'offsetInNanos': 0
               },
               'quality': 'GOOD'
            }]
```

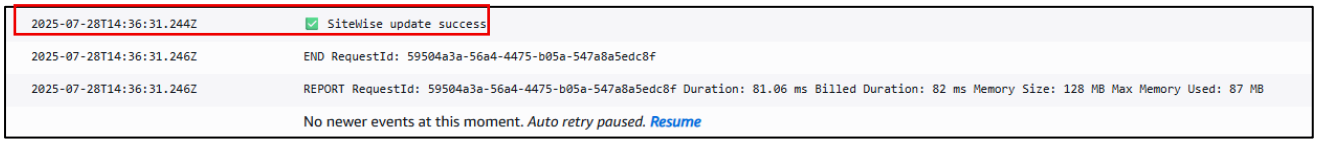
```
        })

    if entries:
        response = sitewise.batch_put_asset_property_value(entries=entries)
        print("✅ SiteWise update success")
        return { 'status': 'success', 'response': response }
    else:
        print("⚠️ No valid entries found")
        return { 'status': 'skipped', 'message': 'No valid properties matched' }

except Exception as e:
    print("❌ Lambda error:", e)
    return { 'status': 'error', 'message': str(e) }
```



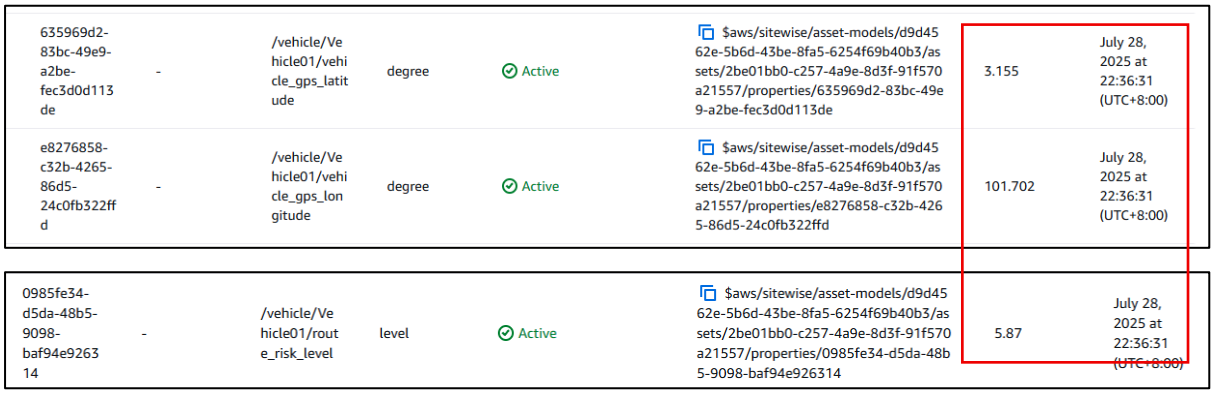
Based on the topic subscribed to, we send the test payload.

Back in the mqtt test client, this set of data is published.



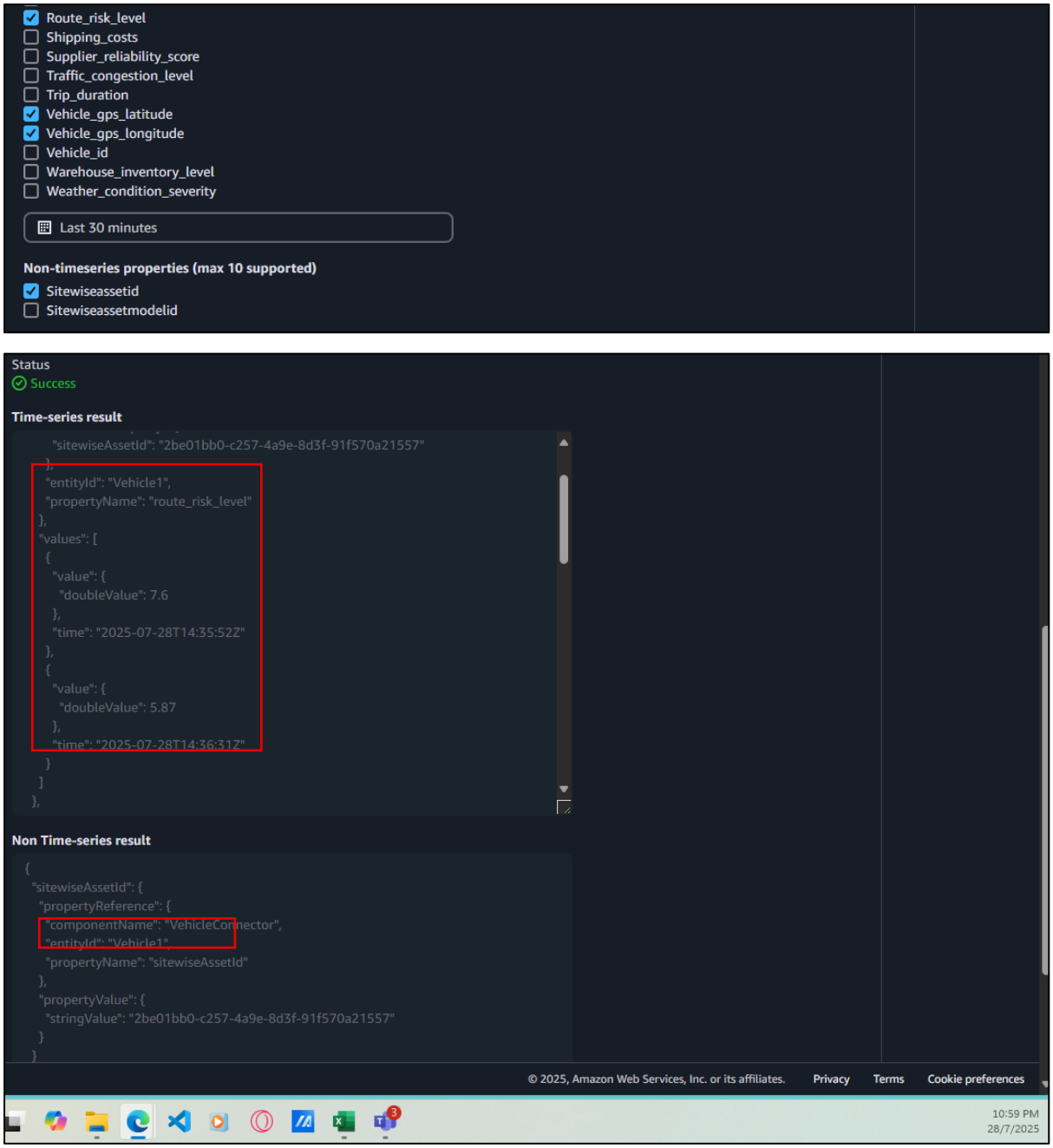


As you can see in Cloudwatch, Sitewise update is a success.



Thus, data connection is a success.

In Twinmaker, we can use the Test connector is check whether the data from sitewise is connecting to twinmaker.



You can see the time at the bottom for proof

Now we shall try streaming the data from a demo csv file created.

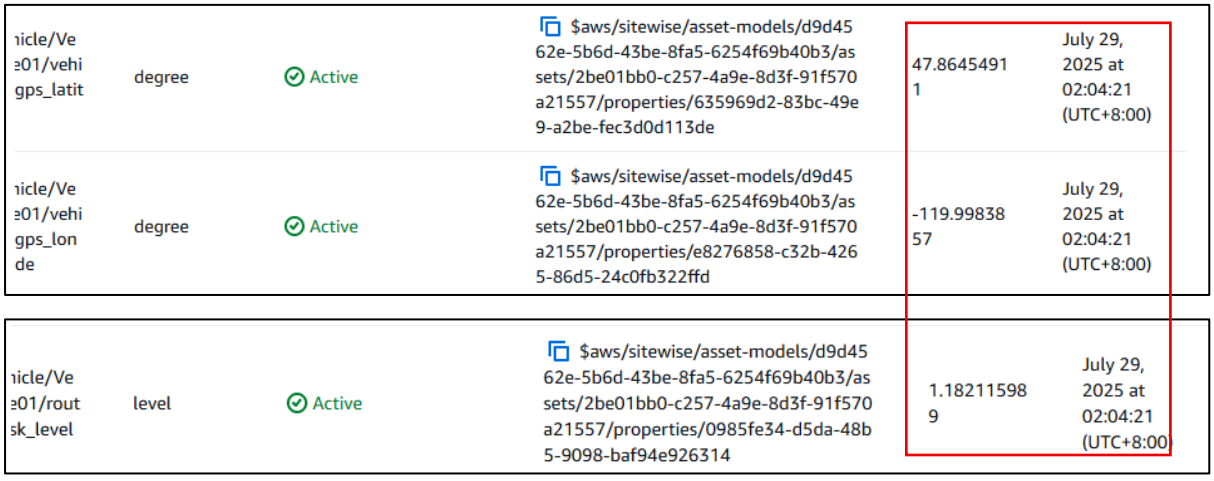
The newer Entity Vehicle01 is created with the prediction value(Vehicle1 has no prediction value)

A	B	C	D
timestamp	vehicle_gps_latitude	vehicle_gps_longitude	route_risk_level
2025-07-20 10:0(	40.37556848	-77.01431774	1.182115989
2025-07-20 10:0(	33.50781834	-117.0369024	1.182115989
2025-07-20 10:0(	30.02063979	-75.26922404	1.182115989
2025-07-20 10:0(	36.64922251	-70.1905293	1.182115989
2025-07-20 10:0(	30.00127928	-70.01219476	1.182115989
2025-07-20 10:0(	47.86454911	-119.9983857	1.182115989

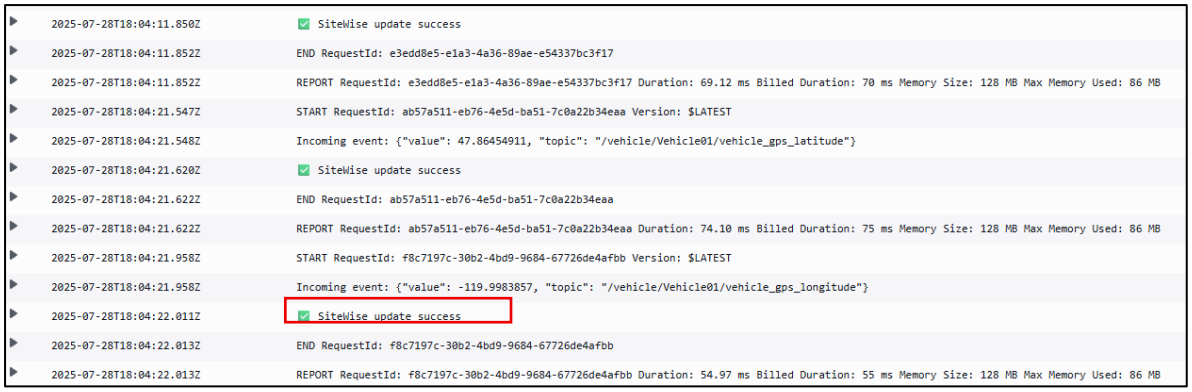
The route_risk_level is the same as it will be a fixed value when doing the prediction as mentioned in Dataset Aggregation. Latitude and Longitude will be realtime data that will be updated every 10 seconds.

```
PS C:\Users\Deads\Downloads\logistics-data-stream> python mqtt_publisher.py
C:\Users\Deads\Downloads\logistics-data-stream\mqtt_publisher.py:21: DeprecationWarning: Callback API version 1 is deprecated,
update to latest version
  client = mqtt.Client(protocol=mqtt.MQTTv311)
Connecting to MQTT broker...
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 40.37556848}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -77.01431774}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 33.50781834}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -117.0369024}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 30.02063979}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -75.26922404}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 36.64922251}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -70.1905293}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 30.00127928}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -70.01219476}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 47.86454911}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -119.9983857}
Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.182115989}
PS C:\Users\Deads\Downloads\logistics-data-stream>
```

(From the terminal)

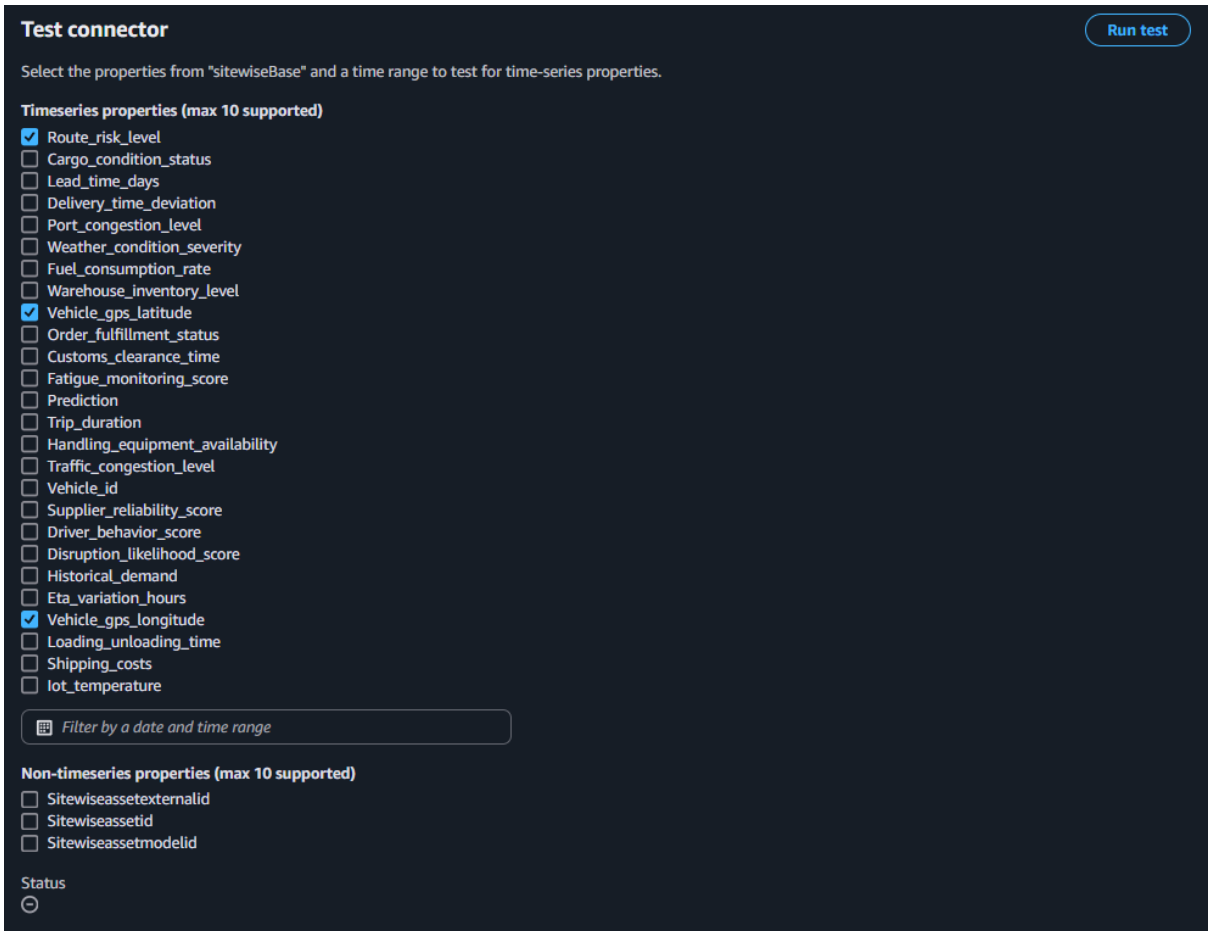


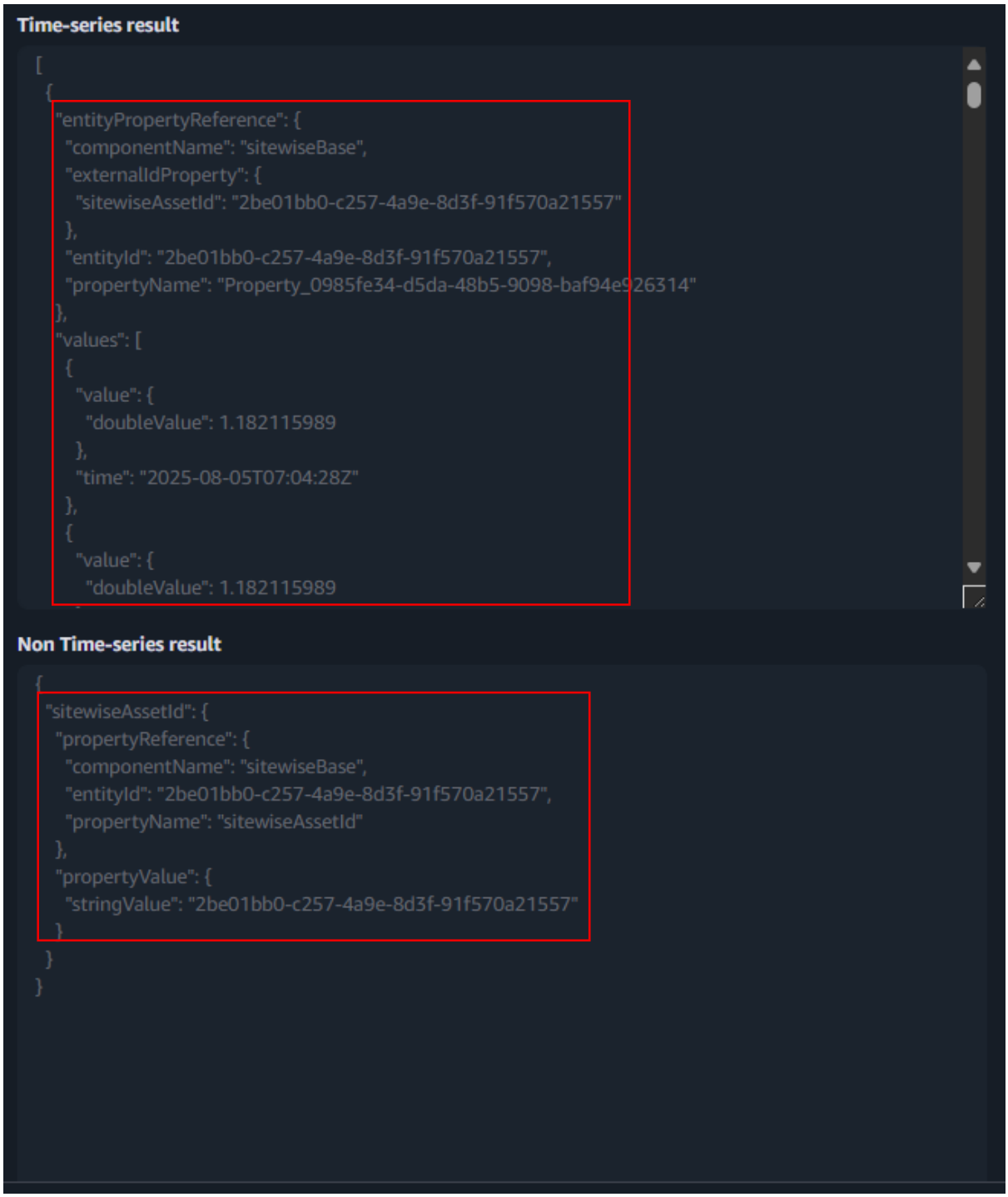
The values appearing in sitewise



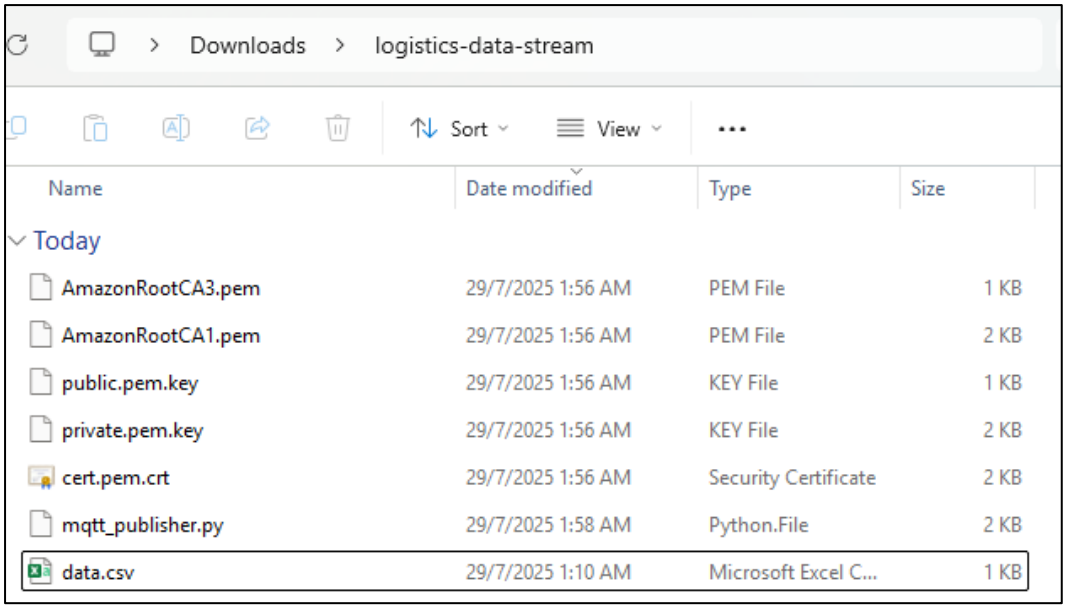
Cloudwatch logs

Another connection test





The Vehicleconnector is similar to sitewiseBase just with the added prediction measurement (success)



- **Security Credentials:** These files are required to establish a secure TLS connection between the local device and AWS IoT Core:
    - AmazonRootCA1.pem, AmazonRootCA3.pem: Certificate Authority (CA) files used to verify AWS server identity.
    - cert.pem.crt: Device certificate for authentication.
    - public.pem.key, private.pem.key: Public and private keys used in securing MQTT communications.

- **Data Stream Source:**
  data.csv: Contains structured logistics or sensor data intended for transmission. This is likely formatted with timestamps and telemetry values suitable for line-by-line streaming.

- **Transmission Script:**
  mqtt_publisher.py: A Python script responsible for reading the contents of data.csv and publishing each data entry to an MQTT topic on AWS IoT Core. It leverages the security credentials to authenticate and establish a connection.

mqtt_publisher.py

```
import csv
import time
import ssl
import json
import paho.mqtt.client as mqtt
from dotenv import load_dotenv
import os

# Load environment variables
load_dotenv()

# AWS IoT Core connection settings
MQTT_ENDPOINT = "ackvac20gj6g5-ats.iot.us-east-1.amazonaws.com"
PORT = 8883
CA_CERT = "AmazonRootCA1.pem"
DEVICE_CERT = "cert.pem.crt"
PRIVATE_KEY = "private.pem.key"
TOPIC_PREFIX = "/vehicle/Vehicle01"

# Connect MQTT client
client = mqtt.Client(protocol=mqtt.MQTTv311)
client.tls_set(ca_certs=CA_CERT,
               certfile=DEVICE_CERT,
               keyfile=PRIVATE_KEY,
               tls_version=ssl.PROTOCOL_TLSv1_2)

print("Connecting to MQTT broker...")
client.connect(MQTT_ENDPOINT, PORT, keepalive=60)
client.loop_start()

# Stream CSV data row by row every 10 seconds
with open('data.csv', 'r', encoding='utf-8') as csvfile:
```

```
    reader = csv.DictReader(csvfile)
    for row in reader:
        if not row:
            continue
        # Clean keys and values
        row = {k.strip(): v.strip() for k, v in row.items() if k and v}

        try:
            payloads = {
                f"{TOPIC_PREFIX}/vehicle_gps_latitude": float(row["vehicle_gps_latitude"]),
                f"{TOPIC_PREFIX}/vehicle_gps_longitude": float(row["vehicle_gps_longitude"]),
                f"{TOPIC_PREFIX}/route_risk_level": float(row["route_risk_level"])
            }

            for topic, value in payloads.items():
                message = json.dumps({"value": value})
                client.publish(topic, message)
                print(f"Published to {topic}: {message}")

        except KeyError as e:
            print(f" ⚠️ Missing key: {e}")
        except Exception as e:
            print(f"❌ Error: {e}")

        time.sleep(10)

client.loop_stop()
client.disconnect()
```

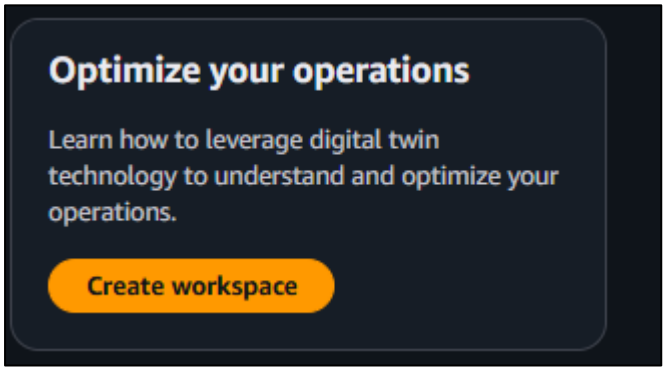
Updated Lambda function

```
import boto3
import time
import json

sitewise = boto3.client('iotsitewise')

alias_map = {
    "vehicle_gps_latitude": "/vehicle/Vehicle01/vehicle_gps_latitude",
    "vehicle_gps_longitude": "/vehicle/Vehicle01/vehicle_gps_longitude",
    "route_risk_level": "/vehicle/Vehicle01/route_risk_level"
}

def lambda_handler(event, context):
    try:
        print("Incoming event:", json.dumps(event))

        value = float(event.get("value"))

        # Get topic
        topic = event.get("topic")
        if not topic and "headers" in event:
            topic = event["headers"].get("mqtt-topic")

        if not topic:
            print("❌ No topic found")
            return { "status": "error", "reason": "No topic provided" }

        # Extract last part of topic
        metric = topic.split("/")[-1]
        alias = alias_map.get(metric)

        if alias is None:
```

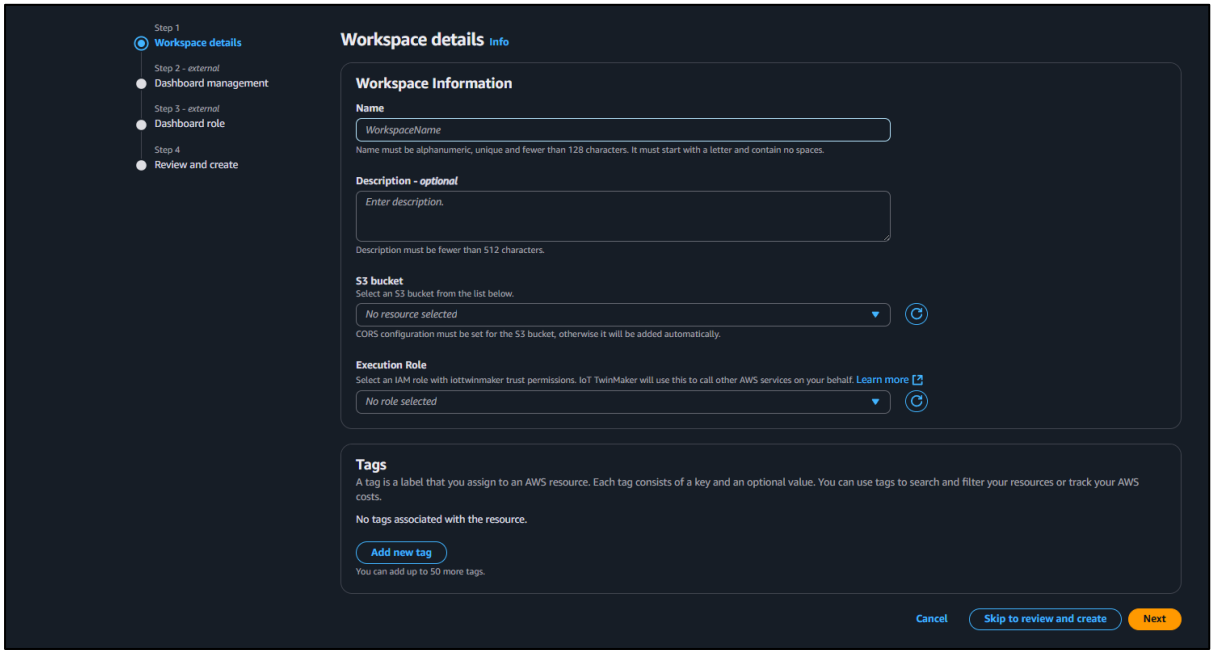
```
        print("⚠️ Unknown metric:", metric)
        return { "status": "skipped", "reason": "unknown metric" }

    timestamp = int(time.time())
    entry = {
        'entryId': f"{metric}_{timestamp}".replace("/", "_"),
        'propertyAlias': alias,
        'propertyValues': [{
            'value': {'doubleValue': value},
            'timestamp': {'timeInSeconds': timestamp},
            'quality': 'GOOD'
        }]
    }

    response = sitewise.batch_put_asset_property_value(entries=[entry])
    print("✅ SiteWise update success")
    return { 'status': 'success', 'response': response }

except Exception as e:
    print("❌ Lambda error:", e)
    return { 'status': 'error', 'message': str(e) }
```

CLO1 Dataset streaming to AWS TwinMaker

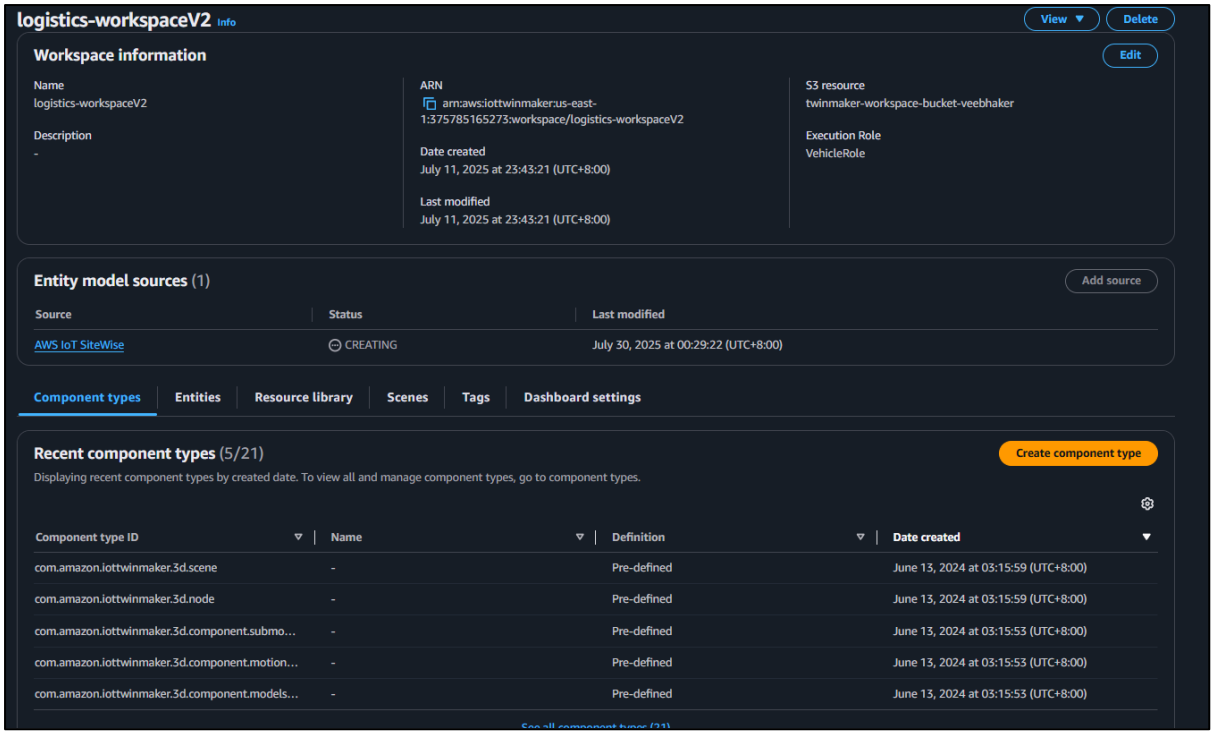
As you seen earlier the data was streamed from my laptop's terminal to mqtt which was published to sitewise through the lambda. From Sitewise the data is connector to Twinmaker.



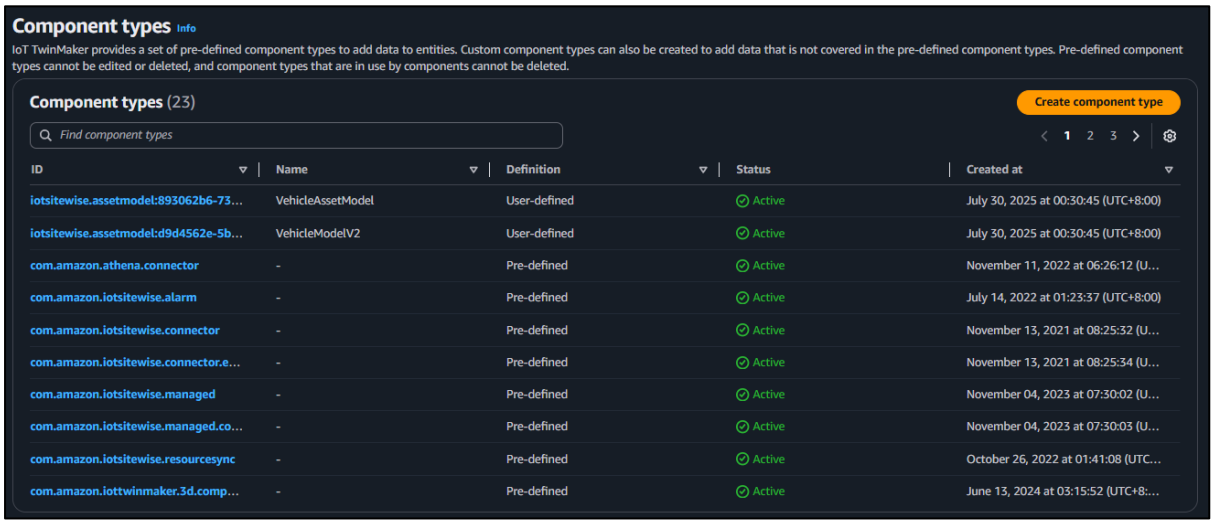
At IoT Twinmaker create your workspace



Fill in your details. An **S3 bucket** is a cloud storage container in AWS used to store files (called **objects**). It's like a folder where you upload, access, and manage data such as images, documents, or backups. Each bucket has a unique name and stores data securely with optional access controls. Select an S3 you have created.



This is the workspace our group has created.



These are the components that can be used to connect the data.

This is the main connector code used for Sitewise to Twinmaker

```
{
 "workspaceId": "logistics-workspaceV2",
 "isSingleton": false,
 "componentTypeId": "iotsitewise.assetmodel:d9d4562e-5b6d-43be-8fa5-6254f69b40b3",
 "description": "Digital twin model for vehicle risk prediction",
 "propertyDefinitions": {
```

```
  "Property_0985fe34-d5da-48b5-9098-baf94e926314": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "0985fe34-d5da-48b5-9098-baf94e926314",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "route_risk_level"
  },
  "Property_118e4a39-00e8-4a05-8b3e-cbd4d3e37e83": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "status"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
```

```
    "notificationState": "DISABLED",
    "sitewisePropertyId": "118e4a39-00e8-4a05-8b3e-cbd4d3e37e83",
    "sitewisePropertyType": "MEASUREMENT"
   },
   "displayName": "cargo_condition_status"
  },
  "Property_3328002e-f9a5-427a-8144-72c551e31496": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "days"
   },
   "isTimeSeries": true,
   "isRequiredInEntity": false,
   "isExternalId": false,
   "isStoredExternally": true,
   "isImported": false,
   "isFinal": false,
   "isInherited": false,
   "configuration": {
    "forwardingConfigState": "ENABLED",
    "notificationState": "DISABLED",
    "sitewisePropertyId": "3328002e-f9a5-427a-8144-72c551e31496",
    "sitewisePropertyType": "MEASUREMENT"
   },
   "displayName": "lead_time_days"
  },
  "Property_474a331d-b71d-4a92-9767-e87708ae2f7d": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "h"
   },
   "isTimeSeries": true,
   "isRequiredInEntity": false,
   "isExternalId": false,
```

```
  "isStoredExternally": true,
  "isImported": false,
  "isFinal": false,
  "isInherited": false,
  "configuration": {
   "forwardingConfigState": "ENABLED",
   "notificationState": "DISABLED",
   "sitewisePropertyId": "474a331d-b71d-4a92-9767-e87708ae2f7d",
   "sitewisePropertyType": "MEASUREMENT"
  },
  "displayName": "delivery_time_deviation"
 },
 "Property_47eb038a-8037-4f31-9fe4-3e881f6da24e": {
  "dataType": {
   "type": "DOUBLE",
   "unitOfMeasure": "level"
  },
  "isTimeSeries": true,
  "isRequiredInEntity": false,
  "isExternalId": false,
  "isStoredExternally": true,
  "isImported": false,
  "isFinal": false,
  "isInherited": false,
  "configuration": {
   "forwardingConfigState": "ENABLED",
   "notificationState": "DISABLED",
   "sitewisePropertyId": "47eb038a-8037-4f31-9fe4-3e881f6da24e",
   "sitewisePropertyType": "MEASUREMENT"
  },
  "displayName": "port_congestion_level"
 },
 "Property_4fba331a-b3eb-492a-acbc-6f2c4d4b7b91": {
  "dataType": {
```

```
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "4fba331a-b3eb-492a-acbc-6f2c4d4b7b91",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "weather_condition_severity"
  },
  "Property_58999917-37eb-4ea5-b2b9-7434fd41090b": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "L/km"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "58999917-37eb-4ea5-b2b9-7434fd41090b",
```

```
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "fuel_consumption_rate"
  },
  "Property_603edb8e-0550-4d79-ac67-84568908b2fd": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "603edb8e-0550-4d79-ac67-84568908b2fd",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "warehouse_inventory_level"
  },
  "Property_635969d2-83bc-49e9-a2be-fec3d0d113de": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "degree"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
```

```
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "635969d2-83bc-49e9-a2be-fec3d0d113de",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "vehicle_gps_latitude"
  },
  "Property_69609821-498d-4e7c-b23c-df942062347f": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "status"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "69609821-498d-4e7c-b23c-df942062347f",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "order_fulfillment_status"
  },
  "Property_6be99e00-1900-4e28-9a4b-2555d4b04e06": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
```

```
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "6be99e00-1900-4e28-9a4b-2555d4b04e06",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "customs_clearance_time"
  },
  "Property_6ce1d9eb-584d-4ee1-baf4-625178d92001": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "6ce1d9eb-584d-4ee1-baf4-625178d92001",
      "sitewisePropertyType": "MEASUREMENT"
    },
```

```
    "displayName": "fatigue_monitoring_score"
  },
  "Property_71f347f2-e4f0-4c6b-a832-3cbf87472b1d": {
    "dataType": {
      "type": "DOUBLE"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "71f347f2-e4f0-4c6b-a832-3cbf87472b1d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "prediction"
  },
  "Property_756965ed-cae2-444b-943c-c75e8ad584fc": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "s"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
```

```
    "forwardingConfigState": "ENABLED",
    "notificationState": "DISABLED",
    "sitewisePropertyId": "756965ed-cae2-444b-943c-c75e8ad584fc",
    "sitewisePropertyType": "MEASUREMENT"
   },
   "displayName": "trip_duration"
  },
  "Property_7599ede6-8076-4120-9fcd-e76cff5c6dc9": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "level"
   },
   "isTimeSeries": true,
   "isRequiredInEntity": false,
   "isExternalId": false,
   "isStoredExternally": true,
   "isImported": false,
   "isFinal": false,
   "isInherited": false,
   "configuration": {
    "forwardingConfigState": "ENABLED",
    "notificationState": "DISABLED",
    "sitewisePropertyId": "7599ede6-8076-4120-9fcd-e76cff5c6dc9",
    "sitewisePropertyType": "MEASUREMENT"
   },
   "displayName": "handling_equipment_availability"
  },
  "Property_975eba82-8cc3-463a-b954-2c3e4203452d": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "level"
   },
   "isTimeSeries": true,
   "isRequiredInEntity": false,
```

```
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "975eba82-8cc3-463a-b954-2c3e4203452d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "traffic_congestion_level"
  },
  "Property_a153f4f1-5dd0-4ee0-b22d-2d5bdf3f6350": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "notificationState": "DISABLED",
      "sitewiseDefaultValue": "Vehicle01",
      "sitewisePropertyId": "a153f4f1-5dd0-4ee0-b22d-2d5bdf3f6350",
      "sitewisePropertyType": "ATTRIBUTE"
    },
    "displayName": "vehicle_id"
  },
  "Property_b4d1cca0-7987-425c-ba13-36c05a5c5ff8": {
    "dataType": {
```

```
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "b4d1cca0-7987-425c-ba13-36c05a5c5ff8",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "supplier_reliability_score"
  },
  "Property_b89143da-c306-4233-ad96-d5895b3cfd2d": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "b89143da-c306-4233-ad96-d5895b3cfd2d",
```

```
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "driver_behavior_score"
  },
  "Property_d2ebfa91-b571-4f6f-9f01-aed873b23507": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "d2ebfa91-b571-4f6f-9f01-aed873b23507",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "disruption_likelihood_score"
  },
  "Property_d4bdbf7d-3d3f-4e8b-b1b9-2340f28f117e": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "units"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
```

```
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "d4bdbf7d-3d3f-4e8b-b1b9-2340f28f117e",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "historical_demand"
  },
  "Property_dd96b4ee-b879-405c-a45d-f3b512253c14": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "dd96b4ee-b879-405c-a45d-f3b512253c14",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "eta_variation_hours"
  },
  "Property_e8276858-c32b-4265-86d5-24c0fb322ffd": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "degree"
```

```
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "e8276858-c32b-4265-86d5-24c0fb322ffd",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "vehicle_gps_longitude"
  },
  "Property_e9594b5c-5f83-4b69-b080-ea1c01fd71a4": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "e9594b5c-5f83-4b69-b080-ea1c01fd71a4",
      "sitewisePropertyType": "MEASUREMENT"
    },
```

```
    "displayName": "loading_unloading_time"
  },
  "Property_ebeb1d51-57ab-4377-8964-b40b49a15229": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "USD"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "ebeb1d51-57ab-4377-8964-b40b49a15229",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "shipping_costs"
  },
  "Property_fdab7aff-2b59-4321-8922-94b571f67ff2": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "°C"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
```

```
    "configuration": {
      "forwardingConfigState": "ENABLED",
      "notificationState": "DISABLED",
      "sitewisePropertyId": "fdab7aff-2b59-4321-8922-94b571f67ff2",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "iot_temperature"
  },
  "sitewiseAssetId": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": false,
    "isRequiredInEntity": true,
    "isExternalId": true,
    "isStoredExternally": false,
    "isImported": false,
    "isFinal": false,
    "isInherited": true
  },
  "sitewiseAssetModelId": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": false,
    "isRequiredInEntity": true,
    "isExternalId": false,
    "isStoredExternally": false,
    "isImported": false,
    "isFinal": false,
    "isInherited": true,
    "defaultValue": {
      "stringValue": "d9d4562e-5b6d-43be-8fa5-6254f69b40b3"
    }
```

```
    }
  },
  "extendsFrom": [
    "com.amazon.iotsitewise.resourcesync"
  ],
  "functions": {
    "dataReaderByEntity": {
      "implementedBy": {
        "lambda": {},
        "isNative": true
      },
      "isInherited": true
    },
    "dataWriter": {
      "implementedBy": {
        "lambda": {},
        "isNative": true
      },
      "isInherited": true
    }
  },
  "creationDateTime": "2025-07-29T16:30:45.629Z",
  "updateDateTime": "2025-07-29T16:30:45.629Z",
  "arn": "arn:aws:iottwinmaker:us-east-1:375785165273:workspace/logistics-
workspaceV2/component-type/iotsitewise.assetmodel:d9d4562e-5b6d-43be-8fa5-
6254f69b40b3",
  "isAbstract": false,
  "isSchemaInitialized": false,
  "status": {
    "state": "ACTIVE",
    "error": {}
  },
  "syncSource": "SITEWISE",
  "componentTypeName": "VehicleModelV2"
```

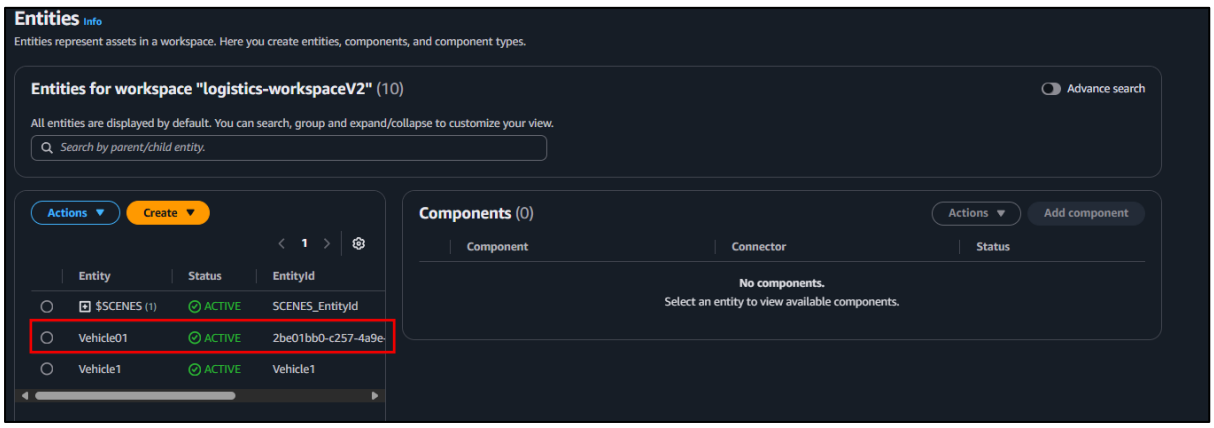
```
}
```

This JSON code represents a TwinMaker component that connects properties from AWS IoT SiteWise to a digital twin in AWS IoT TwinMaker. Instead of defining individual components manually, this version uses a componentTypeId to reference a reusable component type definition, in this case, "com.vehicle.component". This allows consistent schema enforcement and simplifies deployment across multiple entities or vehicles.

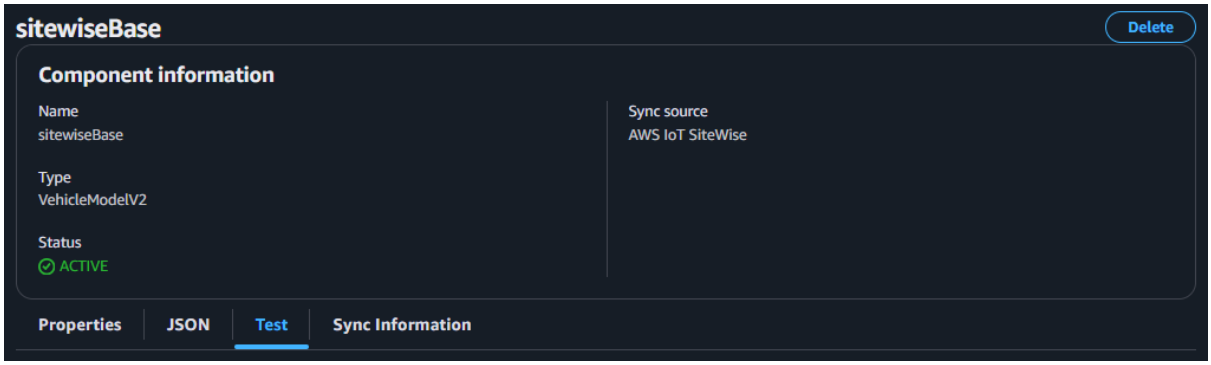
Each property within this component—such as cargo_condition_status, prediction, vehicle_id, vehicle_gps_latitude, and others—is linked directly to a SiteWise property using the sitewisePropertyId field under the configuration section. This creates a mapping between the digital twin and the physical data in SiteWise.

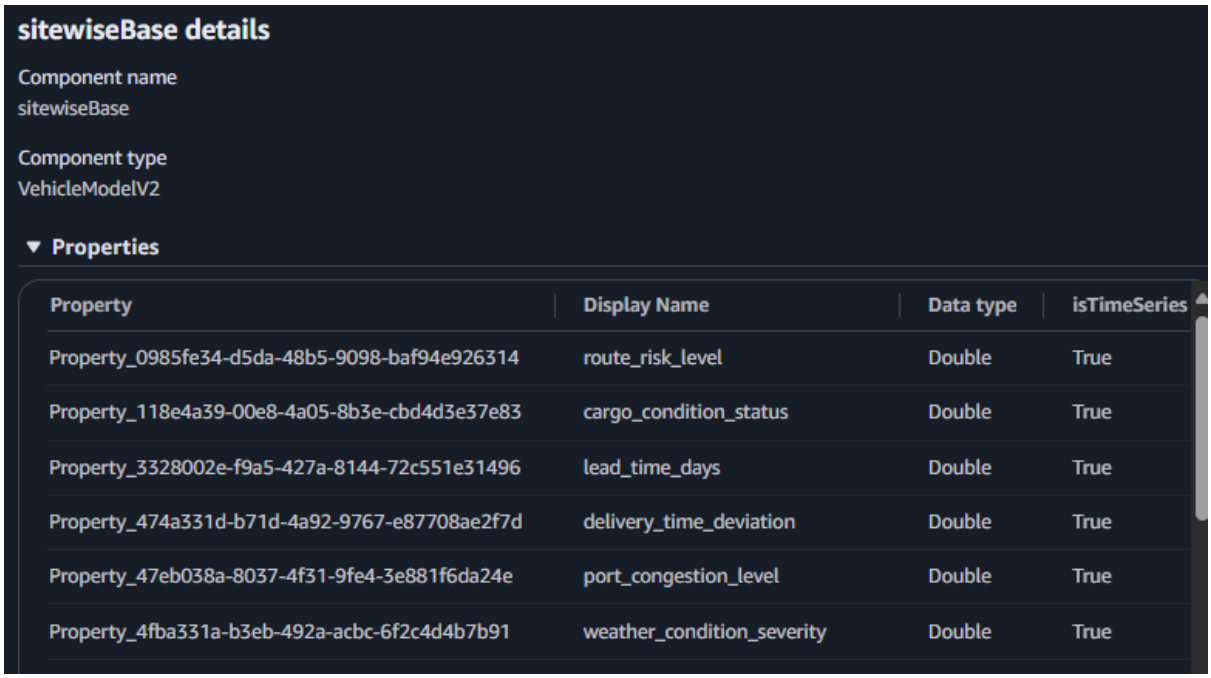
All these properties are defined with metadata fields like dataType, isTimeSeries, and isStoredExternally to specify how they are handled in TwinMaker. The value isTimeSeries: true indicates that the data will update over time, and isStoredExternally: true and isImported: true confirm that the data source is external specifically AWS IoT SiteWise.

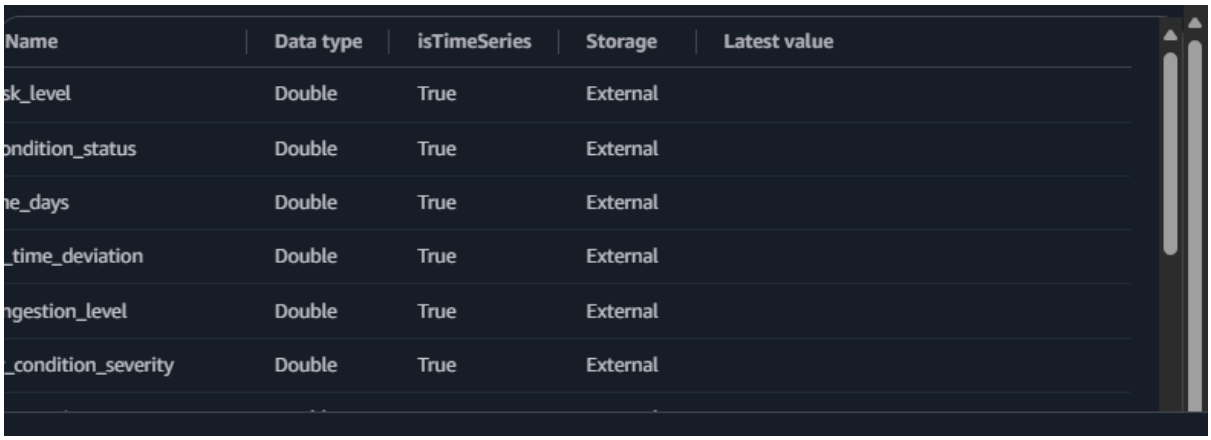
Additionally, the propertyGroups section organizes all the properties under a group called "Default", which is useful for UI presentation and querying in dashboards like Grafana. This helps structure the digital twin entity for better visualization and navigation.



This is the main Entity created in the Workspace.







The "Latest value" for time-series attributes is blank because the data is stored externally. This typically means the system you're viewing doesn't directly store or constantly fetch the most recent data point from the external source due to performance, data volume, or specific display configurations. You'd likely need to use a dedicated tool or feature to retrieve and visualize this time-series data.

Below is the JSON Configuration for the Vehicle 01 Component sitewiseBase


```
{
 "Property_0985fe34-d5da-48b5-9098-baf94e926314": {
  "definition": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "level"
   },
   "isTimeSeries": true,
   "isRequiredInEntity": false,
   "isExternalId": false,
   "isStoredExternally": true,
   "isImported": false,
   "isFinal": false,
   "isInherited": false,
   "configuration": {
    "alias": "/vehicle/Vehicle01/route_risk_level",
    "forwardingConfigState": "ENABLED",
    "notificationState": "ENABLED",
    "sitewisePropertyId": "0985fe34-d5da-48b5-9098-baf94e926314",
    "sitewisePropertyType": "MEASUREMENT"
   },
   "displayName": "route_risk_level"
  }
 },
 "Property_118e4a39-00e8-4a05-8b3e-cbd4d3e37e83": {
  "definition": {
   "dataType": {
    "type": "DOUBLE",
    "unitOfMeasure": "status"
   },
   "isTimeSeries": true,
```

```
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/cargo_condition_status",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "118e4a39-00e8-4a05-8b3e-cbd4d3e37e83",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "cargo_condition_status"
  }
},
"Property_3328002e-f9a5-427a-8144-72c551e31496": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "days"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/lead_time_days",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "3328002e-f9a5-427a-8144-72c551e31496",
```

```
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "lead_time_days"
  }
},
"Property_474a331d-b71d-4a92-9767-e87708ae2f7d": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/delivery_time_deviation",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "474a331d-b71d-4a92-9767-e87708ae2f7d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "delivery_time_deviation"
  }
},
"Property_47eb038a-8037-4f31-9fe4-3e881f6da24e": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
```

```
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/port_congestion_level",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "47eb038a-8037-4f31-9fe4-3e881f6da24e",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "port_congestion_level"
  }
},
"Property_4fba331a-b3eb-492a-acbc-6f2c4d4b7b91": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/weather_condition_severity",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
```

```
      "sitewisePropertyId": "4fba331a-b3eb-492a-acbc-6f2c4d4b7b91",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "weather_condition_severity"
  }
},
"Property_58999917-37eb-4ea5-b2b9-7434fd41090b": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "L/km"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/fuel_consumption_rate",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "58999917-37eb-4ea5-b2b9-7434fd41090b",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "fuel_consumption_rate"
  }
},
"Property_603edb8e-0550-4d79-ac67-84568908b2fd": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
```

```
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/warehouse_inventory_level",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "603edb8e-0550-4d79-ac67-84568908b2fd",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "warehouse_inventory_level"
  }
},
"Property_635969d2-83bc-49e9-a2be-fec3d0d113de": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "degree"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/vehicle_gps_latitude",
      "forwardingConfigState": "ENABLED",
```

```
      "notificationState": "ENABLED",
      "sitewisePropertyId": "635969d2-83bc-49e9-a2be-fec3d0d113de",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "vehicle_gps_latitude"
  }
},
"Property_69609821-498d-4e7c-b23c-df942062347f": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "status"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/order_fulfillment_status",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "69609821-498d-4e7c-b23c-df942062347f",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "order_fulfillment_status"
  }
},
"Property_6be99e00-1900-4e28-9a4b-2555d4b04e06": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
```

```
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/customs_clearance_time",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "6be99e00-1900-4e28-9a4b-2555d4b04e06",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "customs_clearance_time"
  }
},
"Property_6ce1d9eb-584d-4ee1-baf4-625178d92001": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/fatigue_monitoring_score",
```

```
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "6ce1d9eb-584d-4ee1-baf4-625178d92001",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "fatigue_monitoring_score"
  }
},
"Property_71f347f2-e4f0-4c6b-a832-3cbf87472b1d": {
  "definition": {
    "dataType": {
      "type": "DOUBLE"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/prediction",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "71f347f2-e4f0-4c6b-a832-3cbf87472b1d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "prediction"
  }
},
"Property_756965ed-cae2-444b-943c-c75e8ad584fc": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
```

```
      "unitOfMeasure": "s"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/trip_duration",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "756965ed-cae2-444b-943c-c75e8ad584fc",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "trip_duration"
  }
},
"Property_7599ede6-8076-4120-9fcd-e76cff5c6dc9": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/handling_equipment_availability",
```

```
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "7599ede6-8076-4120-9fcd-e76cff5c6dc9",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "handling_equipment_availability"
  }
},
"Property_975eba82-8cc3-463a-b954-2c3e4203452d": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "level"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/traffic_congestion_level",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "975eba82-8cc3-463a-b954-2c3e4203452d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "traffic_congestion_level"
  }
},
"Property_a153f4f1-5dd0-4ee0-b22d-2d5bdf3f6350": {
  "definition": {
    "dataType": {
```

```
      "type": "STRING"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/vehicle_id",
      "notificationState": "DISABLED",
      "sitewiseDefaultValue": "Vehicle01",
      "sitewisePropertyId": "a153f4f1-5dd0-4ee0-b22d-2d5bdf3f6350",
      "sitewisePropertyType": "ATTRIBUTE"
    },
    "displayName": "vehicle_id"
  }
},
"Property_b4d1cca0-7987-425c-ba13-36c05a5c5ff8": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/supplier_reliability_score",
```

```
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "b4d1cca0-7987-425c-ba13-36c05a5c5ff8",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "supplier_reliability_score"
  }
},
"Property_b89143da-c306-4233-ad96-d5895b3cfd2d": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/driver_behavior_score",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "b89143da-c306-4233-ad96-d5895b3cfd2d",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "driver_behavior_score"
  }
},
"Property_d2ebfa91-b571-4f6f-9f01-aed873b23507": {
  "definition": {
    "dataType": {
```

```
      "type": "DOUBLE",
      "unitOfMeasure": "score"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/disruption_likelihood_score",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "d2ebfa91-b571-4f6f-9f01-aed873b23507",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "disruption_likelihood_score"
  }
},
"Property_d4bdbf7d-3d3f-4e8b-b1b9-2340f28f117e": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "units"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
```

```
      "alias": "/vehicle/Vehicle01/historical_demand",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "d4bdbf7d-3d3f-4e8b-b1b9-2340f28f117e",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "historical_demand"
  }
},
"Property_dd96b4ee-b879-405c-a45d-f3b512253c14": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/eta_variation_hours",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "dd96b4ee-b879-405c-a45d-f3b512253c14",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "eta_variation_hours"
  }
},
"Property_e8276858-c32b-4265-86d5-24c0fb322ffd": {
  "definition": {
```

```
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "degree"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/vehicle_gps_longitude",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "e8276858-c32b-4265-86d5-24c0fb322ffd",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "vehicle_gps_longitude"
  }
},
"Property_e9594b5c-5f83-4b69-b080-ea1c01fd71a4": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "h"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
```

```
    "configuration": {
      "alias": "/vehicle/Vehicle01/loading_unloading_time",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "e9594b5c-5f83-4b69-b080-ea1c01fd71a4",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "loading_unloading_time"
  }
},
"Property_ebeb1d51-57ab-4377-8964-b40b49a15229": {
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "USD"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/shipping_costs",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "ebeb1d51-57ab-4377-8964-b40b49a15229",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "shipping_costs"
  }
},
"Property_fdab7aff-2b59-4321-8922-94b571f67ff2": {
```

```
  "definition": {
    "dataType": {
      "type": "DOUBLE",
      "unitOfMeasure": "°C"
    },
    "isTimeSeries": true,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": true,
    "isImported": false,
    "isFinal": false,
    "isInherited": false,
    "configuration": {
      "alias": "/vehicle/Vehicle01/iot_temperature",
      "forwardingConfigState": "ENABLED",
      "notificationState": "ENABLED",
      "sitewisePropertyId": "fdab7aff-2b59-4321-8922-94b571f67ff2",
      "sitewisePropertyType": "MEASUREMENT"
    },
    "displayName": "iot_temperature"
  }
},
"sitewiseAssetExternalId": {
  "definition": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": false,
    "isRequiredInEntity": false,
    "isExternalId": false,
    "isStoredExternally": false,
    "isImported": false,
    "isFinal": false,
    "isInherited": false
```

```
  },
  "value": {
    "stringValue": "Vehicle01"
  }
},
"sitewiseAssetId": {
  "definition": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": false,
    "isRequiredInEntity": true,
    "isExternalId": true,
    "isStoredExternally": false,
    "isImported": false,
    "isFinal": false,
    "isInherited": true
  },
  "value": {
    "stringValue": "2be01bb0-c257-4a9e-8d3f-91f570a21557"
  }
},
"sitewiseAssetModelId": {
  "definition": {
    "dataType": {
      "type": "STRING"
    },
    "isTimeSeries": false,
    "isRequiredInEntity": true,
    "isExternalId": false,
    "isStoredExternally": false,
    "isImported": false,
    "isFinal": false,
    "isInherited": true,
```

```
    "defaultValue": {
      "stringValue": "d9d4562e-5b6d-43be-8fa5-6254f69b40b3"
    }
   },
   "value": {
    "stringValue": "d9d4562e-5b6d-43be-8fa5-6254f69b40b3"
   }
  }
}
```

Each property includes a definition block that describes its characteristics:

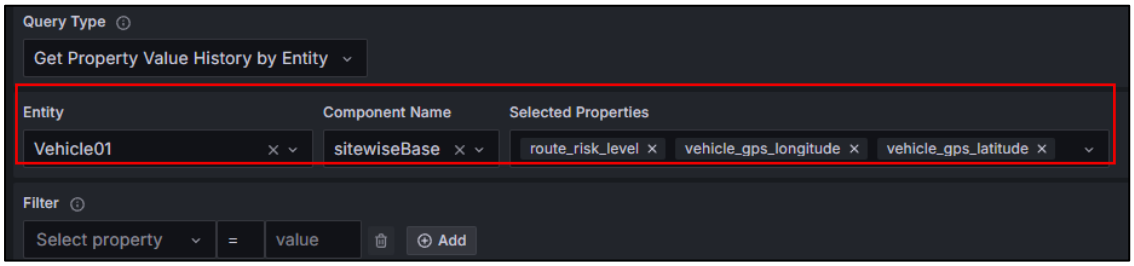
- The dataType indicates whether the property holds a numeric (DOUBLE) or text (STRING) value.
- isTimeSeries specifies whether the property is expected to change over time and be recorded as a time series.
- isStoredExternally indicates the data is managed outside of TwinMaker, typically in AWS IoT SiteWise.
- isImported confirms the data is pulled from an external source like SiteWise.
- isRequiredInEntity and isExternalId determine whether the property is required for the entity or used to uniquely identify it.
- configuration contains a sitewisePropertyId, which links the property to its corresponding asset property in SiteWise.

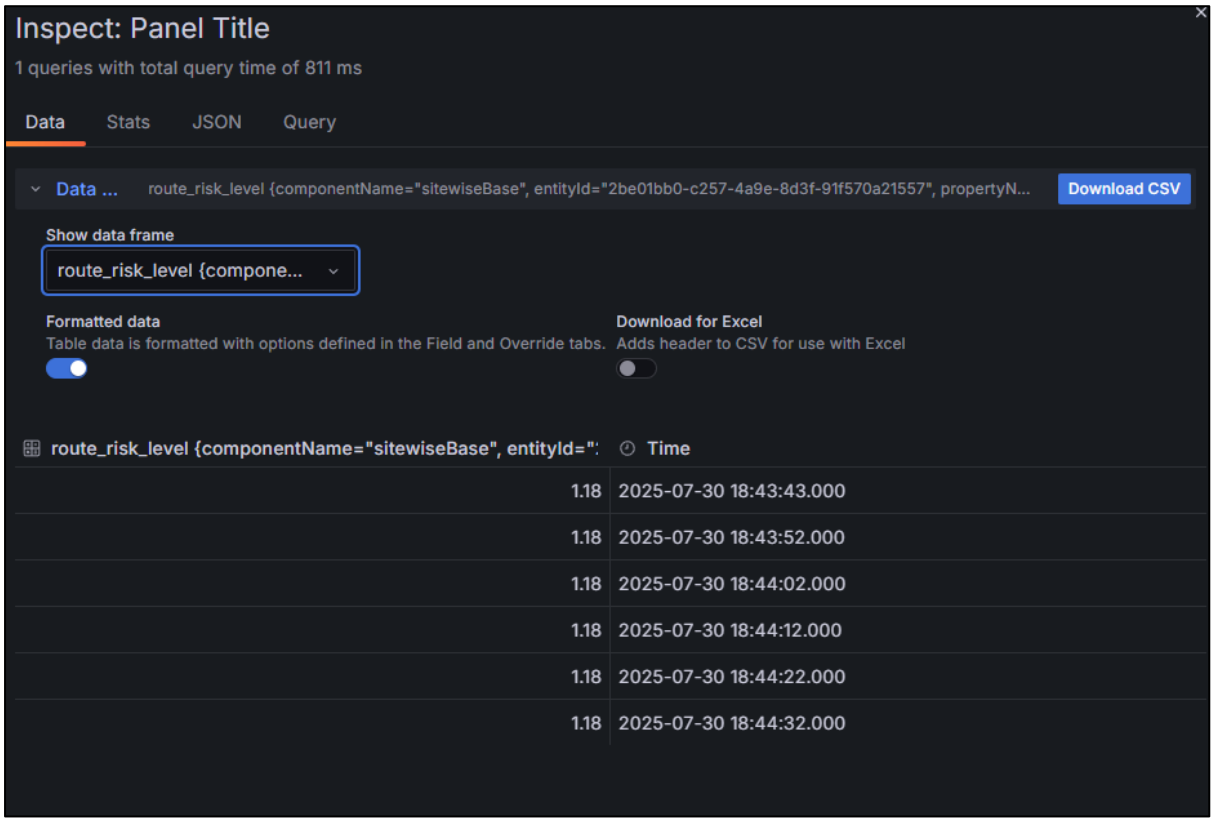
For example, cargo_condition_status, fuel_consumption_rate, and prediction are numeric time-series values imported from SiteWise using specific property IDs. These data streams allow the digital twin to reflect the real-time and historical status of the physical vehicle.

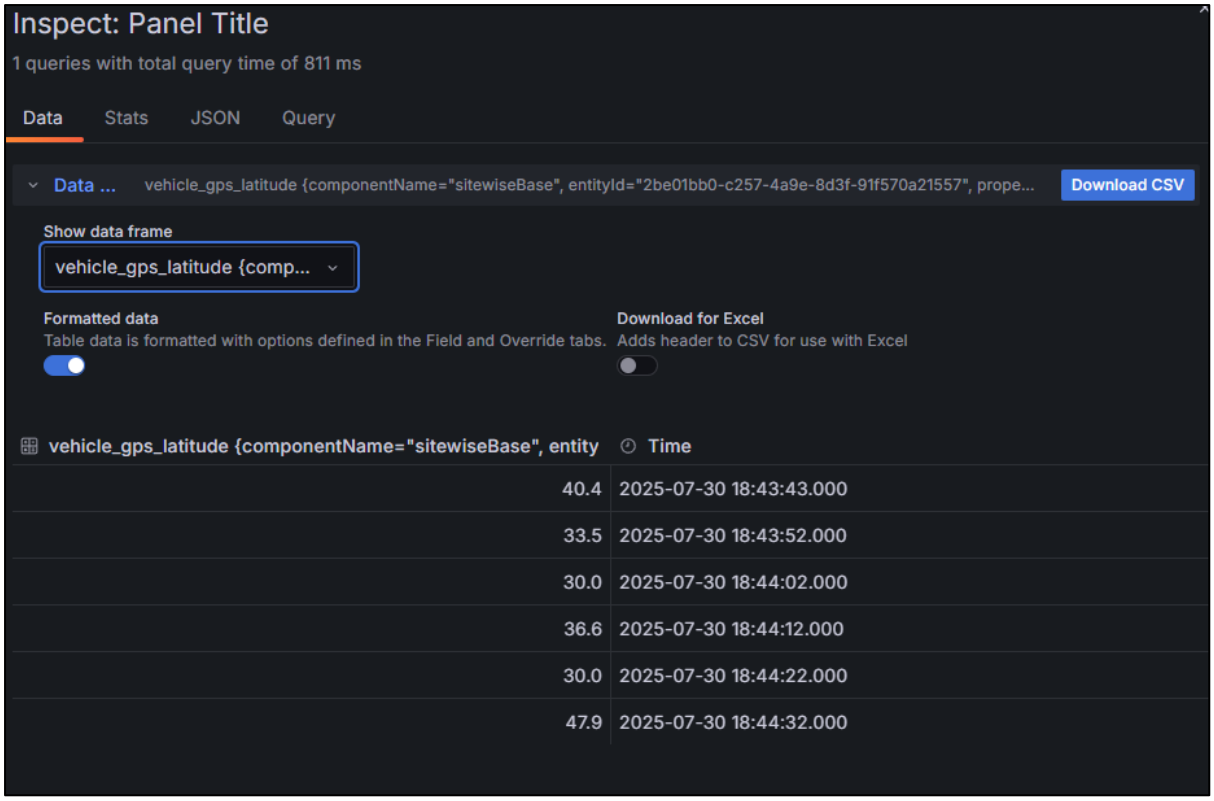
Some properties like sitewiseAssetId and sitewiseAssetModelId are not time series but are essential identifiers used by TwinMaker to associate this component with the correct SiteWise asset and model.

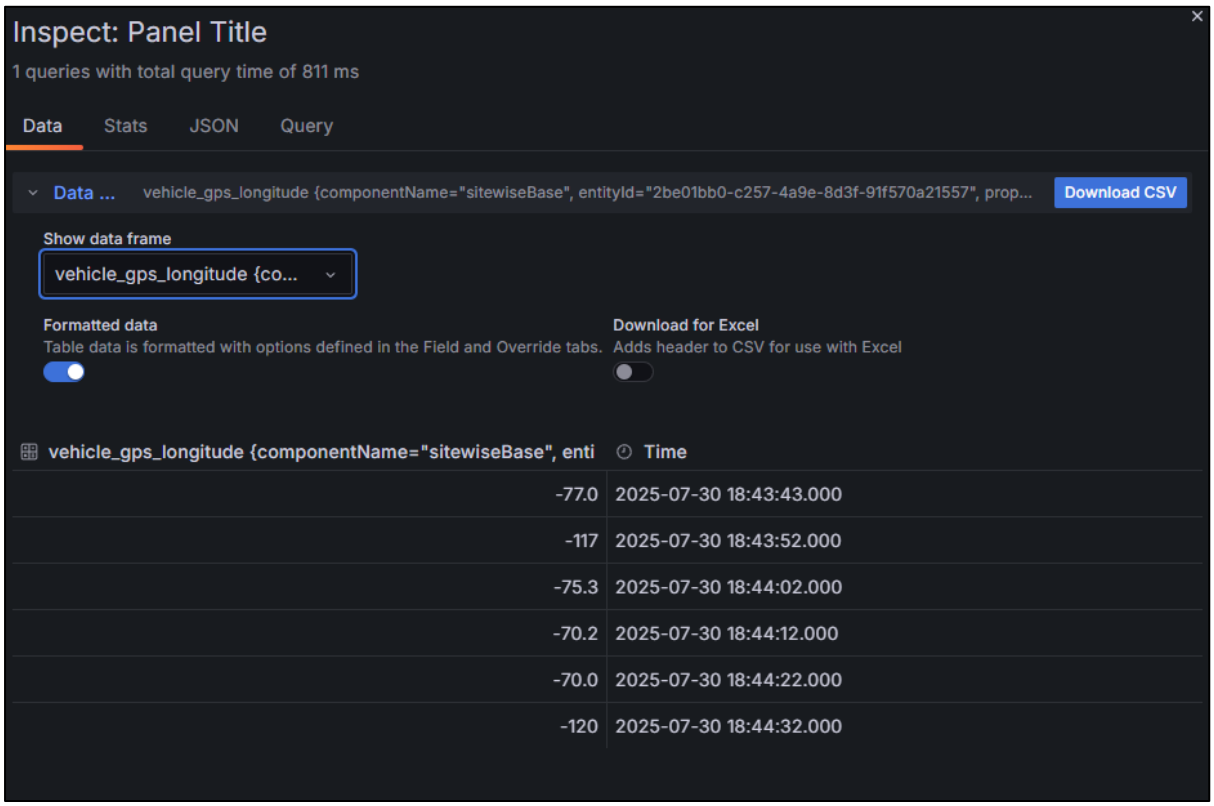
In summary, this configuration allows the digital twin to synchronize with real-world data streams for monitoring and analytics using AWS services.

As proof, this is the Grafana which has been set up (Will be covered later in the report). As you can see the data source is twinmaker. From the inspect Panel you can see the history of the data streamed, for route risk level is the same value each time whereas for the latitude and longitude is the different values following the CSV



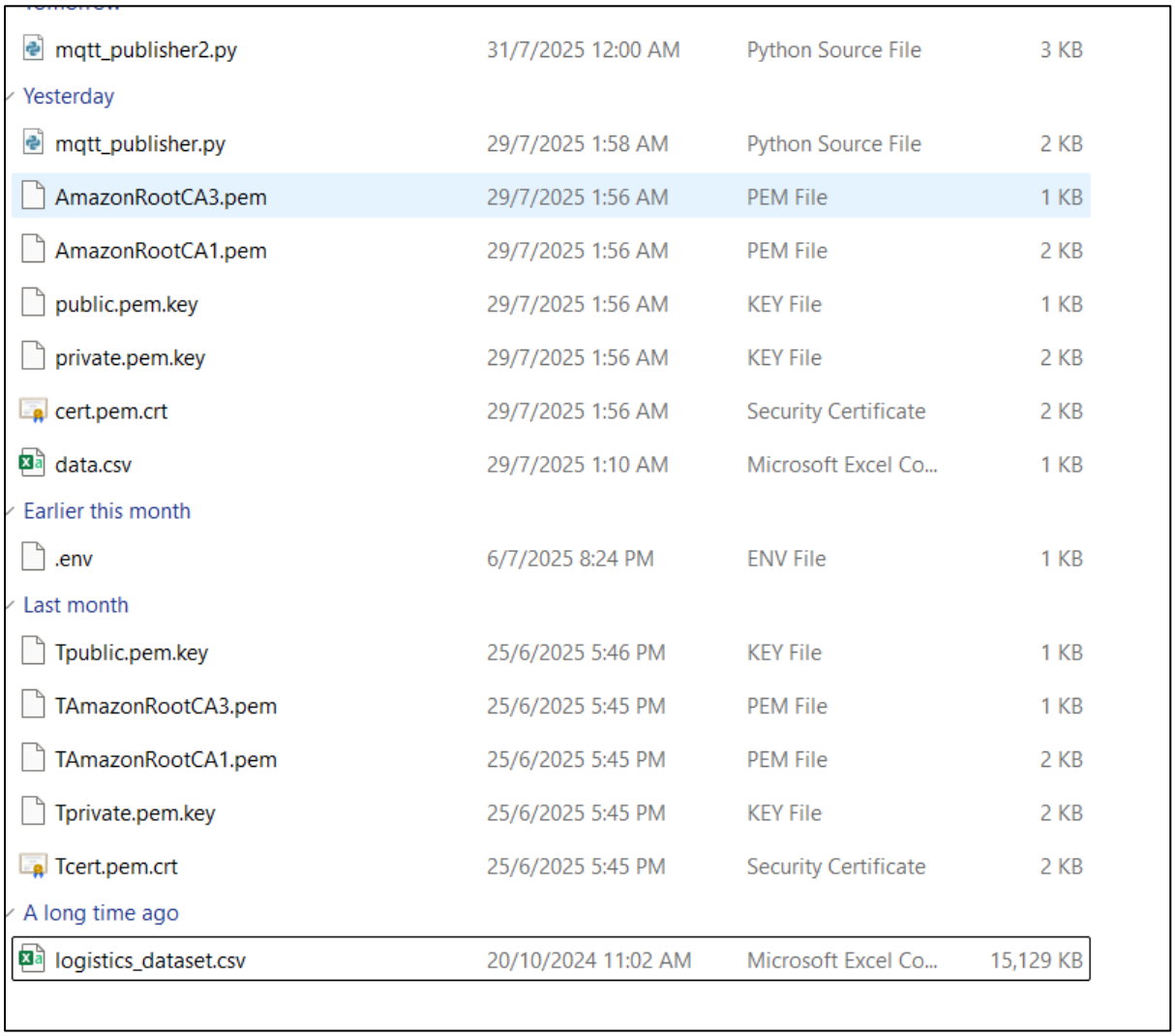






The values are streamed into Grafana which shows that twinmaker is connected.

Now let's try from the original Dataset but only 6 values from 3 attributes.



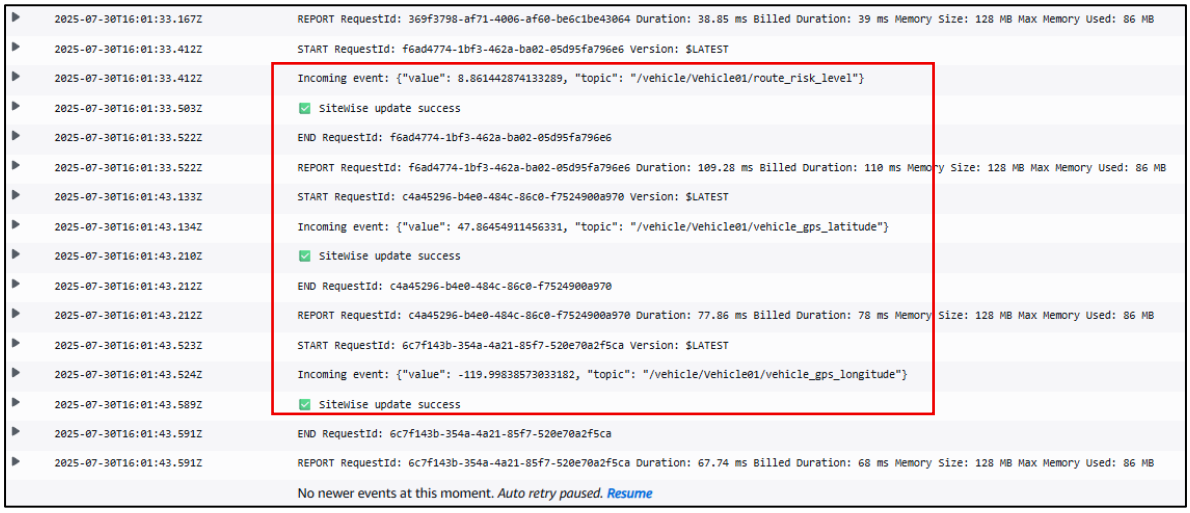
A new python file called mqtt_publisher2.py is created

```
with open('logistics_dataset.csv', 'r', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        if row_count >= row_limit:
            break
```

Now opening the original dataset Logistics_dataset.csv

```
C:\Users\Deads\Downloads\logistics-data-stream\mqtt_publisher2.py:22: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client(protocol=mqtt.MQTTv311)
Connecting to MQTT broker...

📦 Simulated time: 2025-07-31 00:00:53
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 40.3755684751949255}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -77.01431774252585}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 1.1821159890280728}

📦 Simulated time: 2025-07-31 00:01:03
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 33.507818339052975}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -117.03690240506452}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 9.61198830435769}

📦 Simulated time: 2025-07-31 00:01:13
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 30.0206397890309065}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -75.26922403662825}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 6.570431288350116}

📦 Simulated time: 2025-07-31 00:01:23
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 36.6492225127589755}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -70.19052930366378}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 0.5489520412771786}

📦 Simulated time: 2025-07-31 00:01:33
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 30.0012792843956865}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -70.01219476429307}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 8.861442874133289}

📦 Simulated time: 2025-07-31 00:01:43
✅ Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 47.864549114563315}
✅ Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -119.99838573033182}
✅ Published to /vehicle/Vehicle01/route_risk_level: {"value": 9.492603161323121}
PS C:\Users\Deads\Downloads\logistics-data-stream>
```

On the console its running. Now to check cloudwatch and sitewise

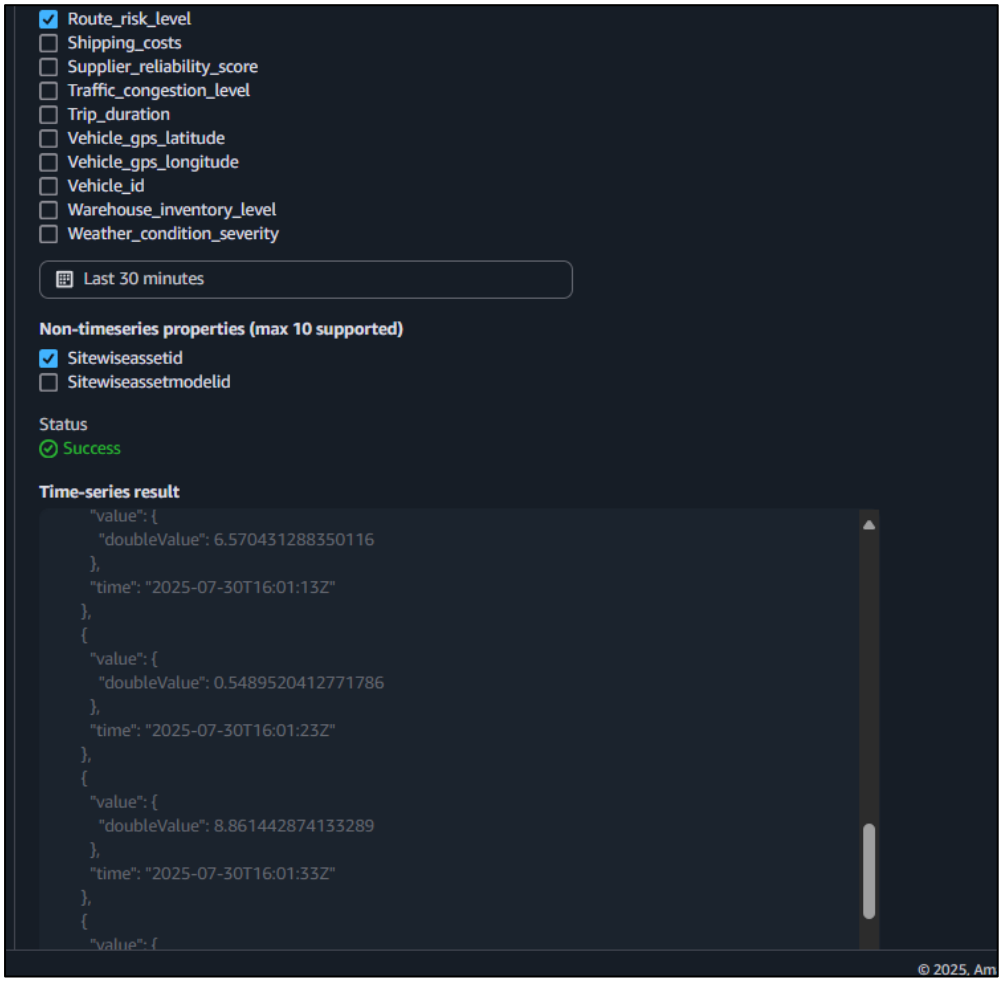


```
2025-07-30T16:01:33.167Z  REPORT RequestId: 369f3798-af71-4006-af60-be6c1be43064 Duration: 38.85 ms Billed Duration: 39 ms Memory Size: 128 MB Max Memory Used: 86 MB
2025-07-30T16:01:33.412Z  START RequestId: f6ad4774-1bf3-462a-ba02-05d95fa796e6 Version: $LATEST
2025-07-30T16:01:33.412Z  Incoming event: {"value": 8.861442874133289, "topic": "/vehicle/Vehicle01/route_risk_level"}
2025-07-30T16:01:33.503Z  ✅ SiteWise update success
2025-07-30T16:01:33.522Z  END RequestId: f6ad4774-1bf3-462a-ba02-05d95fa796e6
2025-07-30T16:01:33.522Z  REPORT RequestId: f6ad4774-1bf3-462a-ba02-05d95fa796e6 Duration: 109.28 ms Billed Duration: 110 ms Memory Size: 128 MB Max Memory Used: 86 MB
2025-07-30T16:01:43.133Z  START RequestId: c4a45296-b4e0-484c-86c0-f7524900a970 Version: $LATEST
2025-07-30T16:01:43.134Z  Incoming event: {"value": 47.86454911456331, "topic": "/vehicle/Vehicle01/vehicle_gps_latitude"}
2025-07-30T16:01:43.210Z  ✅ SiteWise update success
2025-07-30T16:01:43.212Z  END RequestId: c4a45296-b4e0-484c-86c0-f7524900a970
2025-07-30T16:01:43.212Z  REPORT RequestId: c4a45296-b4e0-484c-86c0-f7524900a970 Duration: 77.86 ms Billed Duration: 78 ms Memory Size: 128 MB Max Memory Used: 86 MB
2025-07-30T16:01:43.523Z  START RequestId: 6c7f143b-354a-4a21-85f7-520e70a2f5ca Version: $LATEST
2025-07-30T16:01:43.524Z  Incoming event: {"value": -119.99838573033182, "topic": "/vehicle/Vehicle01/vehicle_gps_longitude"}
2025-07-30T16:01:43.589Z  ✅ SiteWise update success
2025-07-30T16:01:43.591Z  END RequestId: 6c7f143b-354a-4a21-85f7-520e70a2f5ca
2025-07-30T16:01:43.591Z  REPORT RequestId: 6c7f143b-354a-4a21-85f7-520e70a2f5ca Duration: 67.74 ms Billed Duration: 68 ms Memory Size: 128 MB Max Memory Used: 86 MB
No newer events at this moment. Auto retry paused. Resume
```

route_risk_level	0985fe34-d5da-48b5-9098-baf94e926314	-	/vehicle/Vehicle01/route_risk_level	level	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/0985fe34-d5da-48b5-9098-baf94e926314	9.492603161323121	July 31, 2025 at 00:01:43 (UTC+8:00)

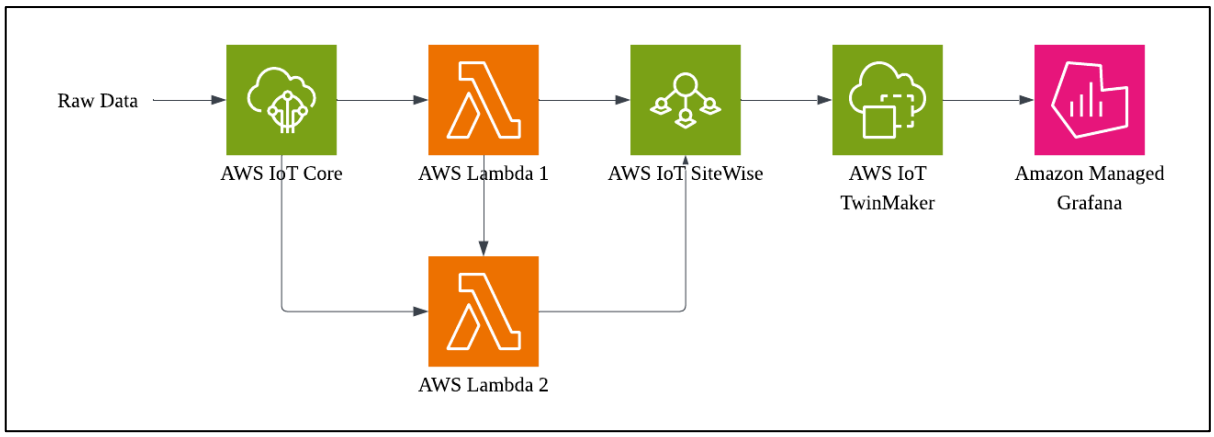
The new value has been updated

And finally, for Twinmaker



So, we can conclude that the CSV dataset can be streamed to twinmaker and that twinmaker is able to receive the data.

CLO3 DIGITAL TWIN AWS ARCITECHTURE DIAGRAM



The architecture diagram represents the complete data and control flow within the Digital Twin-based logistics risk monitoring system. It illustrates how we integrate several AWS services to enable real-time data ingestion, prediction processing, and 3D visualization.

1. **AWS IoT Core acts as the central ingestion layer.** It receives streaming telemetry values (fixed, realtime, and periodic features) from a local CSV simulator via MQTT. These values represent the sensor status of logistics vehicles such as GPS, temperature, fuel consumption, and disruption risk scores.

2. **The first AWS Lambda function is responsible for:**

    - Reading data row-by-row from the CSV file.
    - Publishing individual sensor values to MQTT topics under /vehicle/Vehicle01/*.
    - These MQTT topics are then processed by AWS IoT Core rules to forward the data to AWS IoT SiteWise for storage and visualization.
    - Once every 6 rows, the Lambda aggregates fixed, periodic, and average real-time features into a single JSON payload and publishes it to the /vehicle/Vehicle01/trigger_prediction topic, triggering the prediction process.

3. **AWS IoT SiteWise functions as the time-series data repository. It stores:**

    - Raw sensor values published by Lambda 1.
    - Model prediction outputs published by Lambda 2.

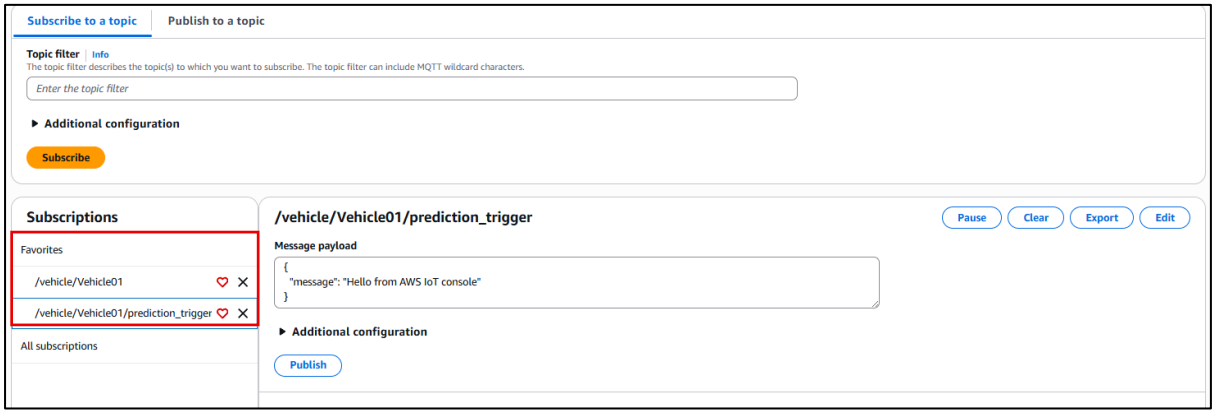
- These values are used to power both real-time dashboards and 3D scene rendering in the digital twin.

4. **The second AWS Lambda function (predictor) is triggered when a message is published to the /vehicle/Vehicle01/trigger_prediction MQTT topic. It:**
    - Extracts fixed, periodic, and averaged real-time features from the trigger payload.
    - Performs preprocessing and invokes a trained Random Forest model stored locally.
    - Encodes the prediction output (Low Risk = 1, High Risk = 0, Moderate Risk = 2).
    - Publishes the predicted risk classification back to AWS IoT SiteWise.
5. **AWS IoT TwinMaker integrates directly with SiteWise and acts as the core 3D visualization engine**. It renders a digital twin of the vehicle and updates the model's color, position, and state in real time using sensor and prediction values from SiteWise.
6. **Amazon Managed Grafana reads data directly from TwinMaker (via SiteWise) and displays it on a live dashboard**

The architecture is modular, scalable, and fully serverless, making it ideal for real-time digital twin simulations and intelligent logistics monitoring in supply chain applications. It enables predictive analytics, situational awareness, and decision support using only managed services and minimal infrastructure overhead.

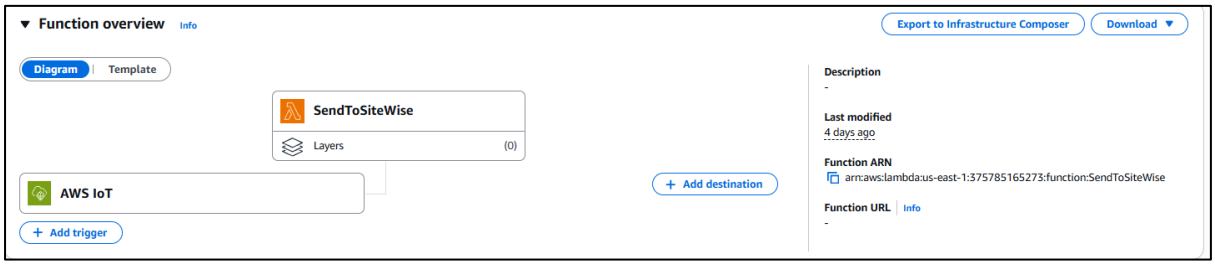
CLO3 Digital Twin Integration & Configuration with AWS Twinmaker

As mentioned in the previous parts. The data (CSV data) is sent from the terminal to mqtt(IoT Core)



These two topic are subscribed to.

Went the data flows the mqtt publishes the data through Lambda 1



Which sends the data to Sitewise based on the intervals stated before.

Lambda_function.py

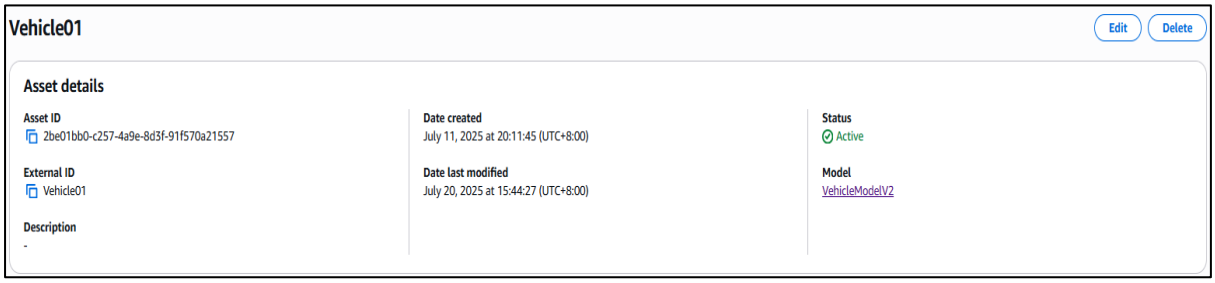
```
import boto3

import time

import json


sitewise = boto3.client('iotsitewise')


alias_map = {

    # === Fixed Features ===

    "lead_time_days": "/vehicle/Vehicle01/lead_time_days",

    "supplier_reliability_score": "/vehicle/Vehicle01/supplier_reliability_score",

    "port_congestion_level": "/vehicle/Vehicle01/port_congestion_level",

    "weather_condition_severity": "/vehicle/Vehicle01/weather_condition_severity",

    "route_risk_level": "/vehicle/Vehicle01/route_risk_level",

    "handling_equipment_availability": "/vehicle/Vehicle01/handling_equipment_availability",

    "historical_demand": "/vehicle/Vehicle01/historical_demand",

    "driver_behavior_score": "/vehicle/Vehicle01/driver_behavior_score",

    "warehouse_inventory_level": "/vehicle/Vehicle01/warehouse_inventory_level",

    "loading_unloading_time": "/vehicle/Vehicle01/loading_unloading_time",


    # === Real-time Features ===

    "vehicle_gps_latitude": "/vehicle/Vehicle01/vehicle_gps_latitude",

    "vehicle_gps_longitude": "/vehicle/Vehicle01/vehicle_gps_longitude",

    "fuel_consumption_rate": "/vehicle/Vehicle01/fuel_consumption_rate",

    "traffic_congestion_level": "/vehicle/Vehicle01/traffic_congestion_level",
```

```
    "iot_temperature": "/vehicle/Vehicle01/iot_temperature",
    "cargo_condition_status": "/vehicle/Vehicle01/cargo_condition_status",
    "trip_duration": "/vehicle/Vehicle01/trip_duration",

    # === Periodic Features ===
    "fatigue_monitoring_score": "/vehicle/Vehicle01/fatigue_monitoring_score",
    "disruption_likelihood_score": "/vehicle/Vehicle01/disruption_likelihood_score",
    "eta_variation_hours": "/vehicle/Vehicle01/eta_variation_hours",
    "order_fulfillment_status": "/vehicle/Vehicle01/order_fulfillment_status",
    "shipping_costs": "/vehicle/Vehicle01/shipping_costs",
    "customs_clearance_time": "/vehicle/Vehicle01/customs_clearance_time",
    "delay_probability": "/vehicle/Vehicle01/delay_probability",
    "delivery_time_deviation": "/vehicle/Vehicle01/delivery_time_deviation"
}

def lambda_handler(event, context):
    try:
        print("Incoming event:", json.dumps(event))

        value = float(event.get("value"))

        # Get topic
        topic = event.get("topic")
        if not topic and "headers" in event:
```

```
    topic = event["headers"].get("mqtt-topic")

if not topic:
    print("❌ No topic found")
    return { "status": "error", "reason": "No topic provided" }

# Extract last part of topic
metric = topic.split("/")[-1]
alias = alias_map.get(metric)

if alias is None:
    print(f"⚠️ Unknown metric: {metric}")
    return { "status": "skipped", "reason": "unknown metric" }

timestamp = int(time.time())
entry = {
    'entryId': f"{metric}_{timestamp}".replace("/", "_"),
    'propertyAlias': alias,
    'propertyValues': [{
        'value': { 'doubleValue': value },
        'timestamp': { 'timeInSeconds': timestamp },
        'quality': 'GOOD'
    }]
}
```

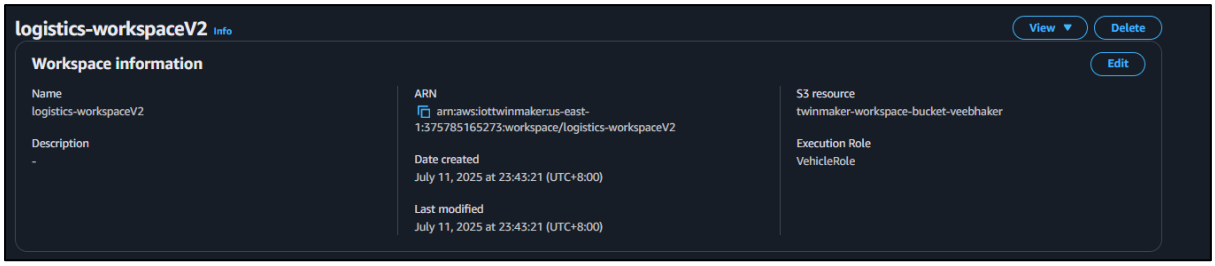
```
    response = sitewise.batch_put_asset_property_value(entries=[entry])
    print(f"✅ SiteWise update success for {alias}")
    return { 'status': 'success', 'response': response }

except Exception as e:
    print("❌ Lambda error:", e)
    return { 'status': 'error', 'message': str(e) }
```

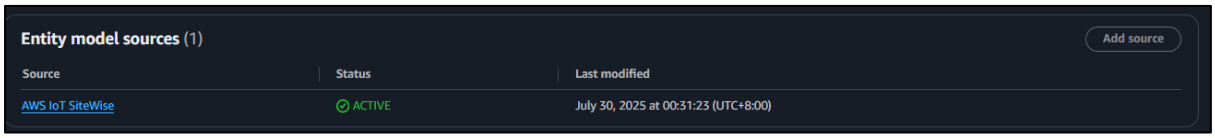
At sitewise the data is sent to the alias for each measurement



Name	ID	External ID	Alias	Unit	MQTT Notification status	Notification topic	Latest value	Latest value timestamp
cargo_condition_status	118e4a39-00e8-4a05-8b3e-cbd4d3e37e83	-	/vehicle/Vehicle01/cargo_condition_status	status	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/118e4a39-00e8-4a05-8b3e-cbd4d3e37e83	0.00000257	August 06, 2025 at 14:50:54 (UTC+8:00)
customs_clearance_time	6be99e00-1900-4e28-9a4b-2555d4b04e06	-	/vehicle/Vehicle01/customs_clearance_time	h	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/6be99e00-1900-4e28-9a4b-2555d4b04e06	1.634058338	August 06, 2025 at 14:50:44 (UTC+8:00)
delivery_time_deviation	474a331d-b71d-4a92-9767-e87708ae2f7d	-	/vehicle/Vehicle01/delivery_time_deviation	h	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/474a331d-b71d-4a92-9767-e87708ae2f7d	9.992328992	August 06, 2025 at 14:50:44 (UTC+8:00)
disruption_likelihood_score	d2ebfa91-b571-4f6f-9f01-aed873b23507	-	/vehicle/Vehicle01/disruption_likelihood_score	score	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/d2ebfa91-b571-4f6f-9f01-aed873b23507	0.070200435	August 06, 2025 at 14:50:44 (UTC+8:00)
driver_behavior_score	b89143da-c306-4233-ad96-d5895b3cfd2d	-	/vehicle/Vehicle01/driver_behavior_score	score	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/b89143da-c306-4233-ad96-d5895b3cfd2d	0.033843494	August 06, 2025 at 14:48:55 (UTC+8:00)

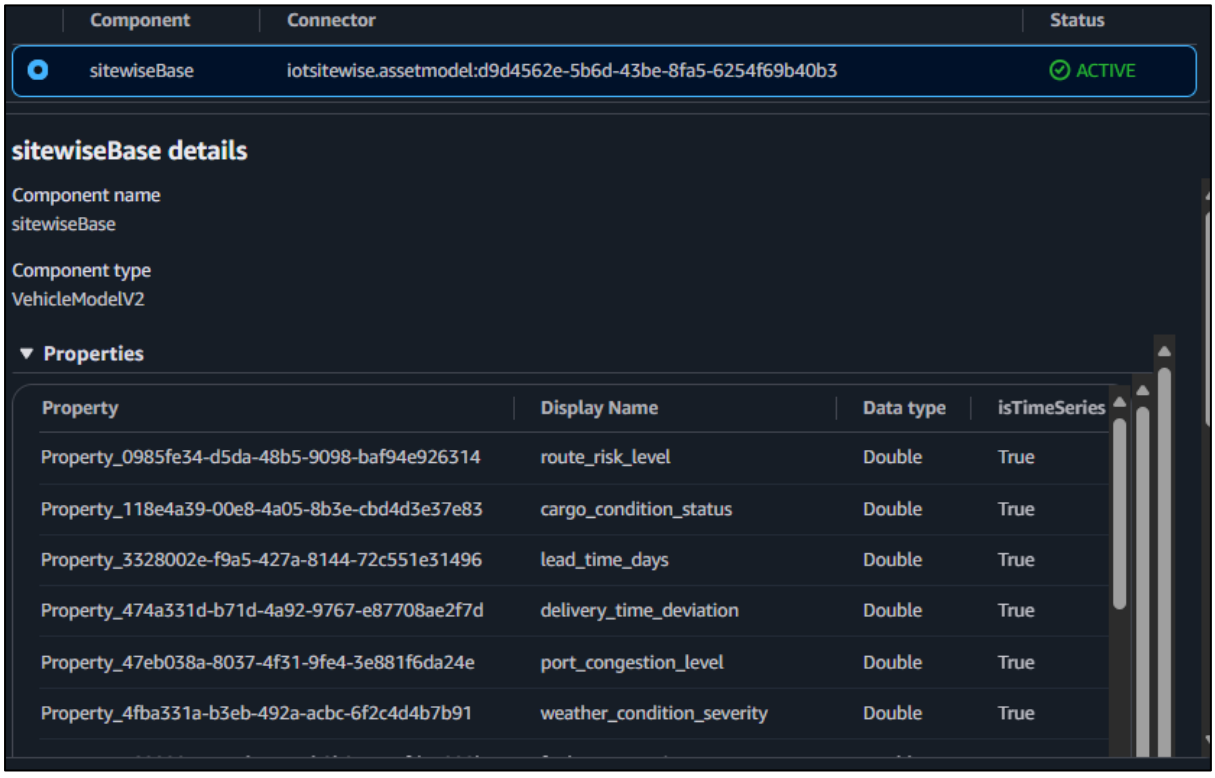
Once it reaches sitewise, the twinmaker connector explained in data streaming will allow the data to be transmitted to



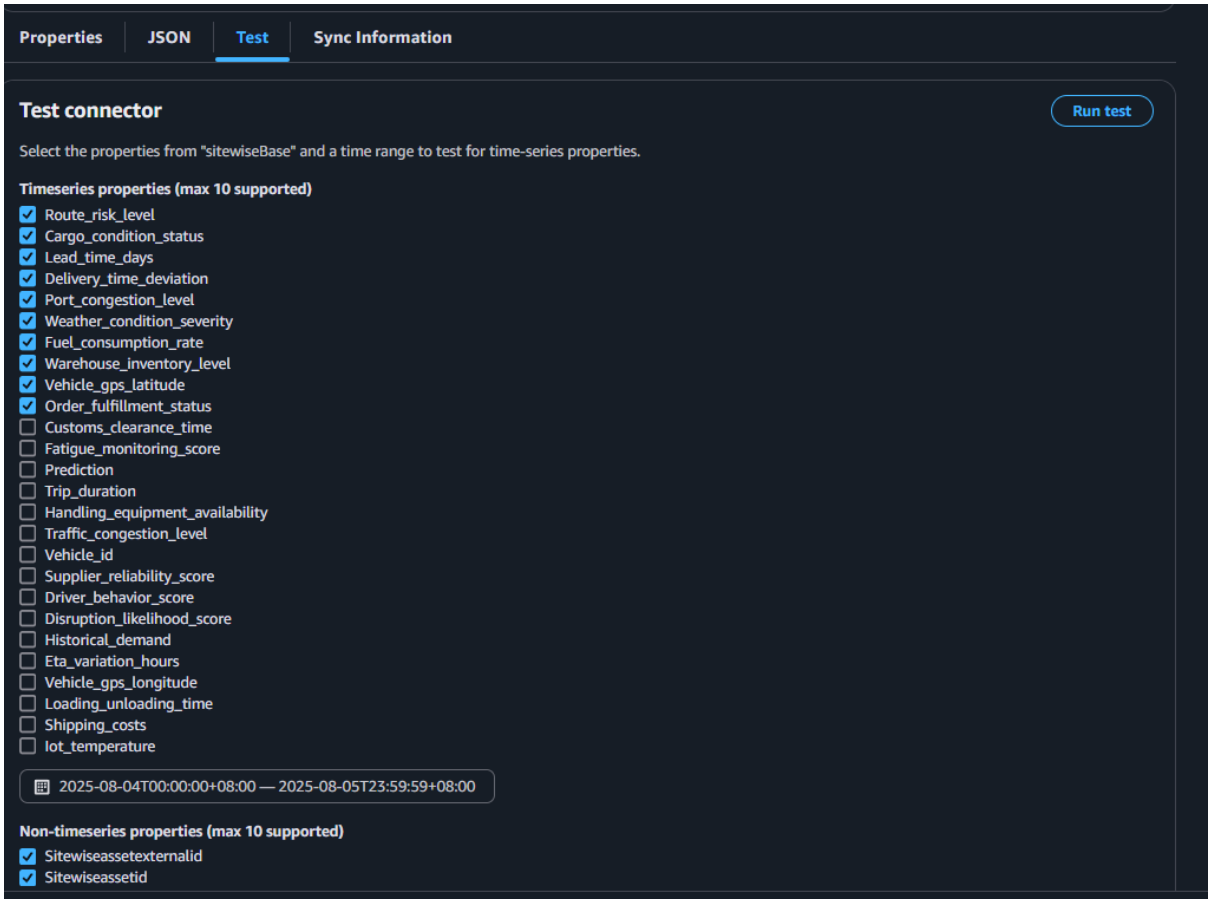


So basically if the data reaches sitewise, (if the component used is correct) the data will reach the designated twinmaker workspace.

The entity we created Vehicle01

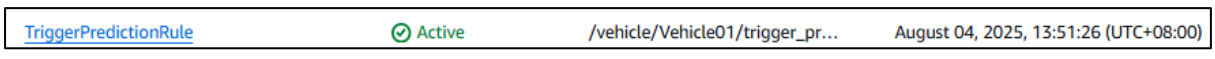


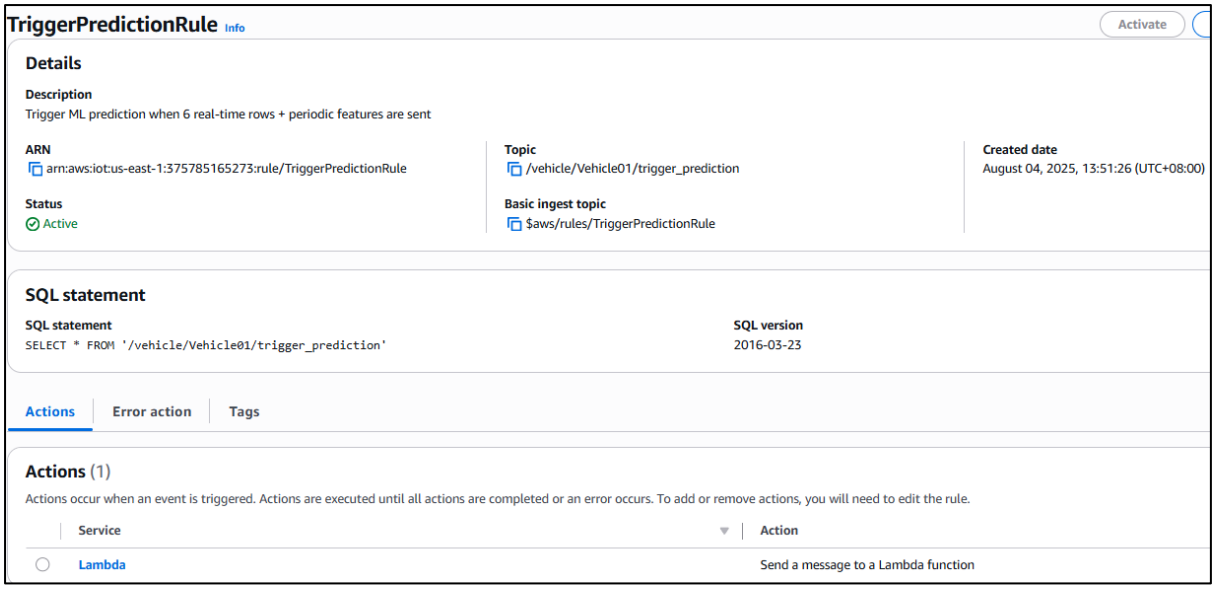
All the values will be here, but we cant view them here as they are stored externally and are timeseries values. Which means we can only view them when we use the sceneviewer integrated with Grafana. We can do the data connection test to view whether the data is received.



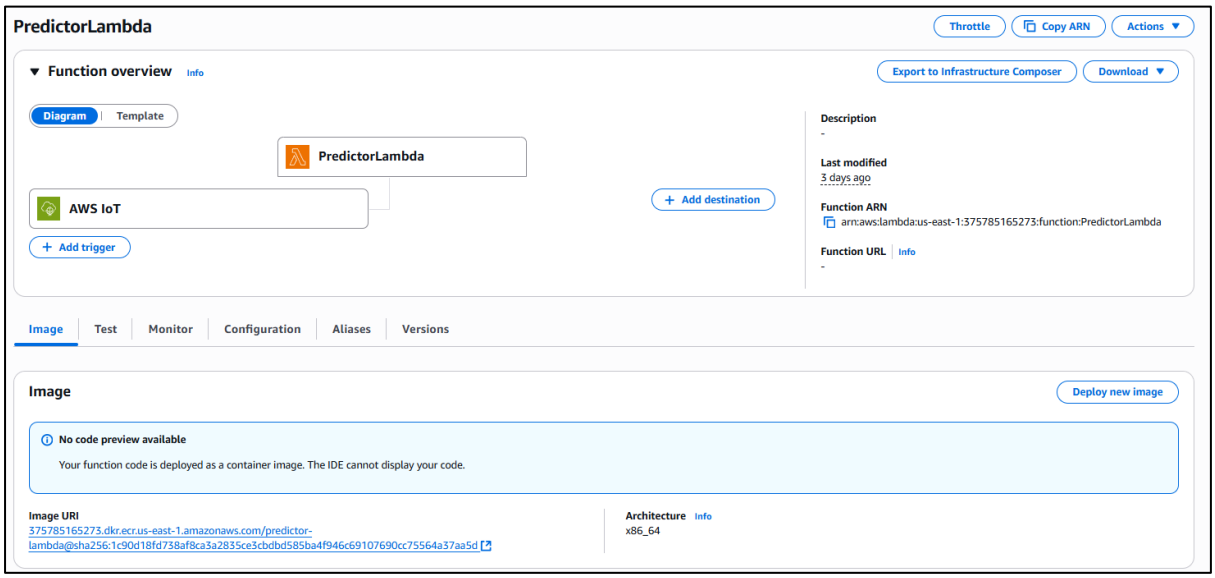
Max(only 10 values at a time)

Then after 1 minute is reached a prediction trigger will be sent at the mqtt topic subscribed and through an IoT rule



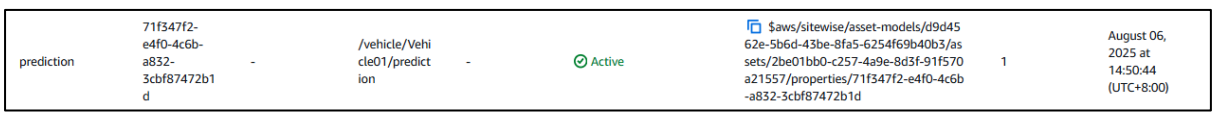


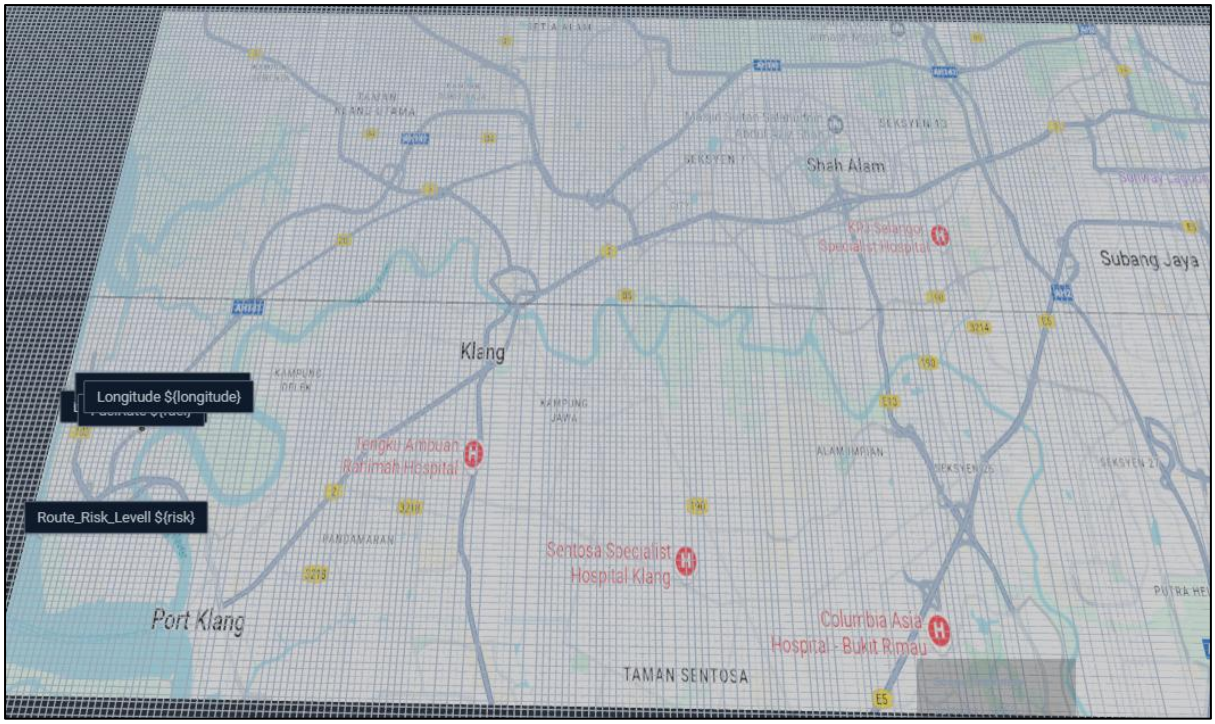
This sends a message to the second lambda



This lambda takes the fixed data, periodic data and averages the realtime data to make the prediction.

Then the prediction is sent to sitewise

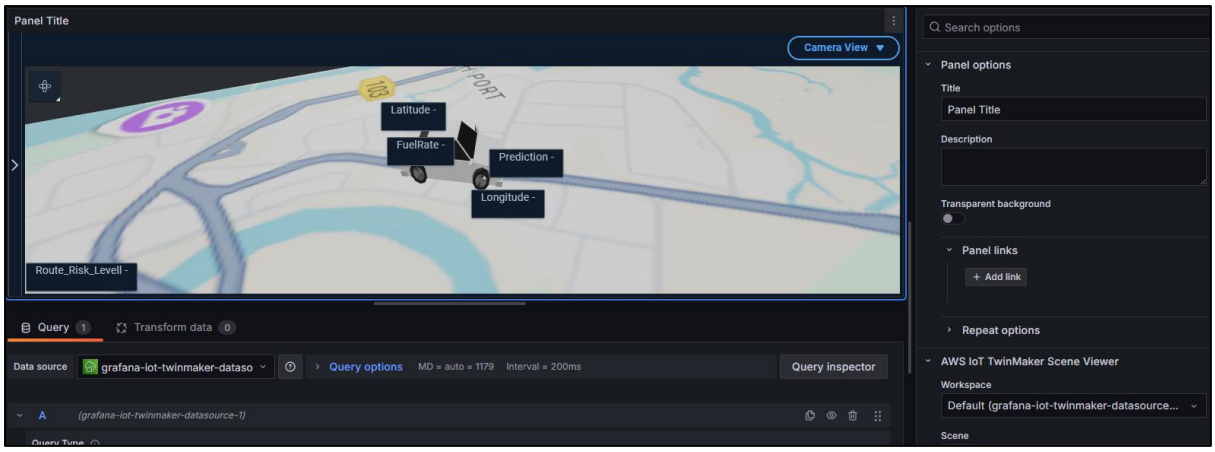




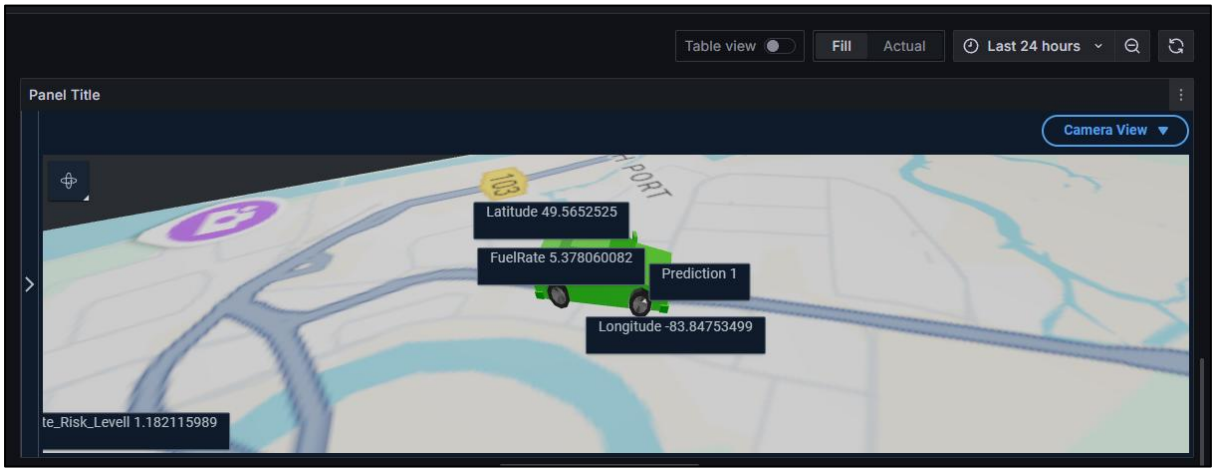
This is the scene viewer. (The detailed explanation for the scene viewer will be explained in the visualisation section)

Note: We are not showing all values in the scene as we have 25 measurements but do know that for the prediction all values are required and without even one value the prediction cannot be made.

Now at Grafana(More detailed explanation in Visualisation section)



Theres no values as its set to take the values based on the time.



If give a baseline of last 24 hours the last value updated will be uploaded. (As you see the prediction is 1 low risk so the car turns green).

CLO2 Develop Digital Twin AI Model (Train)

```
from google.colab import files

uploaded = files.upload()  # Upload 'logistics_dataset.csv'

import pandas as pd
import pickle
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report

# Load dataset
df = pd.read_csv('logistics_dataset.csv')
df = df.ffill()

# Parse timestamps and compute trip duration
df['timestamp'] = pd.to_datetime(df['timestamp'], dayfirst=True)
df['trip_duration'] = df['timestamp'].diff().dt.total_seconds().fillna(0)

# Encode target labels
le = LabelEncoder()
df['risk_classification_encoded'] = le.fit_transform(df['risk_classification'])

# Features and labels
X = df.drop(columns=['risk_classification', 'risk_classification_encoded', 'timestamp'])
y = df['risk_classification_encoded']

# SMOTE to balance classes
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X, y)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size=0.2, random_state=42, stratify=y_sm)

# Train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Save model and objects with pickle
with open('risk_model_smote.pkl', 'wb') as f: pickle.dump(model, f)
with open('label_encoder.pkl', 'wb') as f: pickle.dump(le, f)
with open('X_train.pkl', 'wb') as f: pickle.dump(X_train, f)
with open('X_test.pkl', 'wb') as f: pickle.dump(X_test, f)
with open('y_train.pkl', 'wb') as f: pickle.dump(y_train, f)
with open('y_test.pkl', 'wb') as f: pickle.dump(y_test, f)

# Evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=le.classes_))
```

```
              precision    recall  f1-score   support

    High Risk       1.00      1.00      1.00      4789
     Low Risk       1.00      1.00      1.00      4789
Moderate Risk       1.00      1.00      1.00      4789

     accuracy                           1.00     14367
    macro avg       1.00      1.00      1.00     14367
 weighted avg       1.00      1.00      1.00     14367
```

Interpretation:

- All classes are predicted perfectly.
- Explanation of Code:

- **Dataset**: A CSV file containing logistics-related features and a target column risk_classification.
- **Missing Values Handling**: Forward-fill (ffill) is applied to fill any missing values based on previous rows.
- **Timestamp Parsing**: Converts date strings to Python datetime objects with dayfirst=True due to the format like 13/1/2021.
- **Trip Duration**: A new feature trip_duration is calculated as the time difference between rows in seconds.
- **LabelEncoder**: Converts string labels (e.g., "Low Risk", "Moderate Risk", "High Risk") into numeric labels.

Example encoding:

bash

CopyEdit

'High Risk' → 0

'Low Risk' → 1

'Moderate Risk' → 2

- X = df.drop(columns=['risk_classification', 'risk_classification_encoded', 'timestamp']) y = df['risk_classification_encoded']
- Features (X): All relevant numeric columns used for training.
- Target (y): Encoded classification label to be predicted.

SMOTE (Synthetic Minority Over-sampling Technique):

- Balances the dataset by generating synthetic examples for underrepresented classes.
- Avoids bias in the model toward majority classes.
- **Model**: RandomForestClassifier
- an ensemble model that uses multiple decision trees.
- **Train/Test Split**: Data is split 80/20
- with stratified sampling to maintain class balance.
- risk_model_smote.pkl — the trained model
- label_encoder.pkl — the fitted LabelEncoder

- X_train.pkl, X_test.pkl, y_train.pkl, y_test.pkl — train/test sets

```
from sklearn.model_selection import cross_val_score
import numpy as np

scores = cross_val_score(model, X_sm, y_sm, cv=5, scoring='f1_macro')
print("Cross-validated F1 macro:", np.round(scores, 3))
print("Mean:", scores.mean())
```

```
Cross-validated F1 macro: [0.998 1.    1.    1.    1.   ]
Mean: 0.9996796722519008
```

- **Model Generalizes Well** — It's not just doing well on one train-test split; it performs extremely well across **5 different folds**.
- **Low Class Imbalance Issues** — SMOTE helped balance the classes, and now the classifier treats them equally well.
- **No Immediate Overfitting Detected** — If the features were leaking or overfitting, you'd typically see **variance** in F1 scores across folds, but here they're consistently high.

```
from google.colab import files

files.download('risk_model_smote.pkl')
files.download('label_encoder.pkl')
```

Saves the model and encoder in as pkl files.

```
print(model.feature_names_in_)
['vehicle_gps_latitude' 'vehicle_gps_longitude' 'fuel_consumption_rate'
'eta_variation_hours' 'traffic_congestion_level' 'warehouse_inventory_level'
'loading_unloading_time' 'handling_equipment_availability' 'order_fulfillment_status'
'weather_condition_severity' 'port_congestion_level' 'shipping_costs'
'supplier_reliability_score' 'lead_time_days' 'historical_demand' 'iot_temperature'
'cargo_condition_status' 'route_risk_level' 'customs_clearance_time' 'driver_behavior_score']
```

```
'fatigue_monitoring_score' 'disruption_likelihood_score' 'delay_probability'
'delivery_time_deviation' 'trip_duration']
```

These are the final 25 features the model was trained on.

Test The Model

```
import pandas as pd

test_input = pd.DataFrame([{
    'vehicle_gps_latitude': 3.1421,
    'vehicle_gps_longitude': 101.6890,
    'fuel_consumption_rate': 0.42,              # liters/km
    'eta_variation_hours': 0.5,                 # deviation from ETA in hours
    'traffic_congestion_level': 2,              # 0 (low) to 3 (high)
    'warehouse_inventory_level': 0.7,           # 0 to 1
    'loading_unloading_time': 35,               # in minutes
    'handling_equipment_availability': 0.9,     # 0 (none) to 1 (fully available)
    'order_fulfillment_status': 1,              # 0 (incomplete) or 1 (fulfilled)
    'weather_condition_severity': 1,            # 0 (clear) to 3 (severe)
    'port_congestion_level': 2,                 # 0 to 3
    'shipping_costs': 1200,                     # in local currency or USD
    'supplier_reliability_score': 0.86,         # 0 to 1
    'lead_time_days': 4,
    'historical_demand': 500,                   # units
    'iot_temperature': 23.4,                    # degrees Celsius
    'cargo_condition_status': 0.95,             # 0 (damaged) to 1 (good)
    'route_risk_level': 2,                      # 0 (safe) to 3 (risky)
    'customs_clearance_time': 48,               # in minutes
    'driver_behavior_score': 0.75,              # 0 (poor) to 1 (excellent)
    'fatigue_monitoring_score': 0.35,           # 0 (alert) to 1 (fatigued)
    'disruption_likelihood_score': 0.65,        # 0 to 1
    'delay_probability': 0.45,                  # 0 to 1
    'delivery_time_deviation': 0.2,             # hours
```

```
    'trip_duration': 720                        # seconds
}])
```

```
test_input = test_input[model.feature_names_in_]

# Predict class
pred = model.predict(test_input)[0]
class_name = le.inverse_transform([pred])[0]

# Predict class probabilities
probs = model.predict_proba(test_input)[0]

print("✅ Predicted Risk Class:", class_name)
print("\n🔎 Class Probabilities:")
for i, label in enumerate(le.classes_):
    print(f"{label}: {probs[i]:.4f}")
```

Predicted Risk Class: Moderate Risk : Class Probabilities: High Risk: 0.1000 Low Risk: 0.1300 Moderate Risk: 0.7700

- The model predicts "Moderate Risk" with **77% confidence**.
- Other classes have much lower probabilities, indicating strong model certainty.

This machine learning model was developed to classify delivery trip risk levels—High Risk, Moderate Risk, and Low Risk—based on a wide range of logistics-related features. The pipeline included data cleaning, timestamp parsing, feature engineering, label encoding, class balancing using SMOTE, model training, and robust evaluation.

Initial Class Imbalance and Limitations

In the original dataset, there was a significant class imbalance: the majority of the samples were labeled as High Risk and Moderate Risk, while Low Risk cases were underrepresented. This imbalance caused early versions of the model to be biased toward the majority classes. Although overall accuracy appeared high, performance metrics such as recall and precision for Low Risk predictions were poor. This posed a serious limitation, especially in operational scenarios where accurate identification of Low Risk trips are necessary for proactive resource planning and exception management.

Addressing Imbalance with SMOTE

To address this limitation, the dataset was balanced using **SMOTE (Synthetic Minority Over-sampling Technique)**. SMOTE works by generating synthetic data points for the minority class (*Low Risk*) by interpolating between existing samples in feature space. This method enhances the diversity of the minority class examples without simply duplicating existing rows, thereby helping the model learn meaningful patterns that represent real-world conditions.

The SMOTE process was applied **before splitting the dataset into training and test sets**, which is a best practice to avoid data leakage. The result was a balanced dataset with equal representation from all three risk categories.

Model Selection and Training

A **Random Forest Classifier** was chosen for this task due to its robustness, ability to handle high-dimensional and non-linear data, and resistance to overfitting. It was trained on the SMOTE-balanced dataset using a stratified 80/20 train-test split. The model achieved perfect classification metrics on the test set, with an F1-score of 1.00 for all three classes. While these results were promising, they were further validated through **5-fold cross-validation**, which yielded a mean macro F1-score of approximately **0.9997**, confirming the model's consistency across multiple data partitions.

Final Performance and Usage

After training, the model was tested on a new sample input to simulate deployment behaviour. The model correctly predicted the risk class with high confidence and provided class probability scores, demonstrating its readiness for integration into the digital twin deployment.

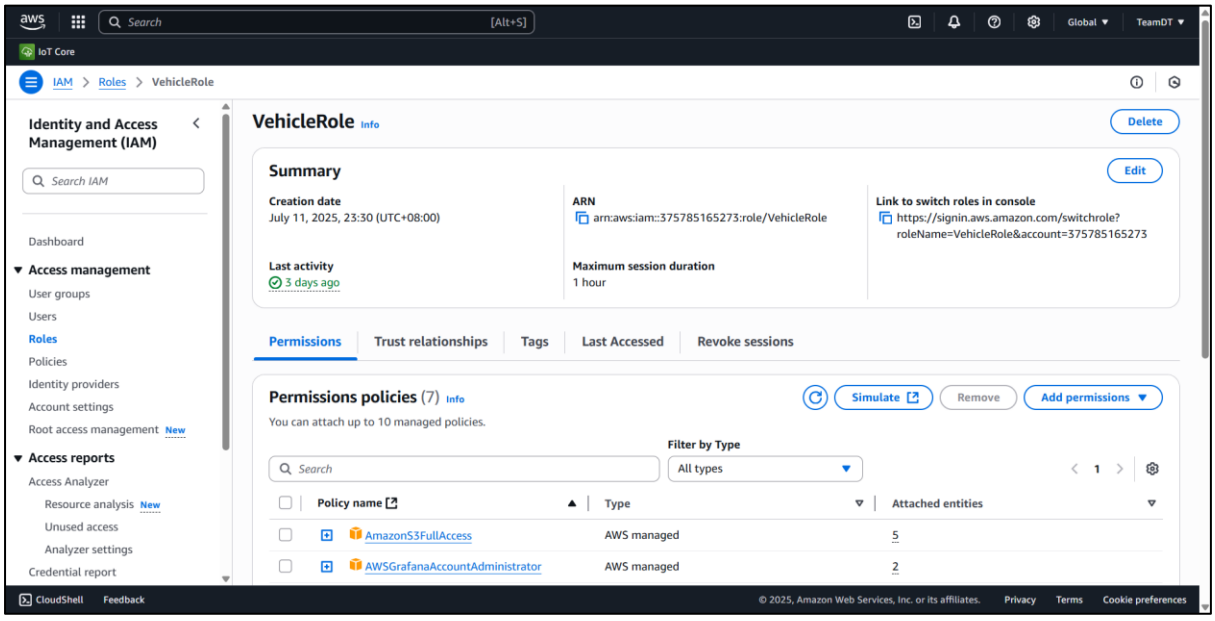
Summary

In summary, the final model represents a well-generalized and validated solution to the risk classification problem. The application of SMOTE was crucial in addressing the skewed class distribution, especially the underrepresentation of Low Risk cases, which significantly improved the model's ability to generalize across all classes. Combined with a Random Forest algorithm and rigorous evaluation, the resulting model is both accurate and reliable for deployment in logistics risk prediction tasks.

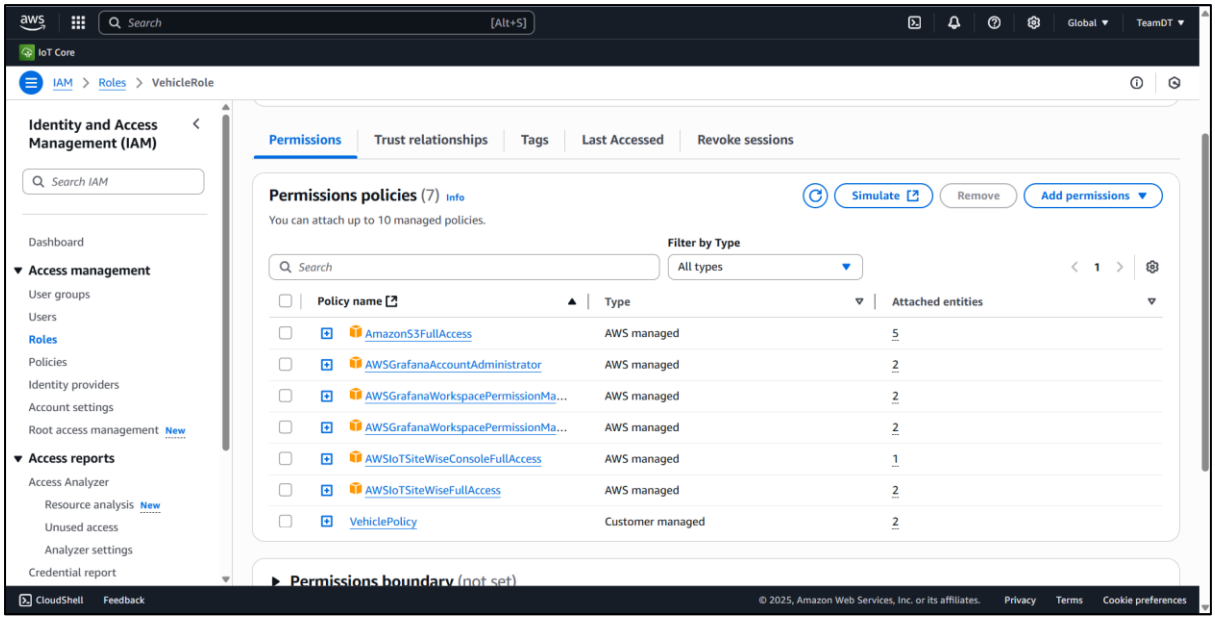
CLO3 Visualization & Dashboard

Setting up Grafana

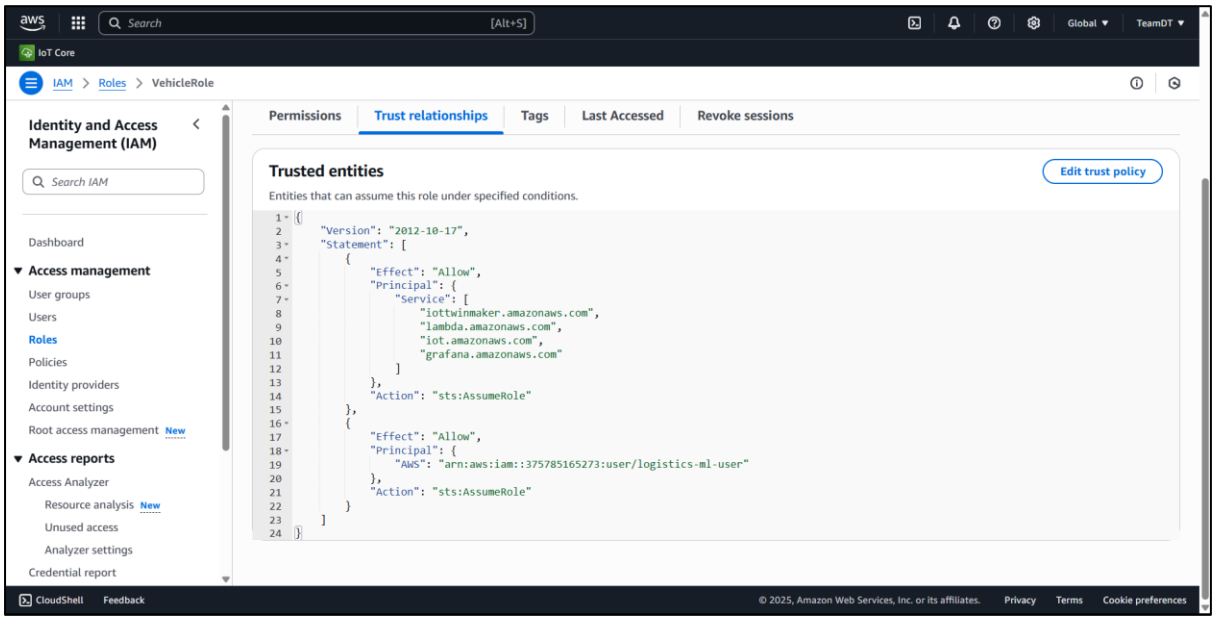
Grafana is chosen in our Digital Twin project because it is free and open-source, so we do not need to pay for using it. Since we already use many AWS services like IoT Core, SiteWise, and TwinMaker, we want to avoid extra cost from adding other paid visualization tools. Grafana can connect directly with AWS TwinMaker using the official plugin, and it helps us to see the real-time and historical data in a nice dashboard. This is useful for checking the vehicle location, fuel usage, and risk level easily. Even though we are not expert, Grafana is still easy to use for creating graphs and panels. Also, it helps us to check if the data from our system is working correctly or not. That is why we choose Grafana.



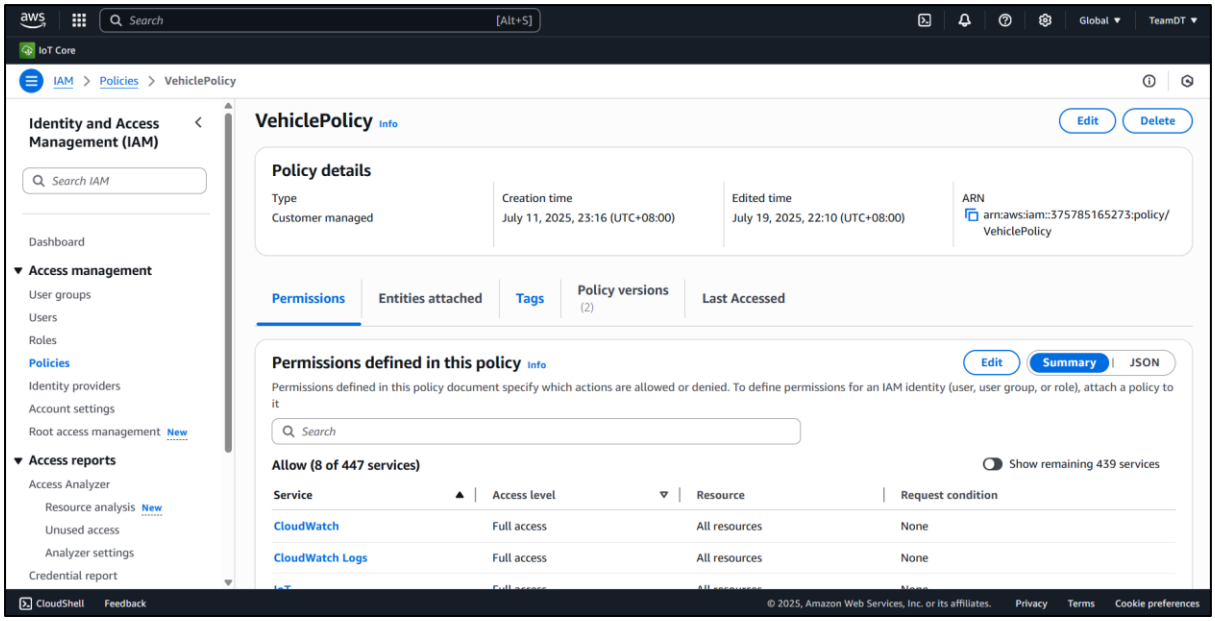
In the IAM, create a role and attached the following policies in the role.



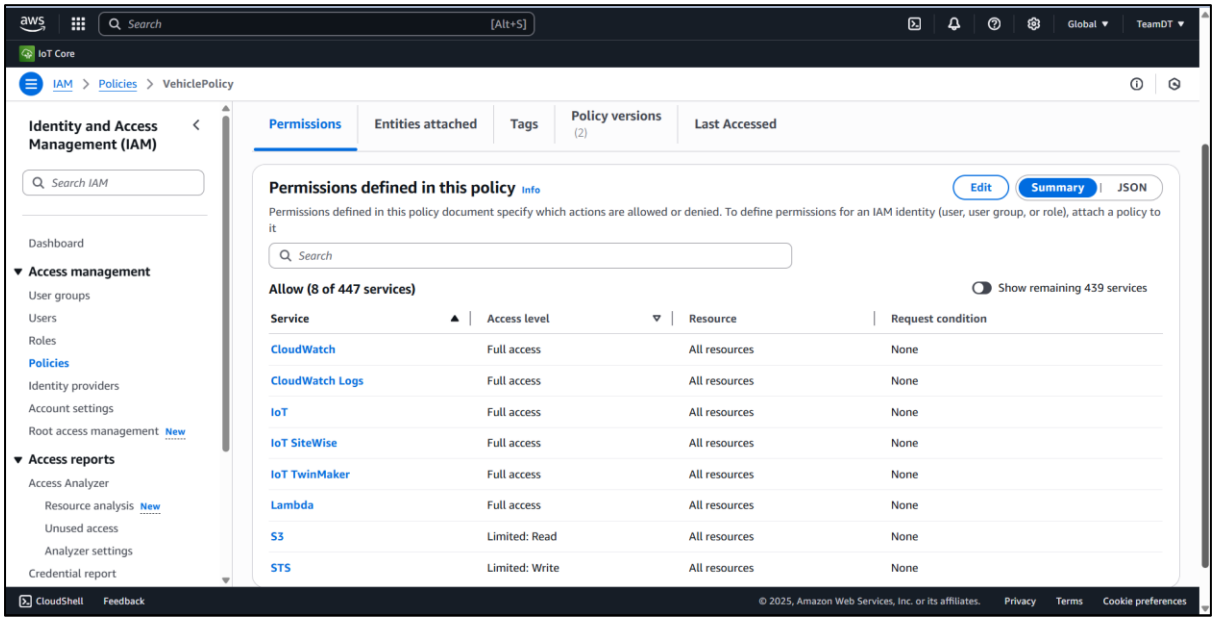
Continuation of the policies to be added in the role.



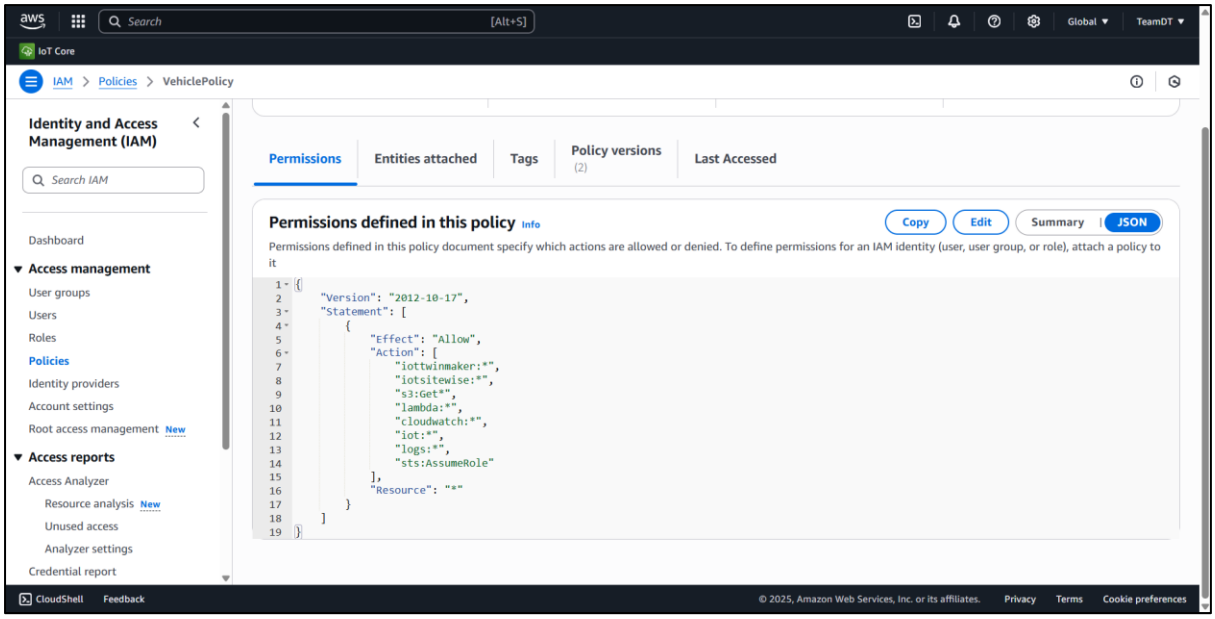
Modify the JSON Trust relationship coding according to the policies attached earlier.



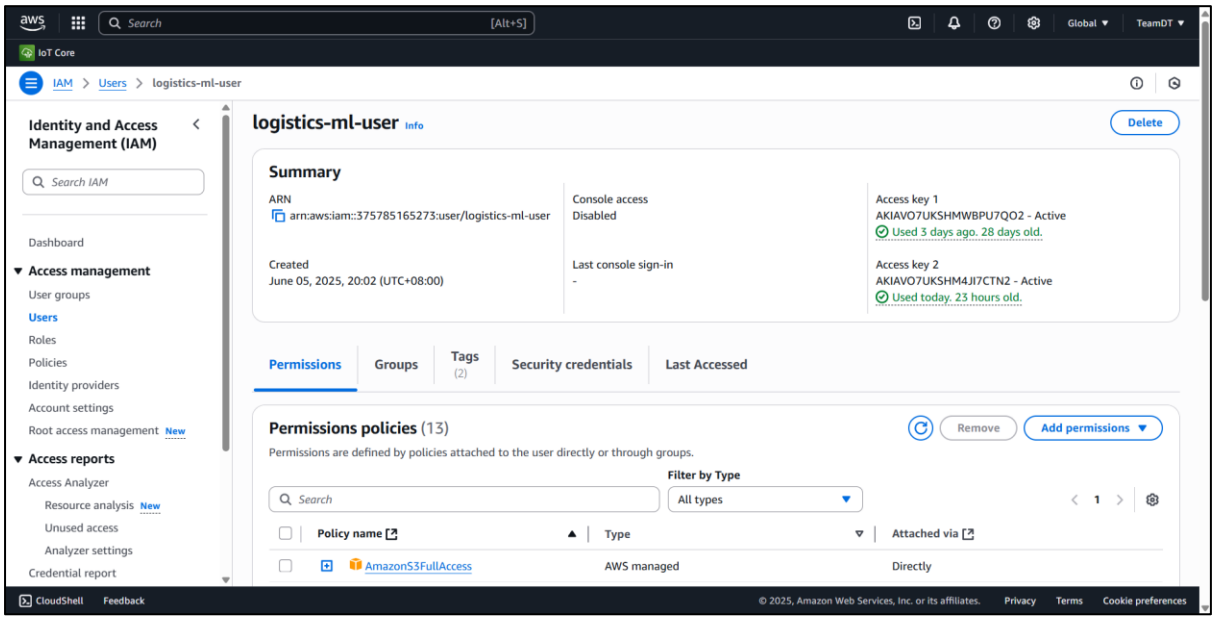
Policy created to be attached in the earlier role.



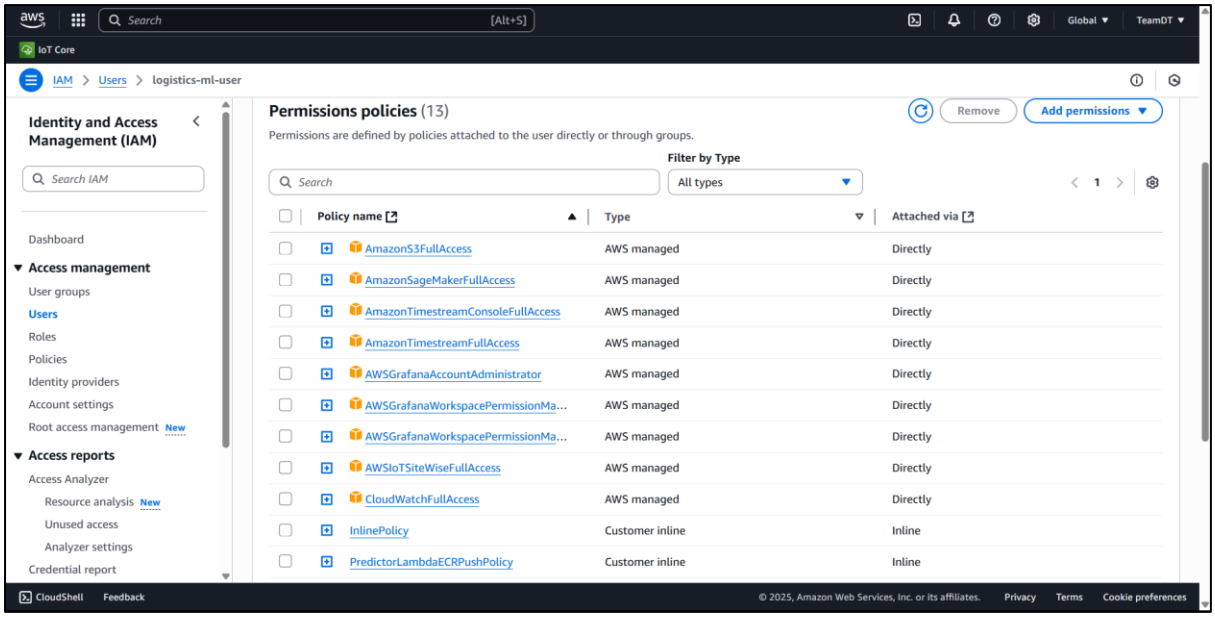
Attached the according list of services to be included in the policy.

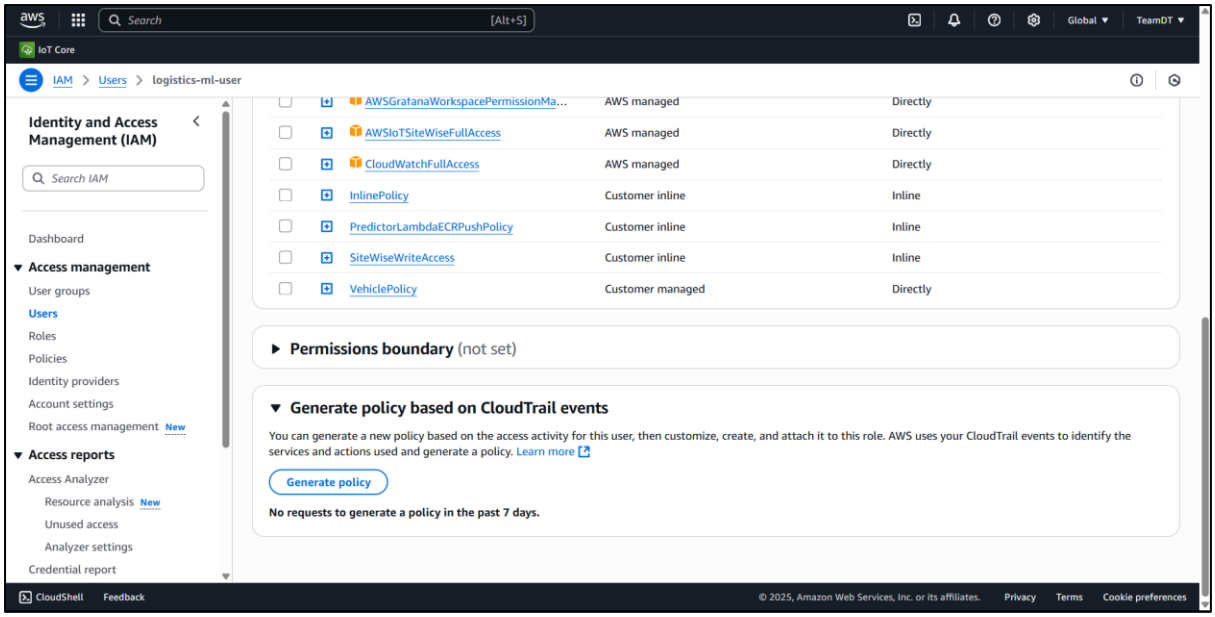


Edit the JSON permission coding based on the list of services attached earlier.

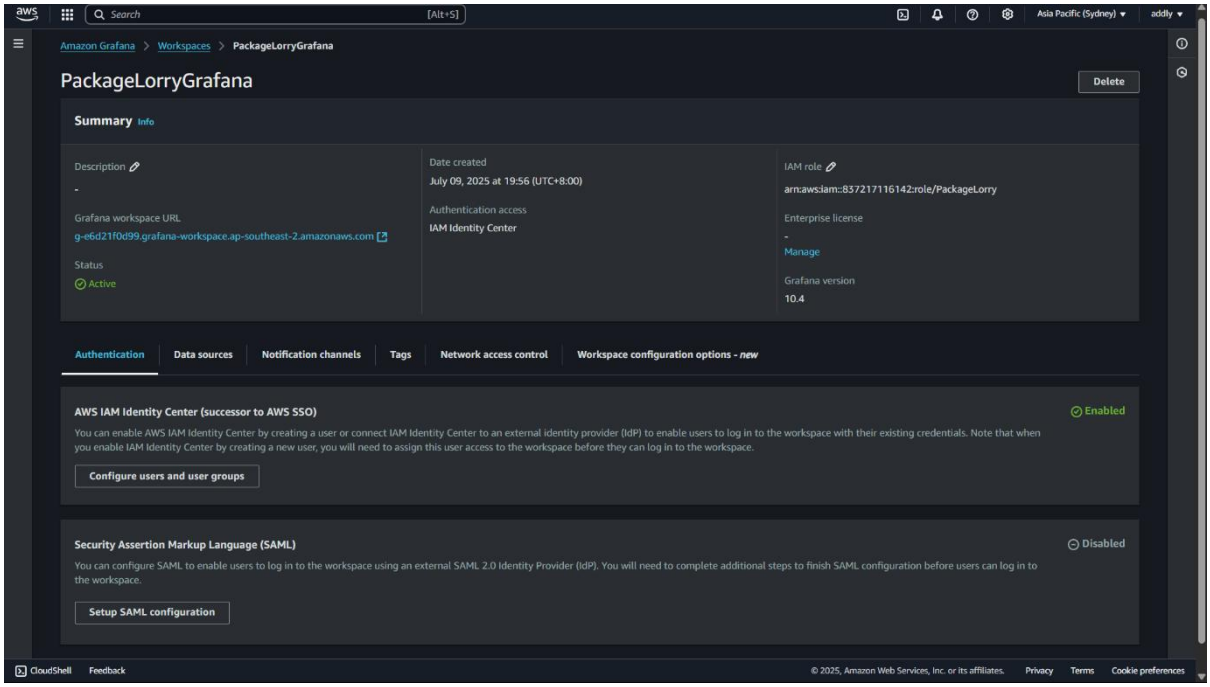


Create an IAM user.



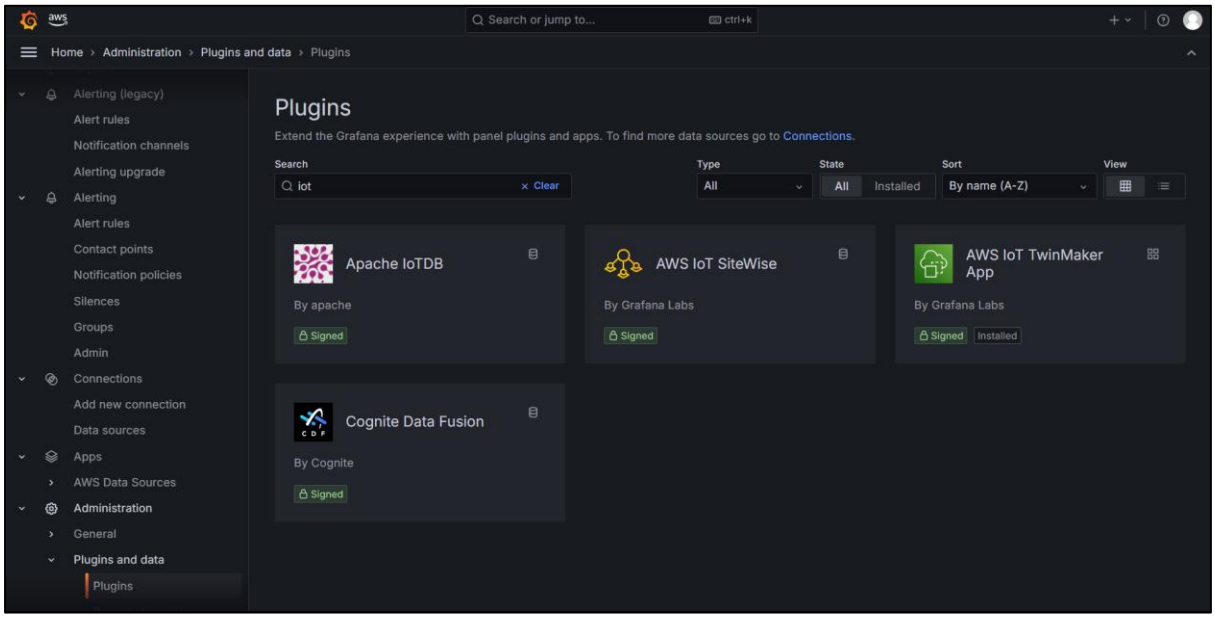


Attached the list of policies to the created user.

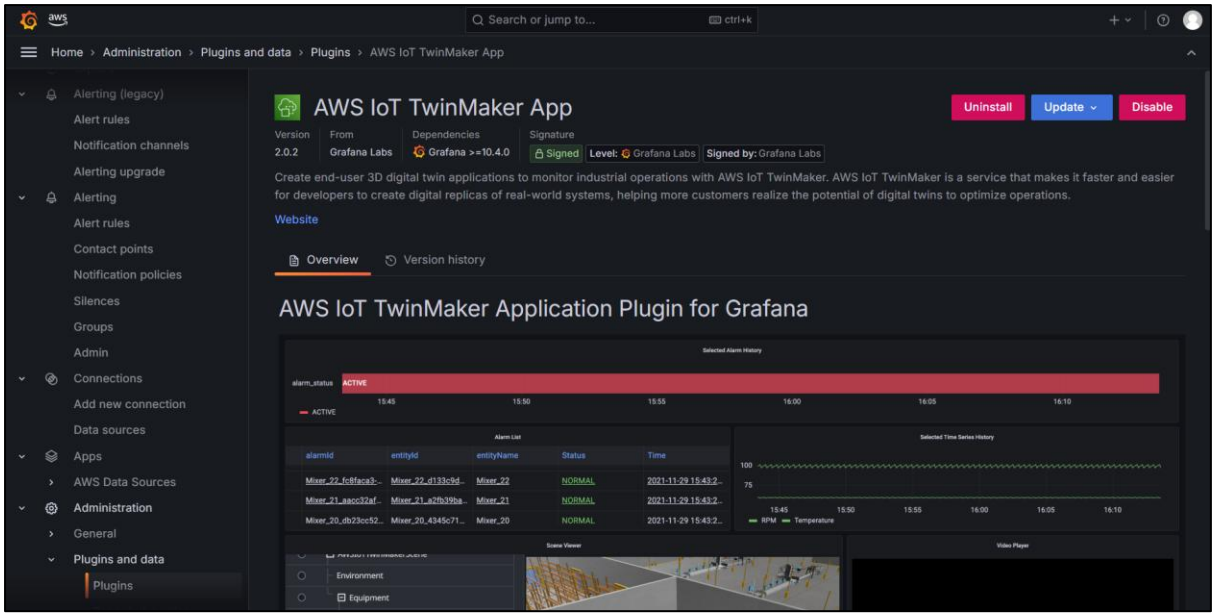


Create a Grafana workspace in Amazon Grafana.

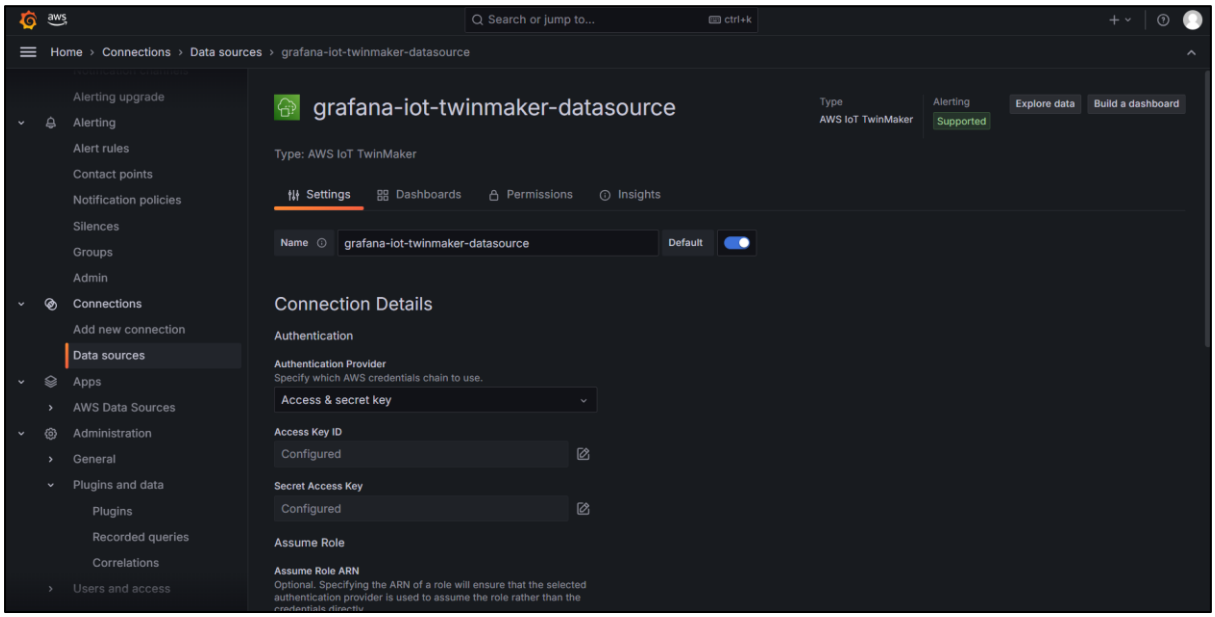
Add a user to be an admin to the created Grafana Workspace.



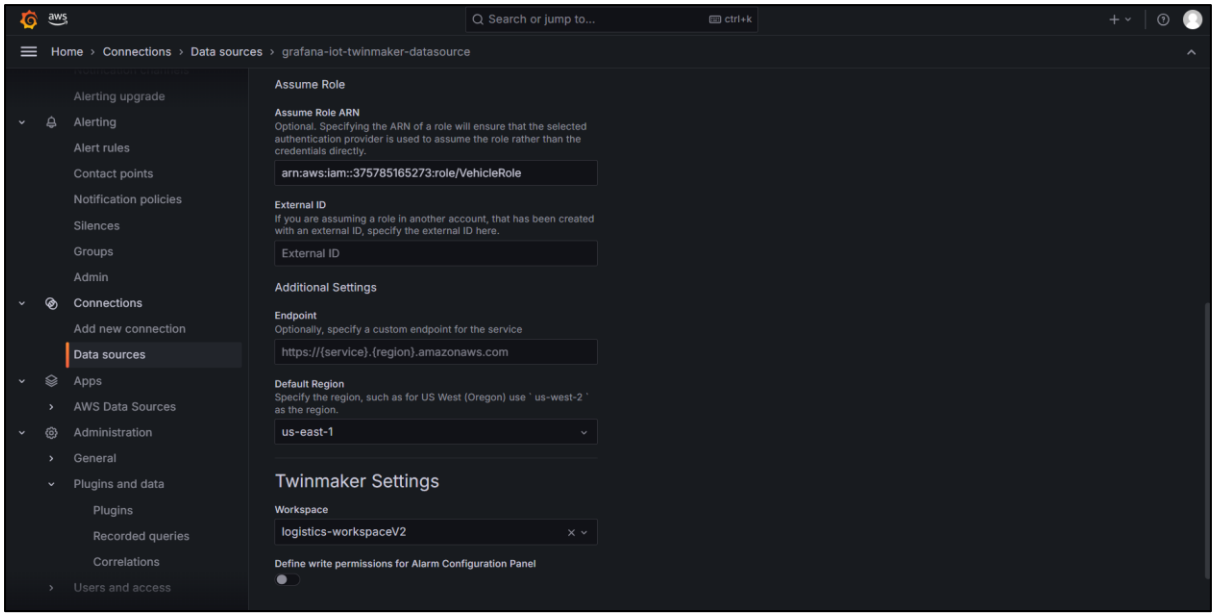
Open the URL link to the Grafana website. Sign in and click plugins and data from administration tab. Search AWS IoT TwinMaker App and install.



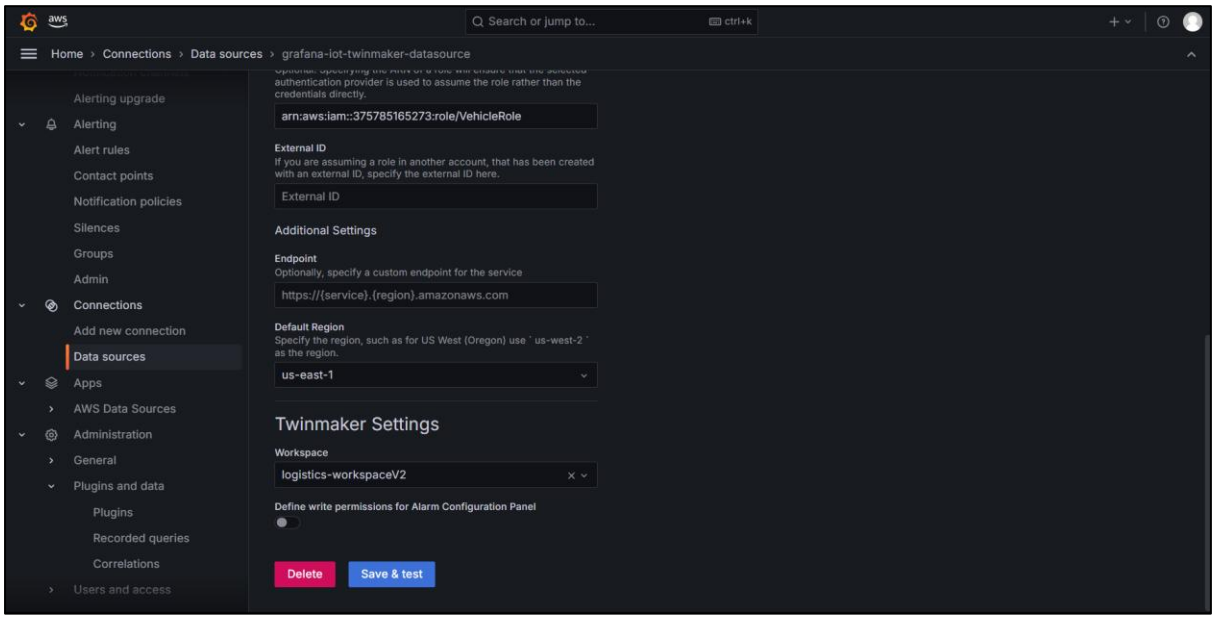
Installed AWS IoT TwinMaker App.



Add datasource AWS IoT TwinMaker adjust the setting to enable the supposed workspace into the Grafana. Choose Access & secret key for the Authentication Process. Input the access key ID and secret access key of the created user in IAM.



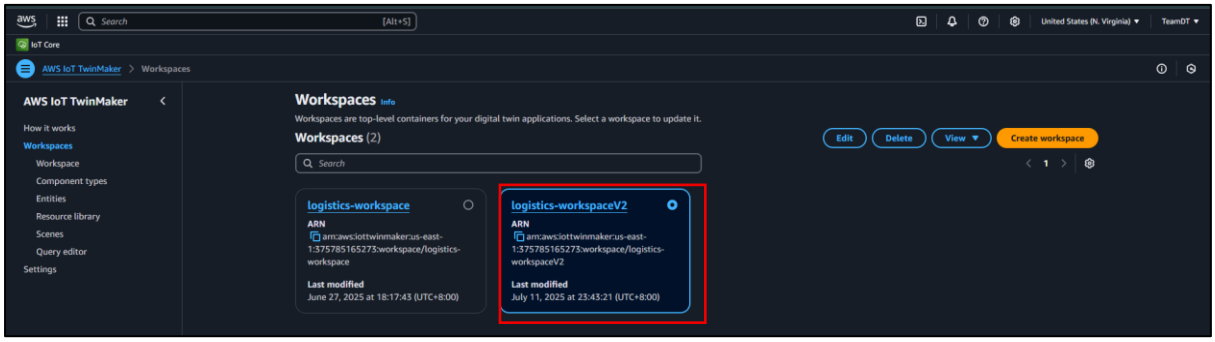
Input the assume Role ARN. Choose the correct region.



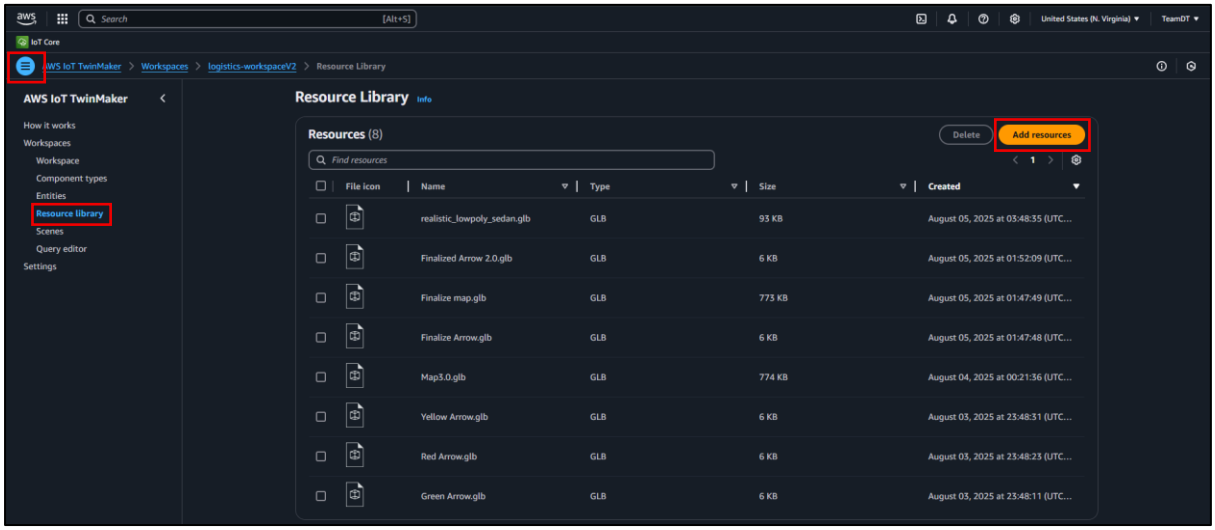
Save & Test first before searching for the workspace. After it is saved and tested, the according workspace will be enabled to the Twinmaker Setting Workspace.

Setting Up AWS IoT TwinMaker Scene

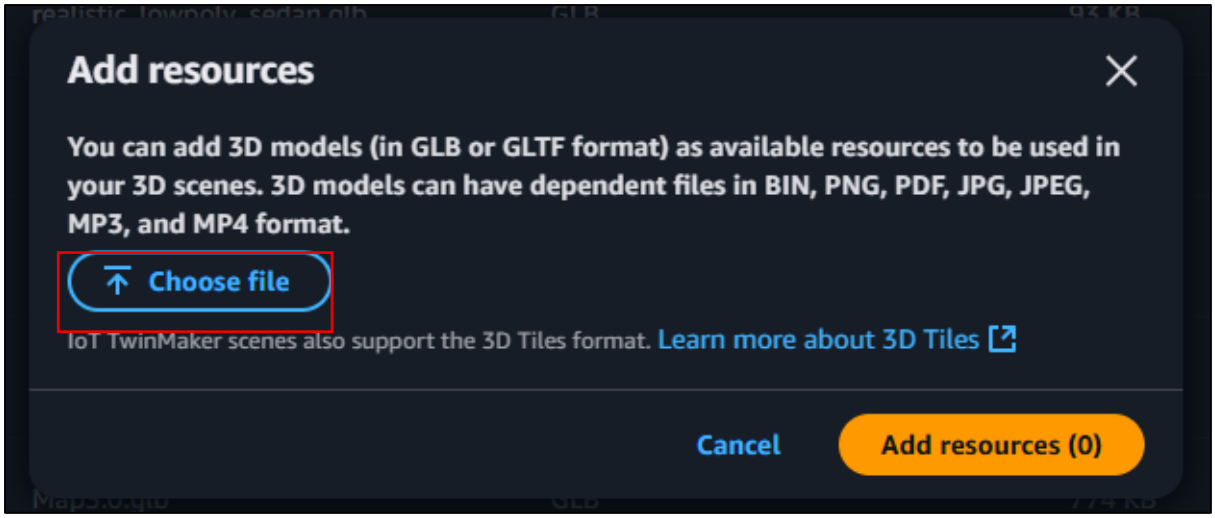
Click the 3-line menu at the top right and select “Workspaces.”



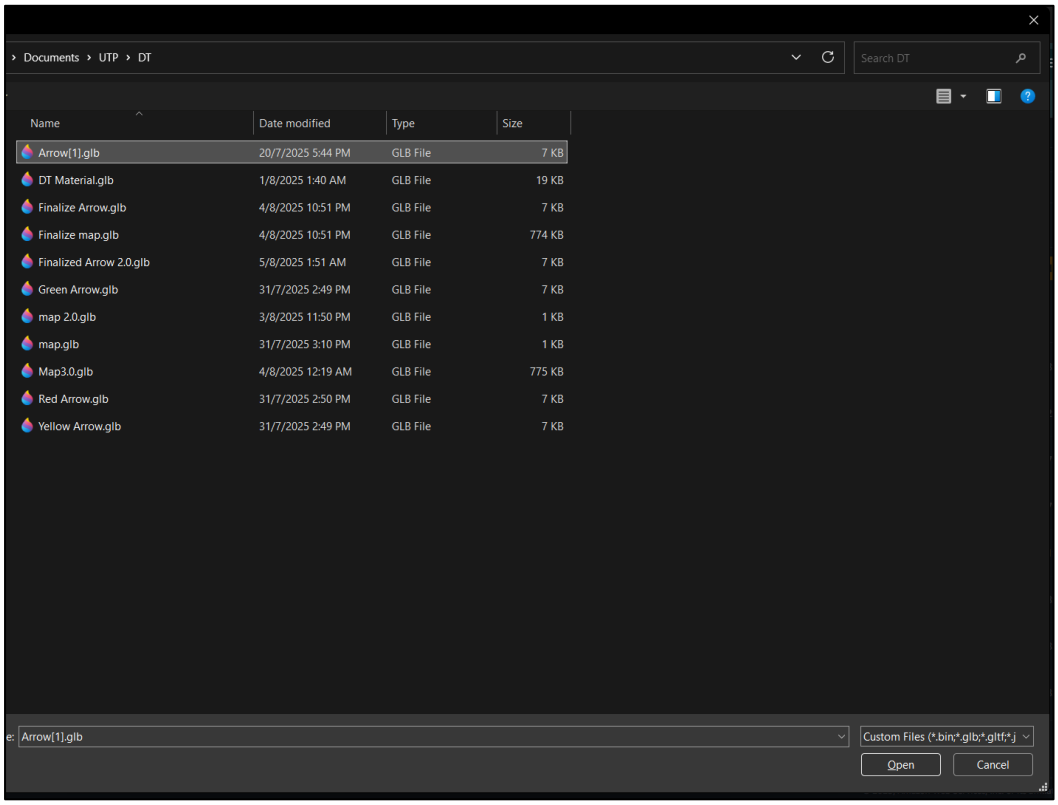
Click on your workspace.



- Click the 3-line menu again and select “Resource Library.”
- To add a 3D model into the resource library, click “Add Resources.”

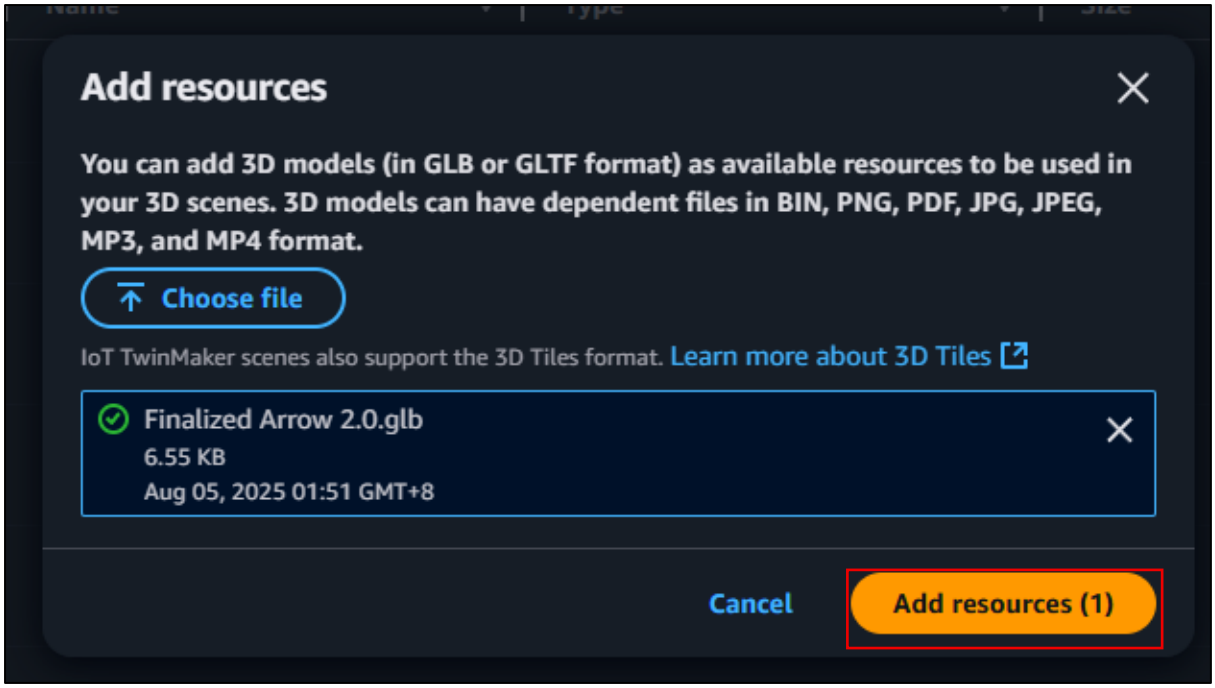


Click “Choose File.”

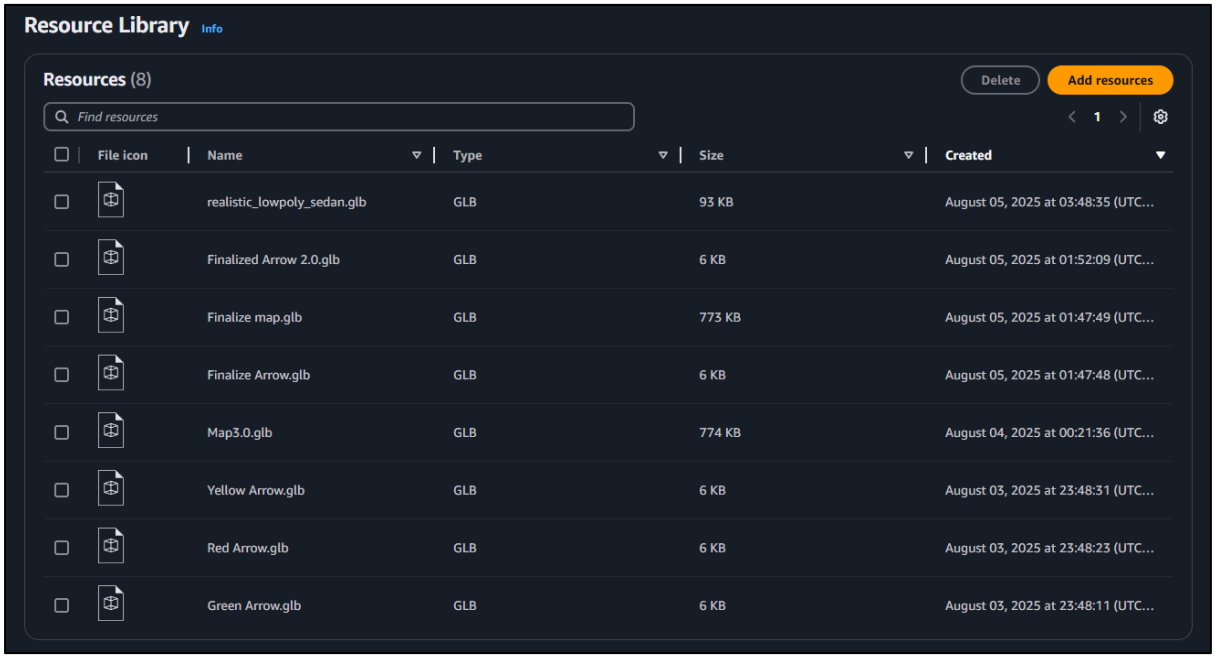


Select the appropriate file for TwinMaker.

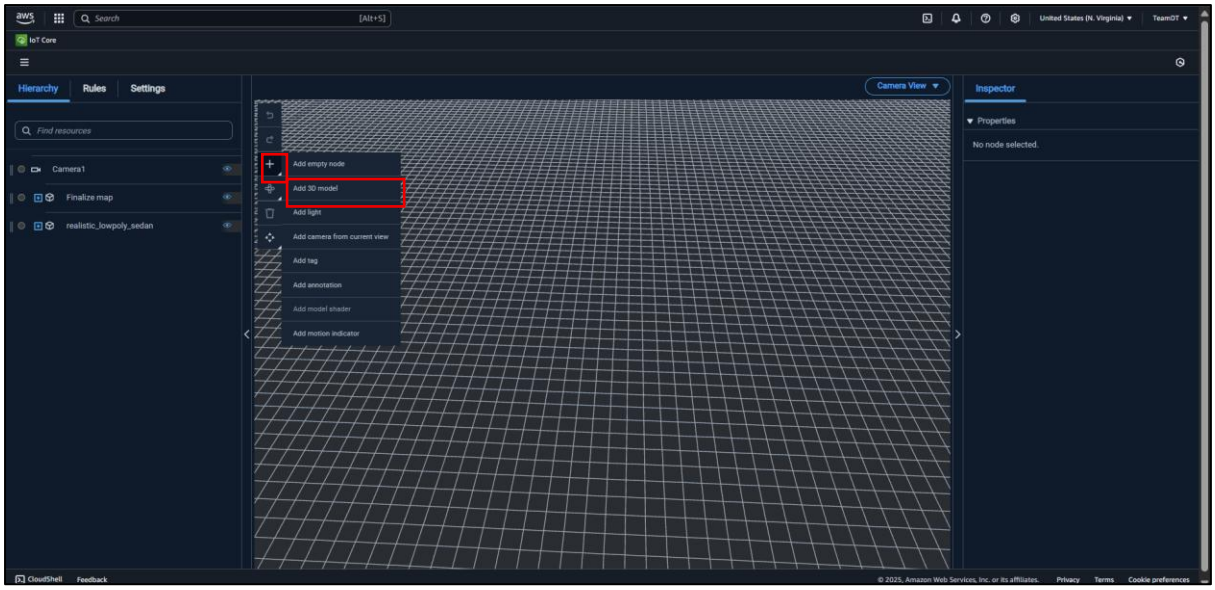
Notes: Ensure that the 3D model is in .glb format.



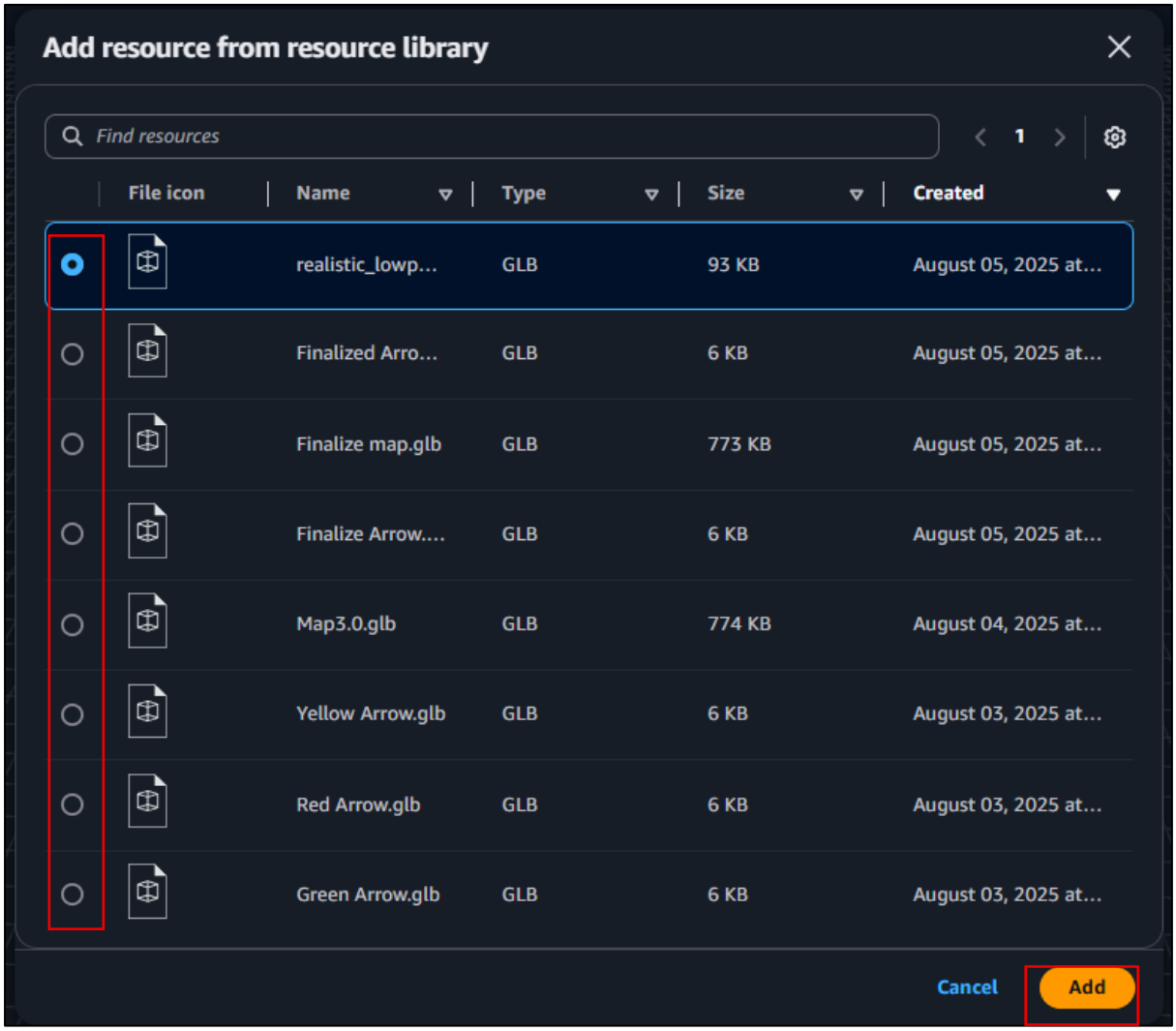
Click “Add Resource.”



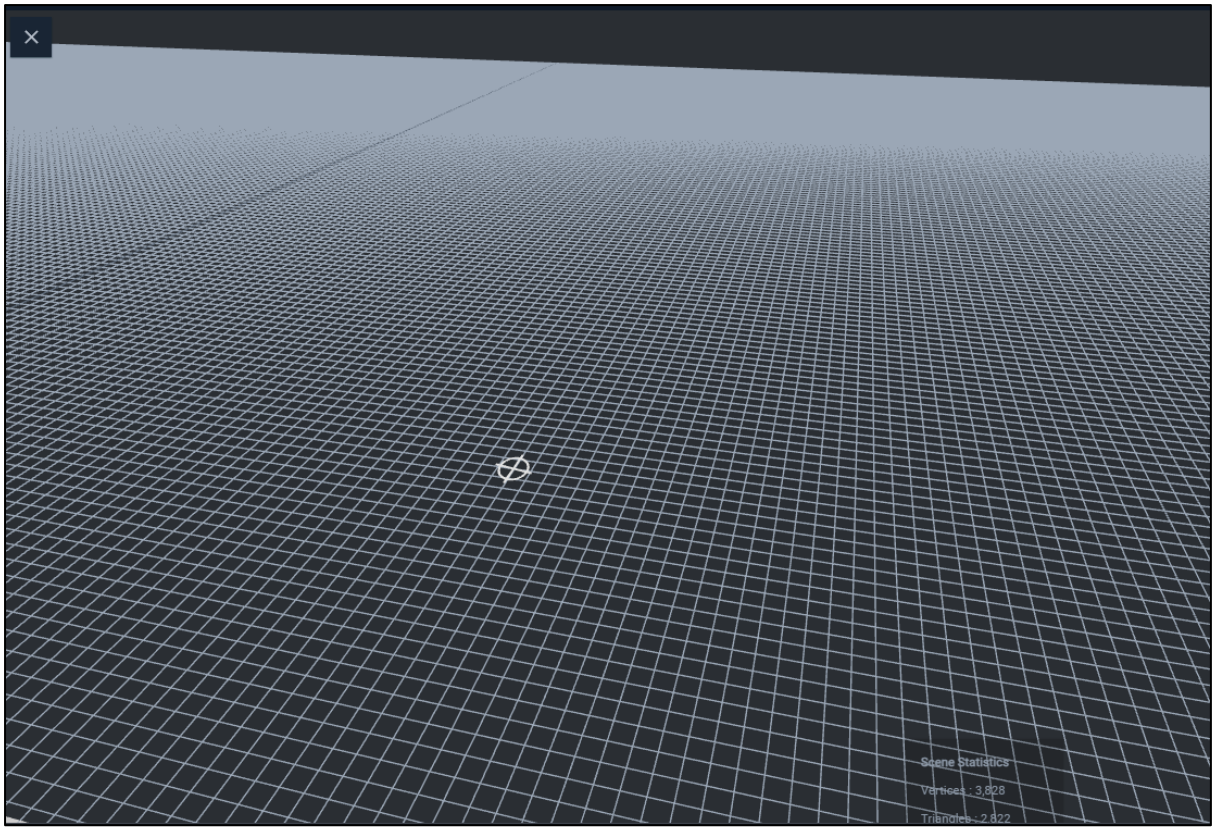
The uploaded file will now appear in the Resource Library.



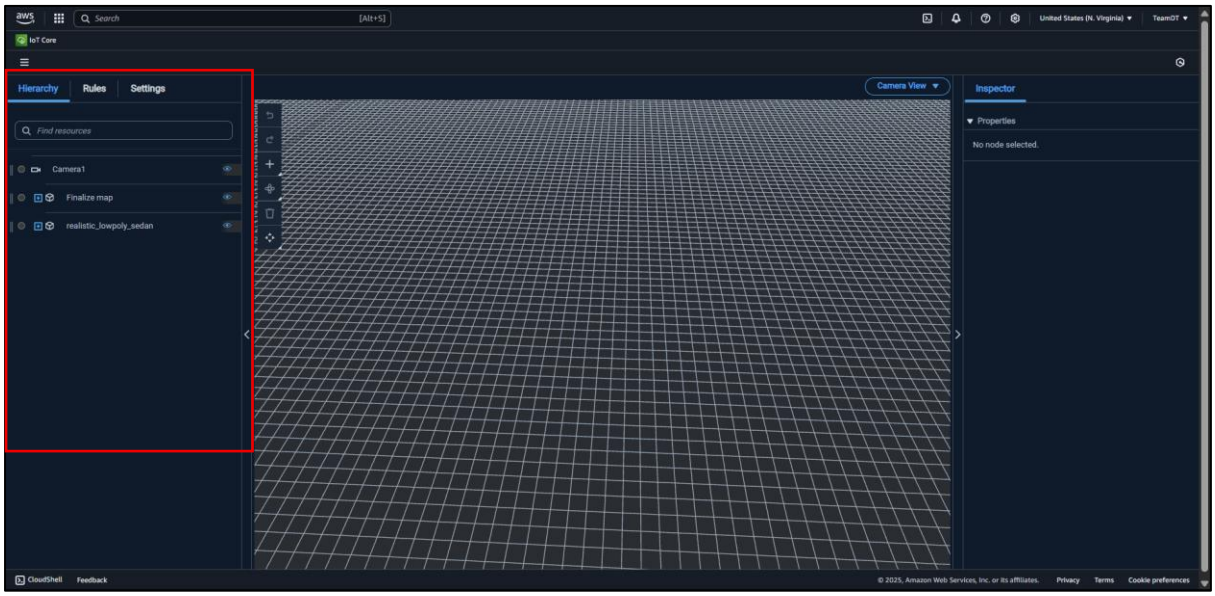
Click the “+” symbol, then choose “Add 3D Model.”



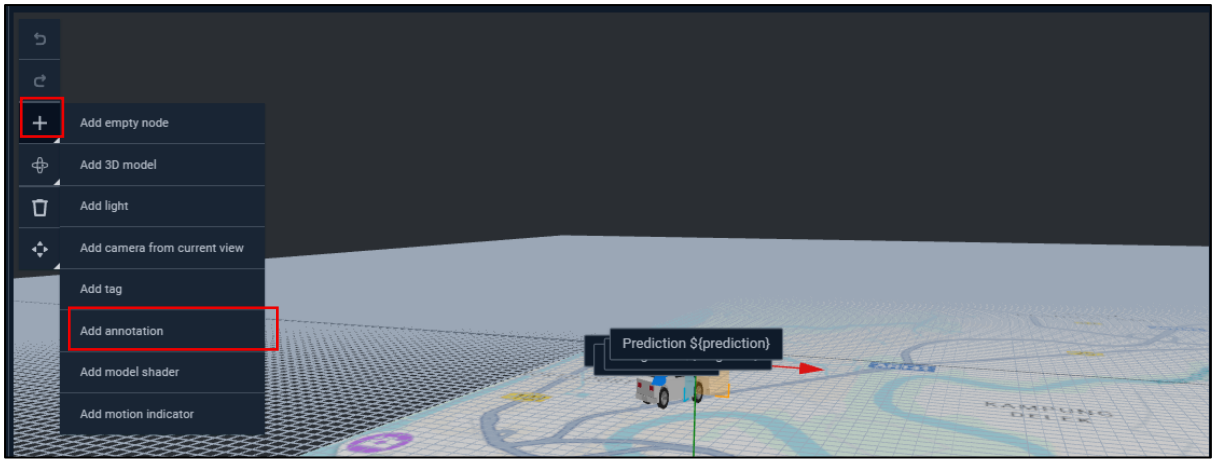
Tick the 3D model to be added to the scene and click “Add” (bottom right of the screen).



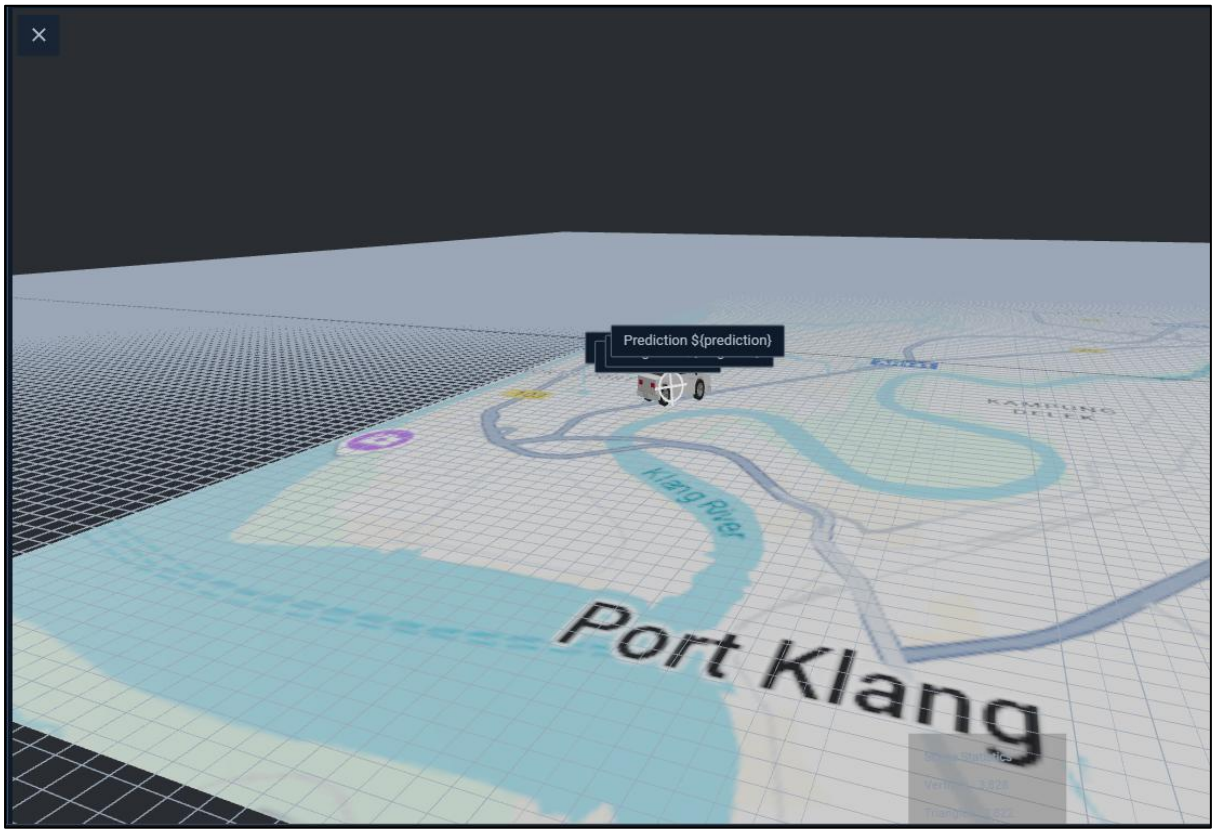
Click anywhere on the screen to place the 3D model.



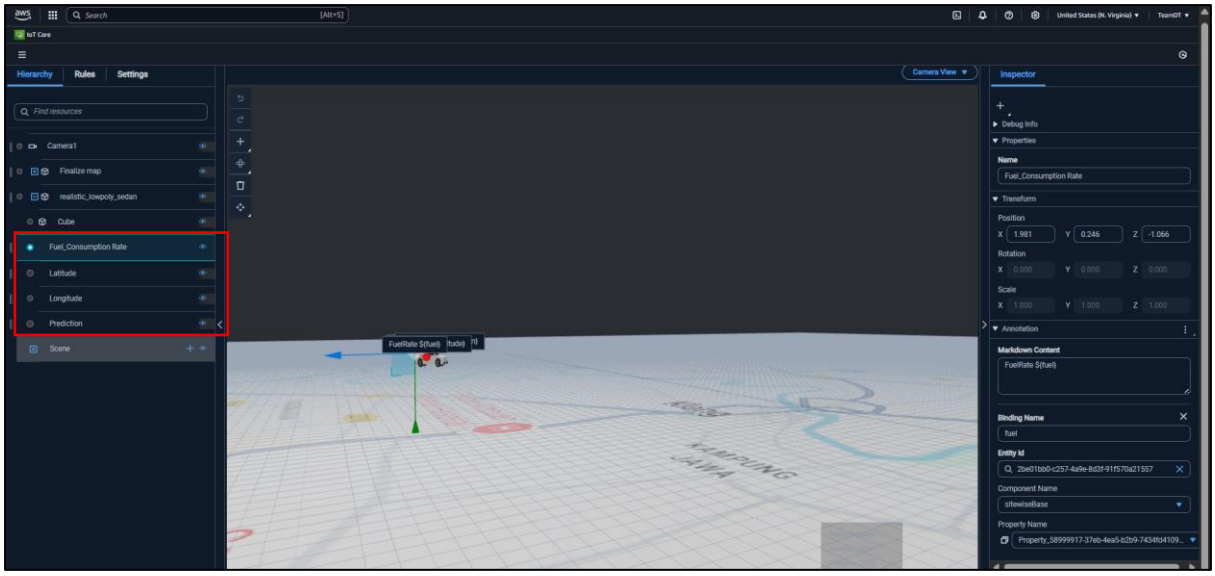
The added model will now appear under the Hierarchy section.



Click the “+” symbol, then choose “Add Annotation.”



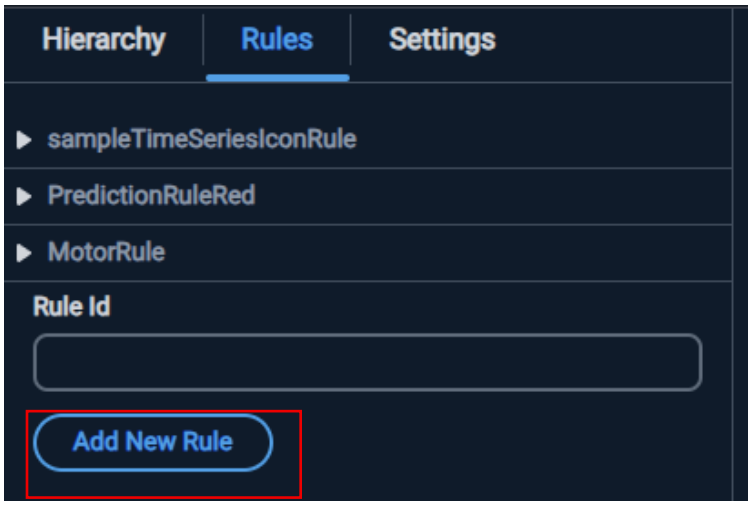
Point the annotation toward the 3D model.



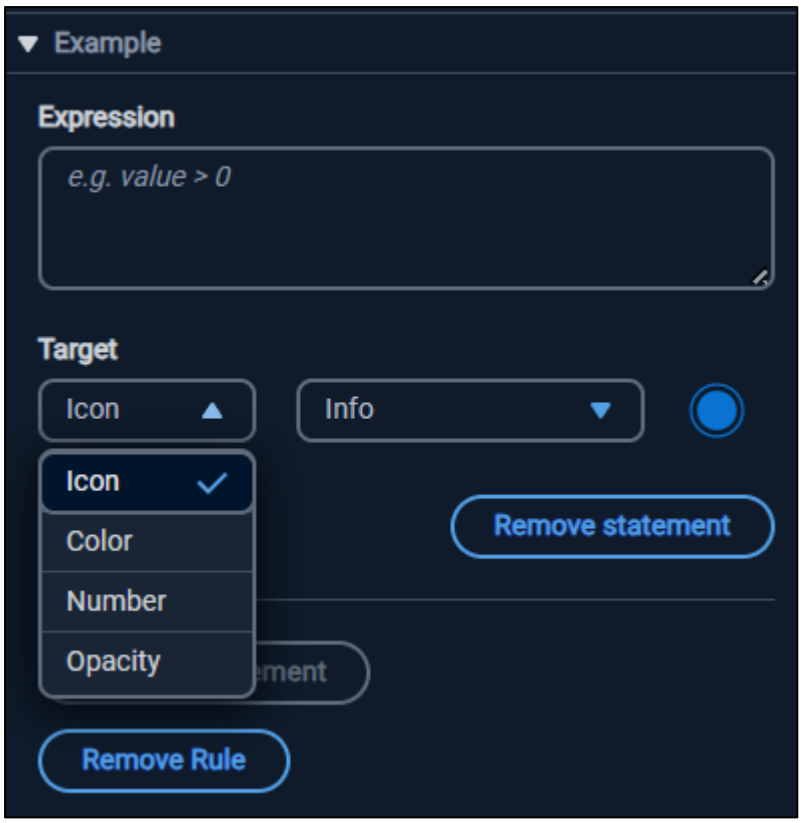
- Repeat Step 13 until you have five annotations in total
- For each annotation, configure the following under the Inspector panel:

Name	Markdown Content	Binding Name	Entity Id	Component Name	Property Name
Fuel_Consumption Rate	FuelRate ${fuel}	fuel	2be01bb0-c257-4a9e-8d3f-91f570a21557	sitewiseBase	fuel_consumption_rate
Latitude	Latitude ${latitude}	latitude			vehicle_gps_latitude
Longitude	Longitude ${longitude}	longitude			vehicle_gps_longitude
Prediction	Prediction ${prediction}	prediction			prediction
Route_Risk_Level	Route_Risk_Levell ${risk}	risk			route_risk_level

Note: Only five data points are shown on-screen due to limited display space.

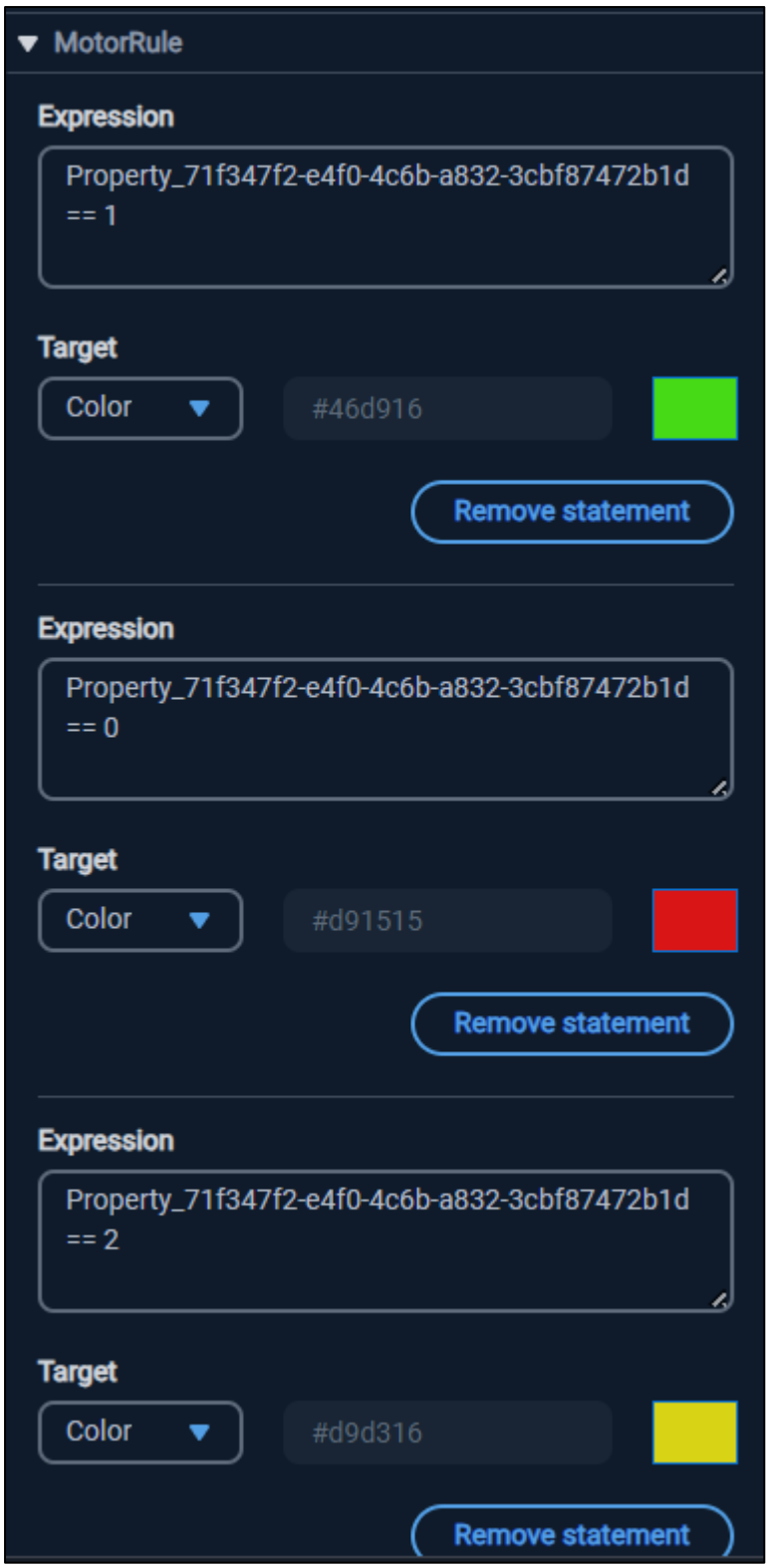


Click on “Rules” and set the Rule ID to MotorRule(created by us) click “Add New Rule.”



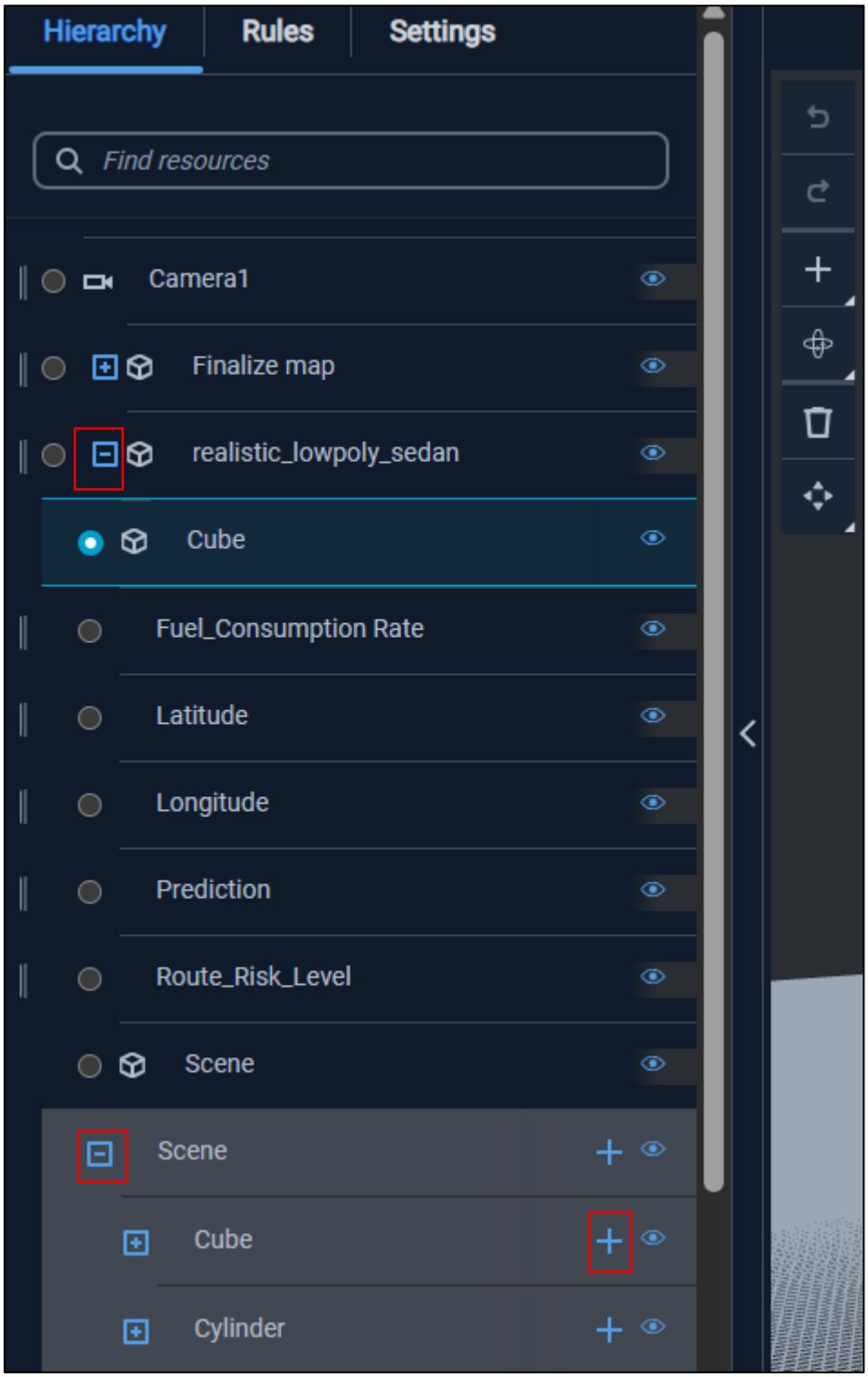
Configure the rule:

- Copy the predictor property ID and paste it inside the Expression field.
- Set Target to Color.
- Click “Add new statement” to define logic conditions:

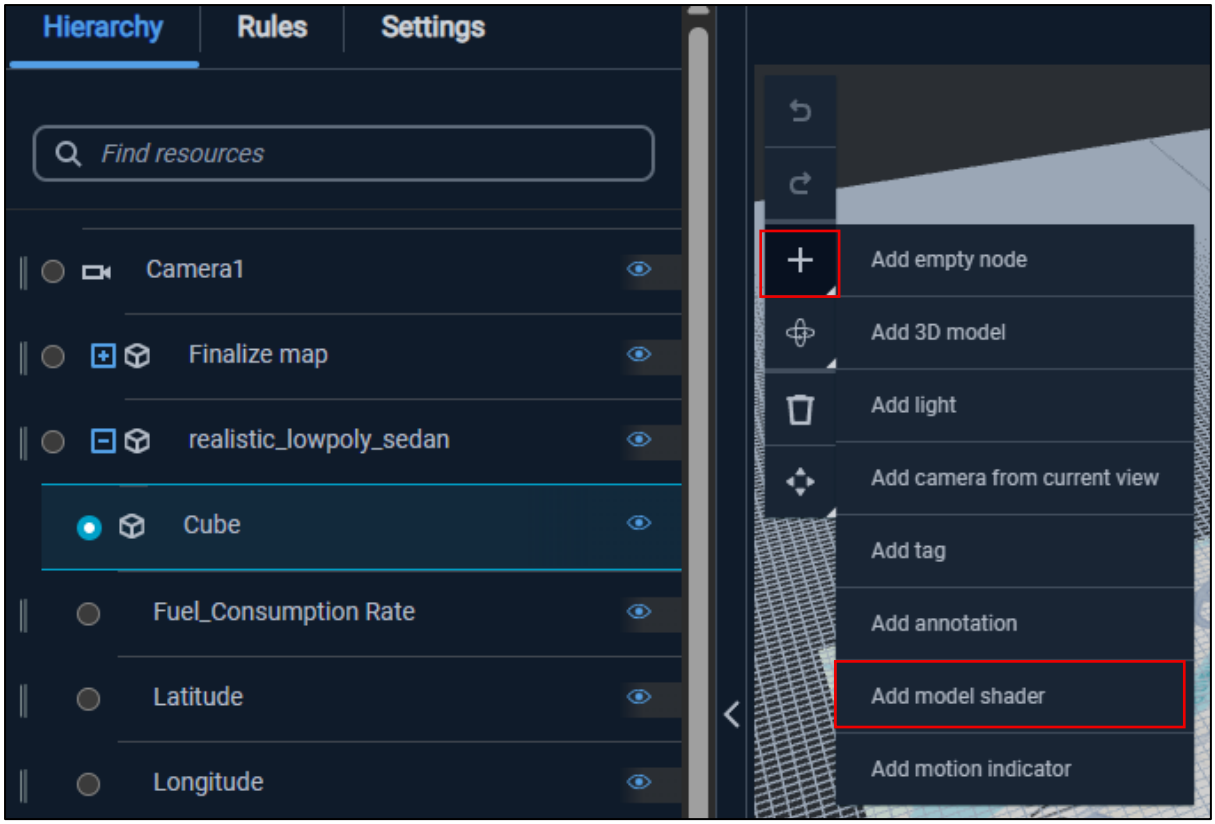


Result:

- Predictor == 1; Green
- Predictor == 0; Red
- Predictor == 2; Yellow

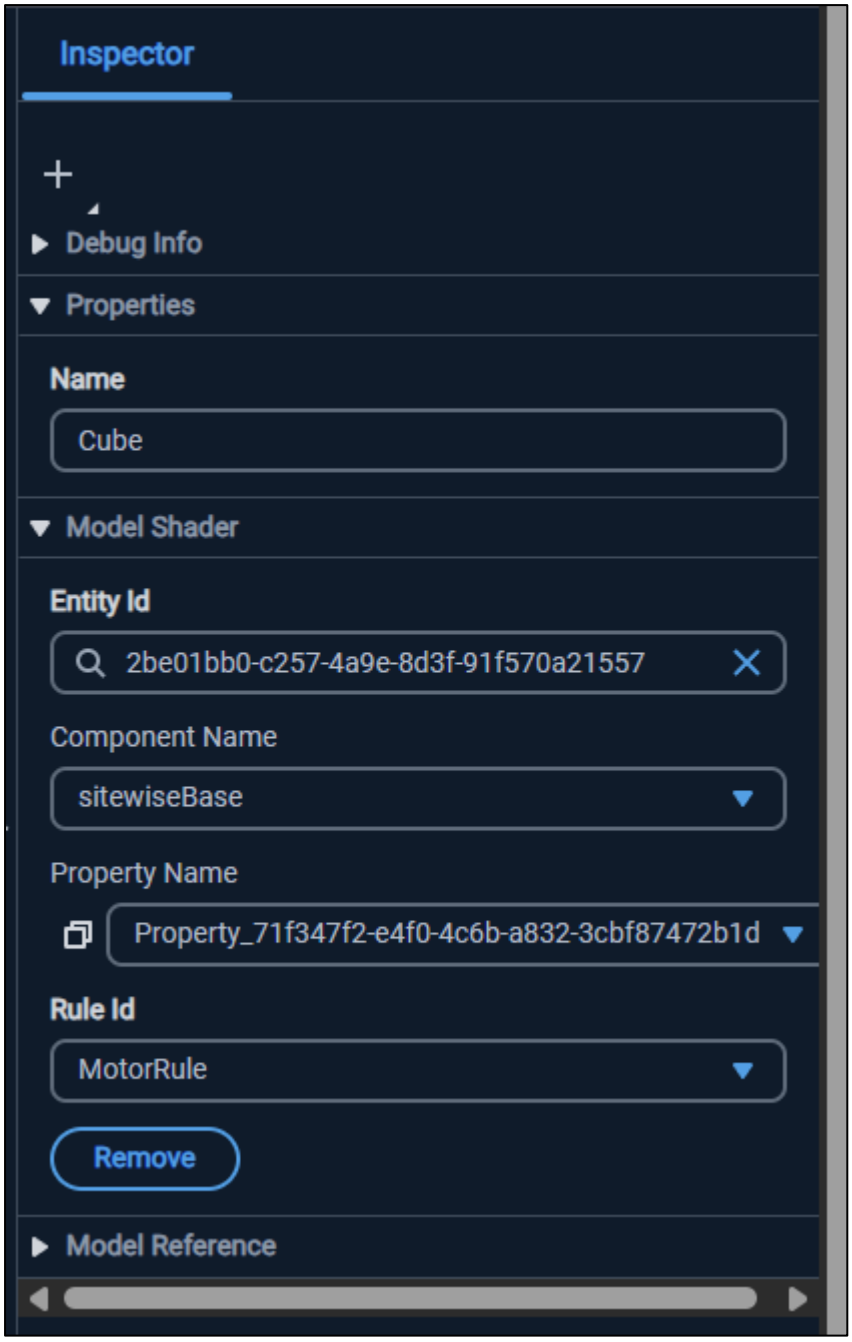


- In the Hierarchy, click the “+” beside realistic_lowpoly_sedan
- Click the icon beside Scene, then click “+” beside Cube.



Select Cube, click the “+” on the scene, and choose “Add Model Shader.”

Note: Cube is used as a marker for shaded areas.

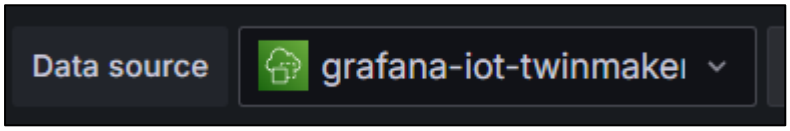


In the Inspector, set the following:

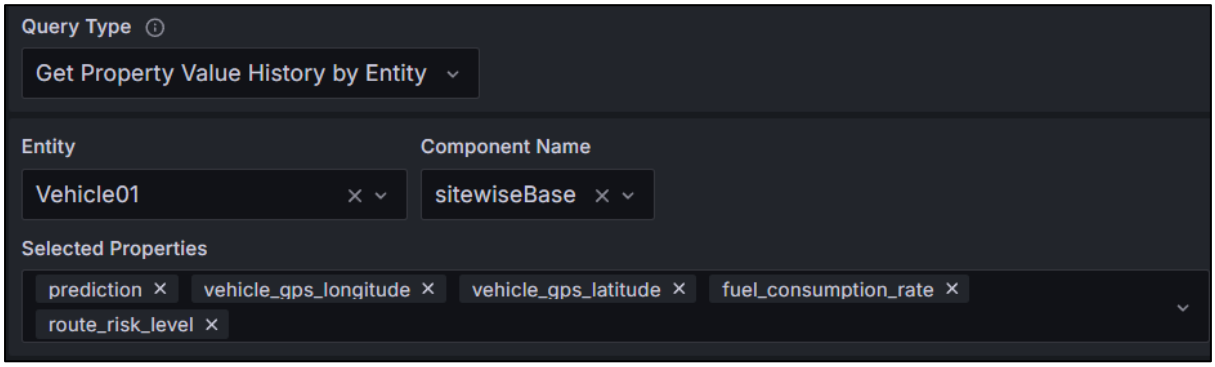
Entity Id	2be01bb0-c257-4a9e-8d3f-91f570a21557
Component Name	sitewiseBase
Property Name	Property_71f347f2-e4f0-4c6b-a832-3cbf87472b1d or prediction
Rule Id	MotorRule

Visualisation

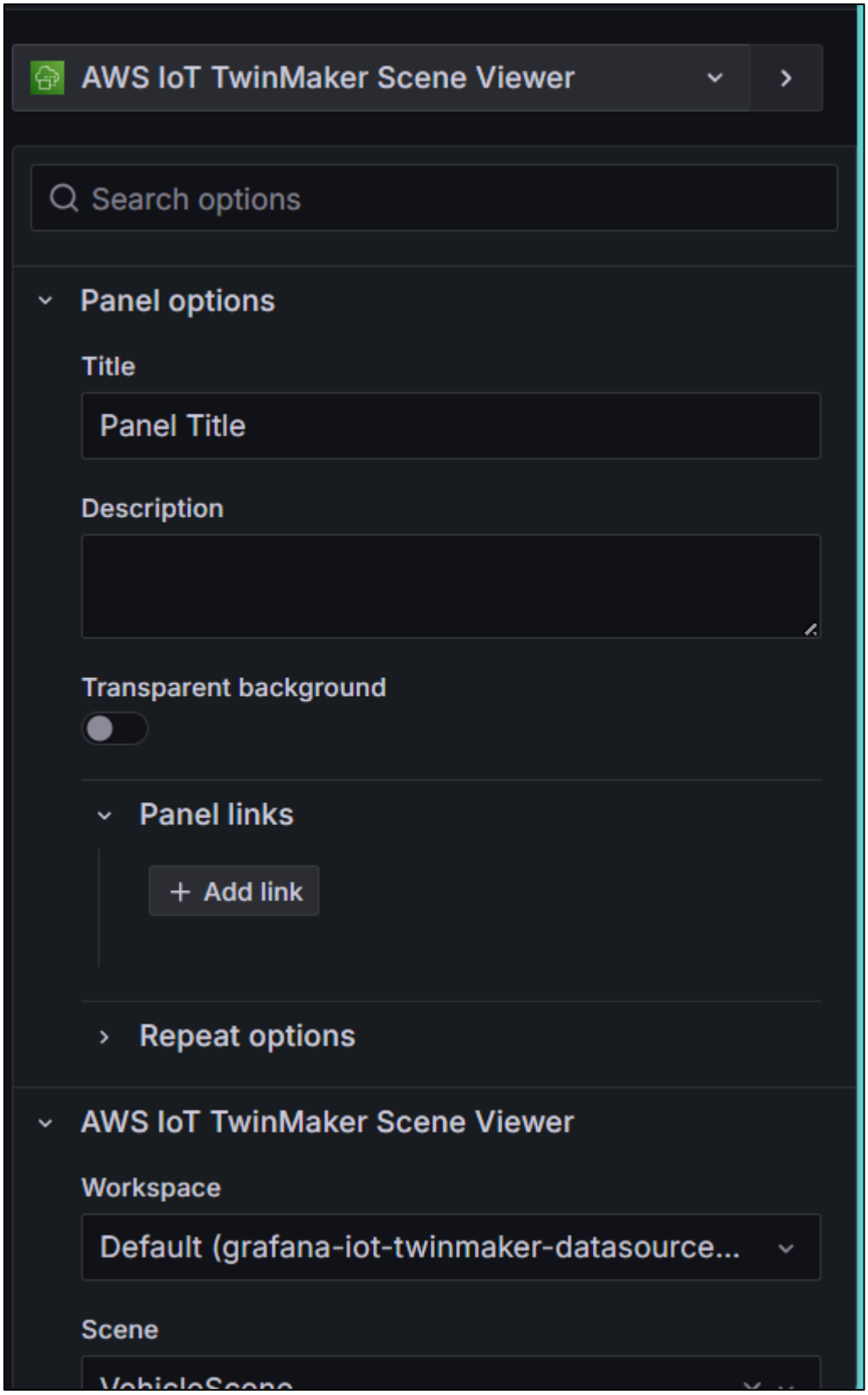
Continuing Visualisation Dashboard for Grafana



Select the twinmaker datasource

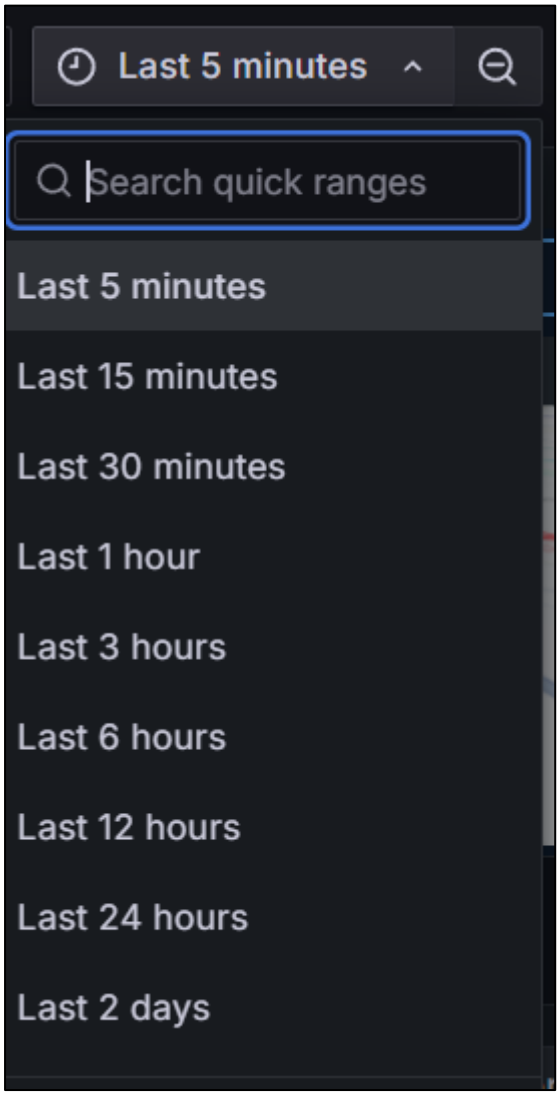


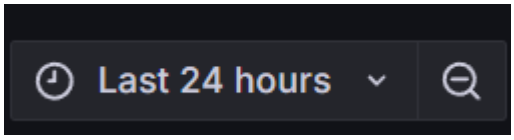
Select the Entity Vehcile01. The component name sitewiseBase. And the properties that we want to show on the scene.



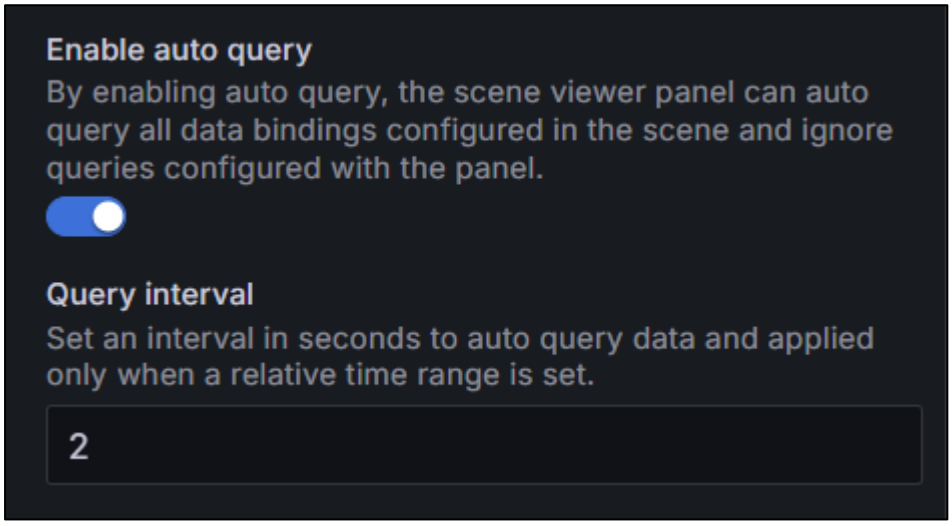
Click on this panel choose the AWS IoT Twinmaker Scene Viewer

Now we can view the scene we created on Twinmaker with the annotations. The data is blank as we haven't sent any data recently. For the earlier test lets try and take those values to see if it updates.

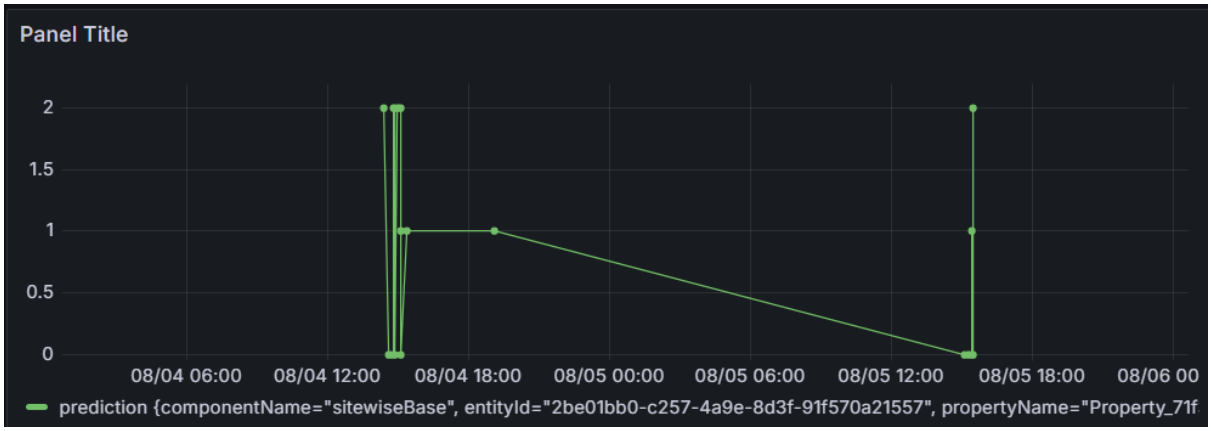




As you see the last data streamed were these values. The Prediction is 2 Thus the model of the car is now Yellow.



We enable auto query and Query interval of 2 seconds. So the values on the scene will update every 2 seconds of being updated in sitewise.



For just the prediction value we can also use a series of other graphs. This is using a time series graph to see all prediction values. As you see its bouncing from 0 to 1 to 2 never in between.

Nows lets try for a measurement we didn't show on the scene.

This is for Traffic congestion level.

CLO2 Deploy Digital Twin AI Model (Inference)

DOCKER

Go to Installing or updating to the latest version of the AWS CLI - AWS Command Line Interface to download docker

Install and update requirements

- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

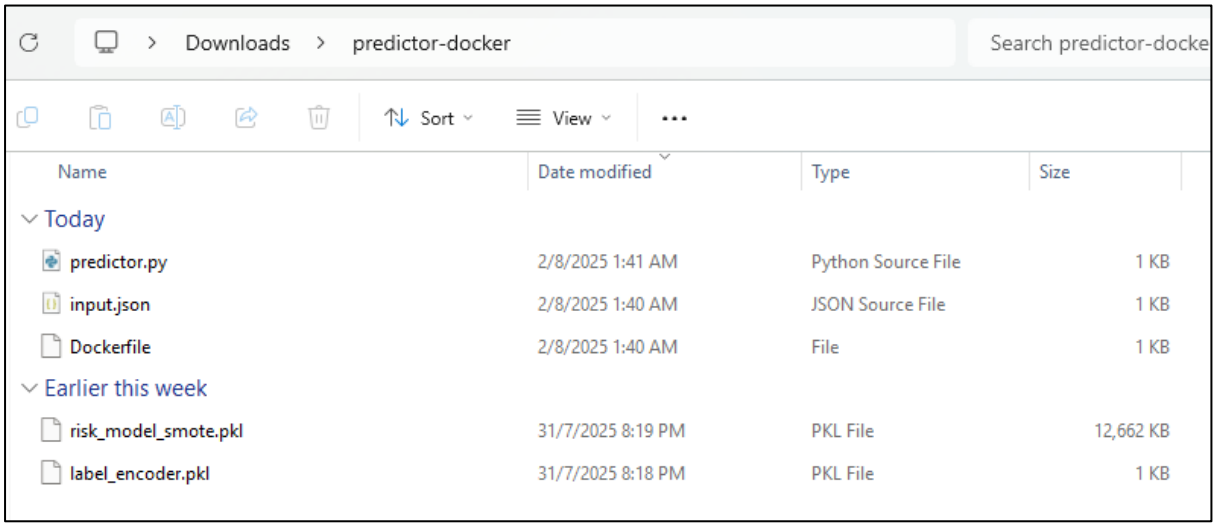
Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the AWS CLI version 2 Changelog on *GitHub*.

1. Download and run the AWS CLI MSI installer for Windows (64-bit):
   https://awscli.amazonaws.com/AWSCLIV2.msi
   Alternatively, you can run the `msiexec` command to run the MSI installer.

After downloading you can create a file on your device

For us it was called predictor-docker



1. Dockerfile

Purpose: Defines how the Docker image is built and what it should run.

What it contains:

- A base image (python:3.10-slim)
- Instructions to copy all necessary files into the container
- A list of required Python packages to install
- A command to execute the prediction script when the container starts

```
FROM python:3.10-slim

WORKDIR /app

# Copy all necessary files
COPY predictor.py .
COPY input.json .
COPY risk_model_smote.pkl .
COPY label_encoder.pkl .

# Install pinned dependencies
RUN pip install joblib numpy pandas scikit-learn==1.6.1

# Run the script on container start
CMD ["python", "predictor.py"]
```

2. predictor.py

Purpose: Loads the model and encoder, reads the input, performs prediction, and prints the result.

What it does:

- Loads the trained model and label encoder from .pkl files
- Loads input features from input.json
- Converts the input to a DataFrame with the correct feature order
- Runs the prediction
- Prints the prediction as a float (0.0, 1.0, or 2.0)

```
import joblib
import json
import pandas as pd

# Load model and encoder
model = joblib.load("risk_model_smote.pkl")
encoder = joblib.load("label_encoder.pkl")

# Load input data
with open("input.json") as f:
    input_data = json.load(f)

# Convert to DataFrame to preserve feature names
X = pd.DataFrame([input_data])[model.feature_names_in_]

# Predict risk class as double
pred = model.predict(X)[0]
print(float(pred))
```

3. input.json

Purpose: A test input file containing values for all 25 features.

Format: A JSON dictionary matching the model's expected input features by name.

4. risk_model_smote.pkl

Purpose: The trained RandomForestClassifier model saved using joblib.

Details:

- Trained using your logistics dataset
- Balanced using SMOTE to handle class imbalance
- Predicts one of three risk classes (0, 1, 2)

5. label_encoder.pkl

Purpose: A LabelEncoder object used to encode and decode risk classification labels.

Details:

- Maps class names (High Risk, Low Risk, Moderate Risk) to numbers (0, 1, 2)
- Used during training and prediction to match encoded labels

How it Works Together

1. The Dockerfile builds an image that contains Python, your model, input data, and script.
2. When you run the container, it executes predictor.py.
3. The script loads the model and input, runs the prediction, and prints the numeric output.
4. You can replace input.json for each new prediction or extend the script to take real-time data.

   Due to AWS Lambda's size limitations—50 MB for zipped deployment packages and 250 MB unzipped when using layers—we faced issues deploying our trained model (risk_model_smote.pkl) and its required Python dependencies directly. The model and libraries (defined in requirements.txt) exceeded these size thresholds.

To resolve this, we adopted the container image deployment approach for AWS Lambda. This method allows packaging the entire application—including large models, dependencies, and runtime code—into a single Docker image, which can be up to 10 GB in size.

We used the official AWS Lambda base image for Python 3.11 (public.ecr.aws/lambda/python:3.11) as the foundation. Our Dockerfile installed all necessary packages and included the model and inference code. After building the image locally, we pushed it to Amazon Elastic Container Registry (Amazon ECR), which is a fully managed Docker container registry by AWS.

Once stored in ECR, the container image was referenced by the Lambda function. This setup allows Lambda to pull the image at runtime and execute our model inference code without the traditional size constraints.

Summary

- Problem: Model + dependencies too large for regular Lambda deployment
- Solution: Package as Docker image using AWS Lambda container support
- Storage: Push image to Amazon ECR for managed, scalable access
- Runtime: Lambda pulls and runs the image without modifying function code

Introducing the new Amazon ECR console experience

We updated the console navigation. Amazon ECR public and private now have separate, dedicated sections in the console. As you navigate this updated console experience, make sure to update any bookmarks that you use.

Private repositories

Public repositories

Create private repository

General settings

Repository name

Enter a concise name. Repositories support namespaces, which you can use to group similar repositories.

375785165273.dkr.ecr.us-east-1.amazonaws.com/ predictor-lambda

16 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, and special characters ._-/.

Encryption settings Info

⚠ The encryption settings for a repository can't be changed once the repository is created.

Encryption configuration

By default, repositories use the industry standard Advanced Encryption Standard (AES) encryption. You can optionally choose to use a key stored in the AWS Key Management Service (KMS) to encrypt the images in your repository.

- (●) AES-256
  Industry standard Advanced Encryption Standard (AES) encryption
- ( ) AWS KMS
  AWS Key Management Service (KMS)

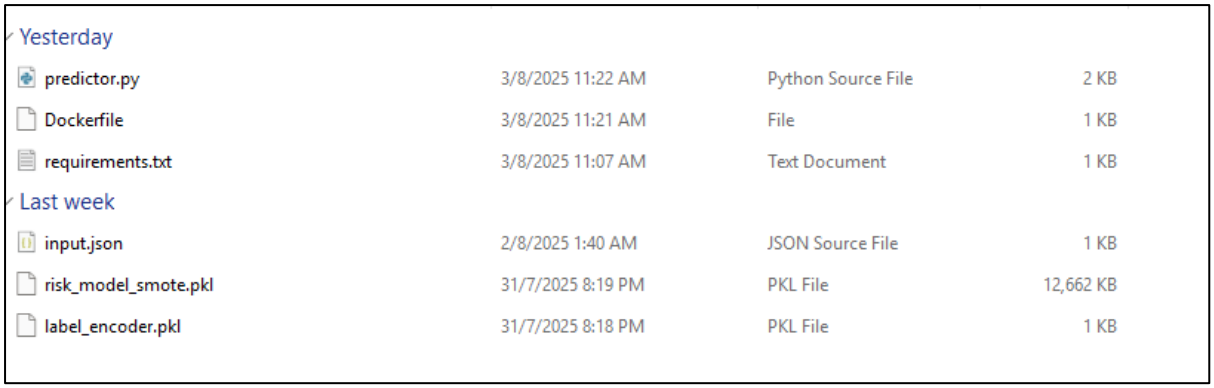
✓ Successfully created predictor-lambda ✕

Private repositories (1)

View push commands | Delete | Actions | Create repository

Search by repository substring

Repository name	URI	Created at	Tag immutability	Encryption type
predictor-lambda	375785165273.dkr.ecr.us-east-1.amazonaws.com/predictor-lambda	August 02, 2025, 23:46:55 (UTC+08)	Mutable	AES-256



This is the improved predictor-docker

```
PS C:\Users\Deads\Downloads\predictor-docker> docker buildx create --use
zen_hellman
PS C:\Users\Deads\Downloads\predictor-docker> docker buildx build --platform linux/amd64 -t predictor-lambda . --load
[+] Building 497.3s (13/13) FINISHED                                                    docker-container:zen_h
ellman
```

```
PS C:\Users\Deads\Downloads\predictor-docker> docker tag predictor-lambda:latest 375785165273.dkr.ecr.us-east-1.amazonaw
s.com/predictor-lambda:latest
PS C:\Users\Deads\Downloads\predictor-docker> aws ecr get-login-password --region us-east-1 | docker login --username AW
S --password-stdin 375785165273.dkr.ecr.us-east-1.amazonaws.com
Login Succeeded
PS C:\Users\Deads\Downloads\predictor-docker> docker push 375785165273.dkr.ecr.us-east-1.amazonaws.com/predictor-lambda:
latest
The push refers to repository [375785165273.dkr.ecr.us-east-1.amazonaws.com/predictor-lambda]
7a8b45083a3d: Pushed
2866816e2aca: Layer already exists
94a631ddf6b1: Pushed
879e7a058eb9: Pushed
96fc691bef2d: Pushed
c8b0a55813ef: Layer already exists
9b561df83c47: Layer already exists
5632717bd051: Layer already exists
05480f6c0a23: Layer already exists
c46918a4cdb7: Pushed
020e547580a2: Layer already exists
latest: digest: sha256:1c90d18fd738af8ca3a2835ce3cbdbd585ba4f946c69107690cc75564a37aa5d size: 2387
PS C:\Users\Deads\Downloads\predictor-docker>
```

To create the container image, we first defined a Dockerfile that used the official AWS Lambda Python 3.11 base image as its foundation. In this file, we specified all necessary dependencies by referencing a requirements.txt file and included our inference script along with the pre-trained model file (risk_model_smote.pkl). We also set the container's entry point to the appropriate handler function so that AWS Lambda would know how to invoke the model prediction logic upon execution.

After preparing the Dockerfile, we built the container image locally using the Docker CLI. Once the image was successfully built, we tagged it and pushed it to Amazon Elastic Container Registry (ECR)—a secure, fully managed Docker image repository provided by AWS. After the image was uploaded to ECR, we configured our Lambda function to reference it directly from the registry. This setup allows the Lambda function to pull the container image at runtime, enabling us to run our model inference logic reliably and without being constrained by Lambda's standard package size limits.

This is the predictor.py

```
import json

import joblib

import numpy as np

import boto3

import time

import os



# Load model and encoder once (reused across invocations)

model = joblib.load("/var/task/model/risk_model_smote.pkl")

label_encoder = joblib.load("/var/task/model/label_encoder.pkl")



sitewise = boto3.client("iotsitewise", region_name="us-east-1")



def lambda_handler(event, context):

    try:

        fixed = event["fixed"]

        periodic = event["periodic"]

        realtime_rows = event["realtime"]  # List of 6 dicts



        # Compute average of real-time values
```

```
realtime_avg = {
    k: float(np.mean([float(row[k]) for row in realtime_rows if k in row]))
    for k in realtime_rows[0]
}

# Merge all inputs
features = {}
features.update(fixed)
features.update(realtime_avg)
features.update(periodic)

# Format input for model
X = [features[k] for k in model.feature_names_in_]
prediction = model.predict([X])[0]
predicted_class = int(prediction)

# Send to SiteWise
timestamp = int(time.time())
sitewise.batch_put_asset_property_value(entries=[
    {
        "entryId": f"prediction_{timestamp}",
```

```
            "propertyAlias": "/vehicle/Vehicle01/prediction",
            "propertyValues": [{
                "value": {"doubleValue": predicted_class},
                "timestamp": {"timeInSeconds": timestamp},
                "quality": "GOOD"
            }
        ])
```

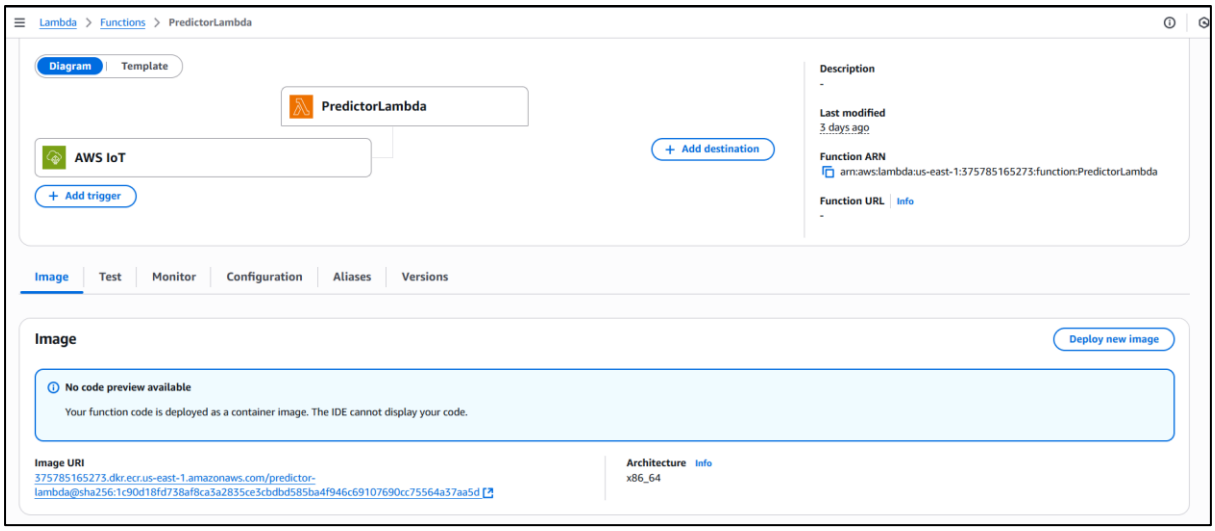
This is the requirements

```
numpy==2.0.2
scikit-learn==1.6.1
joblib==1.5.1
pandas==2.2.2
boto3
```

This is the Docker File.

```
FROM public.ecr.aws/lambda/python:3.11

# Install dependencies

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copy code and model files

COPY predictor.py .

COPY risk_model_smote.pkl ./model/

COPY label_encoder.pkl ./model/
```

```
# Lambda handler

CMD ["predictor.lambda_handler"]
```



This is the second lambda. It receives the trigger and runs the model to give the prediction based on the data streamed. After that it sends the prediction value to Sitewise which is then connected to Twinmaker.

To run the data stream continuously there were changes made to **mqttpublisher2.py**

```
import csv

import time

import ssl

import json

import paho.mqtt.client as mqtt

from dotenv import load_dotenv

import os

from datetime import datetime, timedelta
```

```
# Load environment variables

load_dotenv()



# AWS IoT Core connection settings

MQTT_ENDPOINT = "ackvac20gj6g5-ats.iot.us-east-1.amazonaws.com"

PORT = 8883

CA_CERT = "AmazonRootCA1.pem"

DEVICE_CERT = "cert.pem.crt"

PRIVATE_KEY = "private.pem.key"

TOPIC_PREFIX = "/vehicle/Vehicle01"



# MQTT client setup

client = mqtt.Client(protocol=mqtt.MQTTv311)

client.tls_set(ca_certs=CA_CERT,

               certfile=DEVICE_CERT,

               keyfile=PRIVATE_KEY,

               tls_version=ssl.PROTOCOL_TLSv1_2)



print("Connecting to MQTT broker...")

client.connect(MQTT_ENDPOINT, PORT, keepalive=60)

client.loop_start()
```

```
# Simulated start time
simulated_time = datetime.now()


# === Load CSV and clean ===
with open('logistics_dataset.csv', 'r', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    rows = list(reader)


if not rows:
    print("No data found in CSV.")
    exit()


# Clean keys/values
cleaned_rows = []
for row in rows:
    if row:
        cleaned_rows.append({k.strip(): v.strip() for k, v in row.items() if k and v})


# === Send Fixed Features from the First Row ===
fixed = cleaned_rows[0]
```

```
fixed_payloads = {
    f"{TOPIC_PREFIX}/lead_time_days": float(fixed["lead_time_days"]),
    f"{TOPIC_PREFIX}/supplier_reliability_score": float(fixed["supplier_reliability_score"]),
    f"{TOPIC_PREFIX}/port_congestion_level": float(fixed["port_congestion_level"]),
                                            f"{TOPIC_PREFIX}/weather_condition_severity":
float(fixed["weather_condition_severity"]),
    f"{TOPIC_PREFIX}/route_risk_level": float(fixed["route_risk_level"]),
                                          f"{TOPIC_PREFIX}/handling_equipment_availability":
float(fixed["handling_equipment_availability"]),
    f"{TOPIC_PREFIX}/historical_demand": float(fixed["historical_demand"]),
    f"{TOPIC_PREFIX}/driver_behavior_score": float(fixed["driver_behavior_score"]),
                                            f"{TOPIC_PREFIX}/warehouse_inventory_level":
float(fixed["warehouse_inventory_level"]),
    f"{TOPIC_PREFIX}/loading_unloading_time": float(fixed["loading_unloading_time"])
}


print("\nSending fixed features (once at start):")
for topic, value in fixed_payloads.items():
    message = json.dumps({"value": value})
    client.publish(topic, message)
    print(f"Published to {topic}: {message}")


# === Real-time and Periodic loop (infinite) ===
```

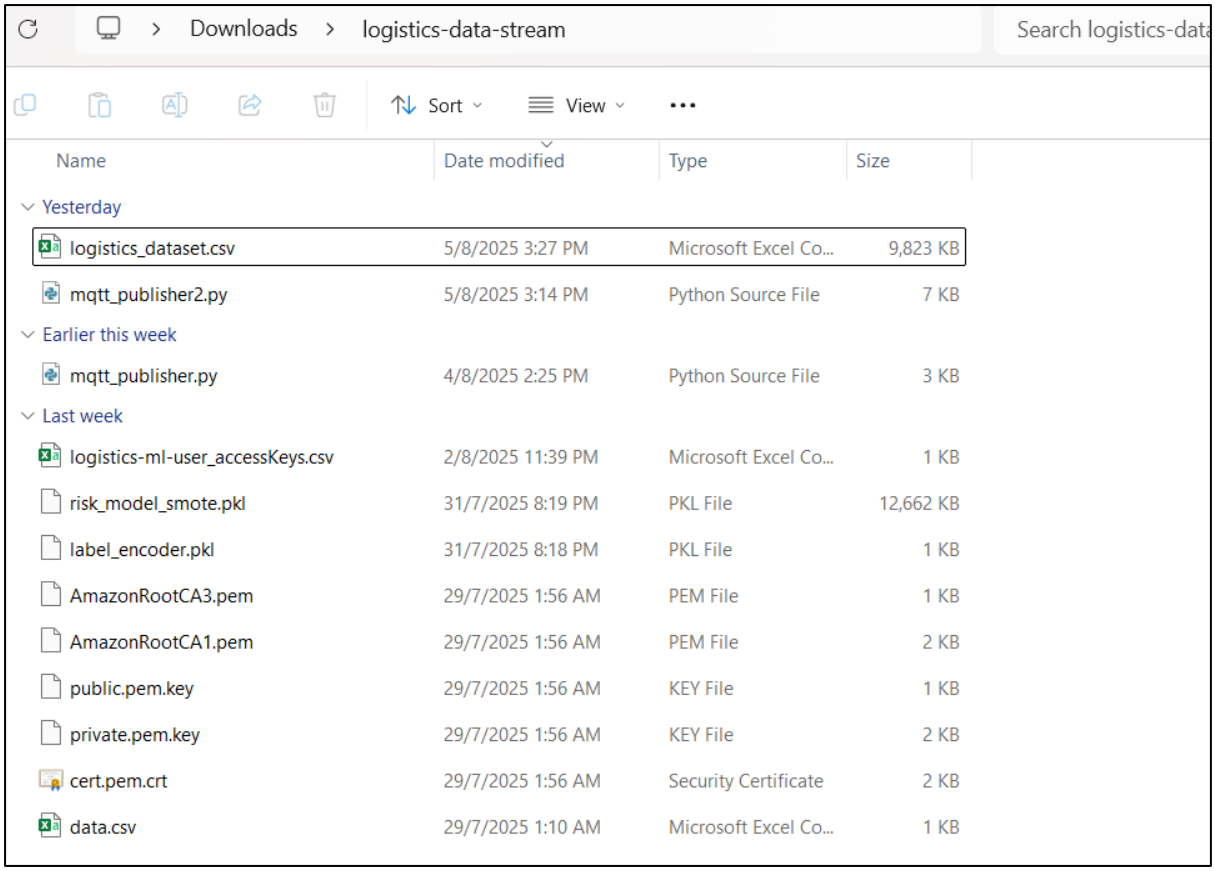
```
row_count = 0

realtime_history = []

print("\nStarting real-time + periodic data stream (infinite loop):")



try:

    while True:

        # Cycle through cleaned rows (excluding first fixed row)

        row = cleaned_rows[(row_count % (len(cleaned_rows) - 1)) + 1]



        trip_duration_seconds = (row_count + 1) * 10



        # Real-time features

        realtime_payloads = {

            f"{TOPIC_PREFIX}/vehicle_gps_latitude": float(row["vehicle_gps_latitude"]),

            f"{TOPIC_PREFIX}/vehicle_gps_longitude": float(row["vehicle_gps_longitude"]),

            f"{TOPIC_PREFIX}/fuel_consumption_rate": float(row["fuel_consumption_rate"]),

           f"{TOPIC_PREFIX}/traffic_congestion_level": float(row["traffic_congestion_level"]),

            f"{TOPIC_PREFIX}/iot_temperature": float(row["iot_temperature"]),

            f"{TOPIC_PREFIX}/cargo_condition_status": float(row["cargo_condition_status"]),

            f"{TOPIC_PREFIX}/trip_duration": float(trip_duration_seconds)

        }
```

```
print(f"\nSimulated time: {simulated_time.strftime('%Y-%m-%d %H:%M:%S')}")
for topic, value in realtime_payloads.items():
    message = json.dumps({"value": value})
    client.publish(topic, message)
    print(f"Published to {topic}: {message}")

# Store for averaging
realtime_row_clean = {
    "vehicle_gps_latitude": float(row["vehicle_gps_latitude"]),
    "vehicle_gps_longitude": float(row["vehicle_gps_longitude"]),
    "fuel_consumption_rate": float(row["fuel_consumption_rate"]),
    "traffic_congestion_level": float(row["traffic_congestion_level"]),
    "iot_temperature": float(row["iot_temperature"]),
    "cargo_condition_status": float(row["cargo_condition_status"]),
    "trip_duration": float(trip_duration_seconds)
}
realtime_history.append(realtime_row_clean)

# Every 6 rows → send periodic + trigger
if (row_count + 1) % 6 == 0:
```

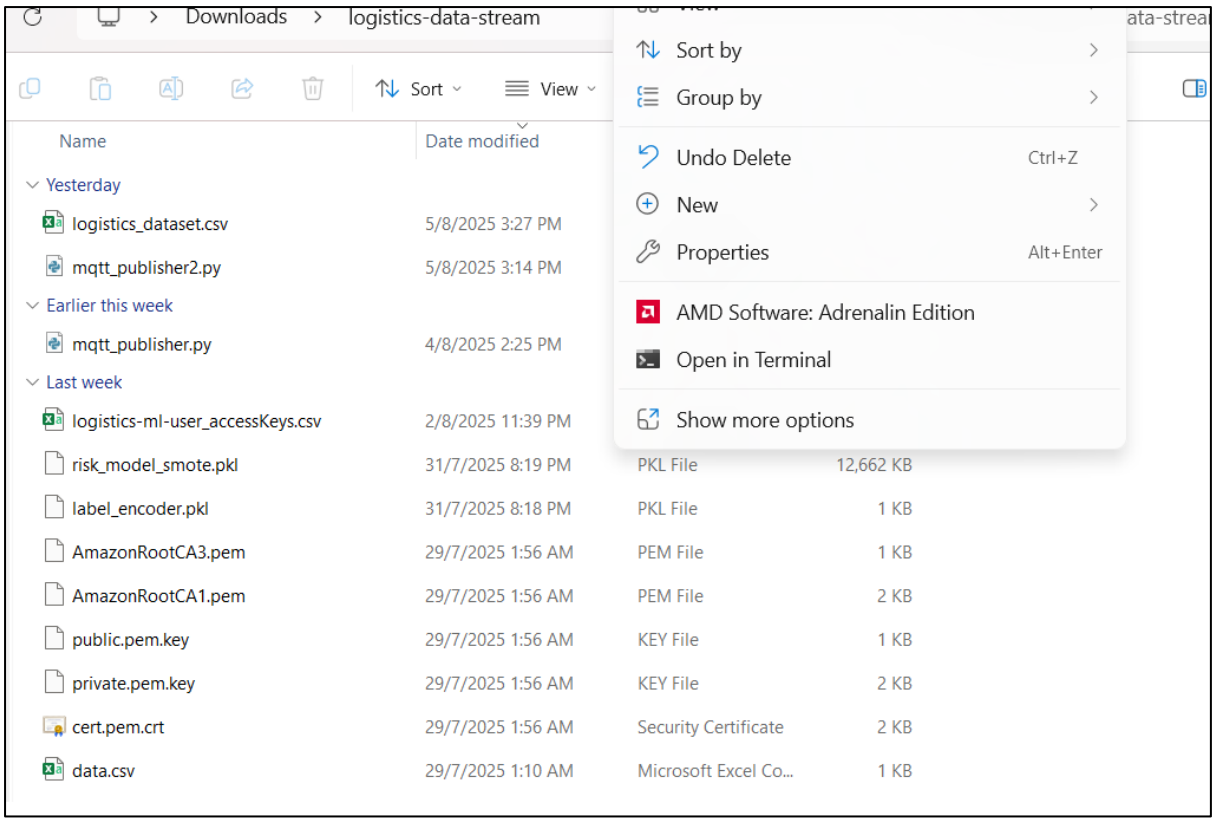
```
        periodic_payloads = {
                                                f"{TOPIC_PREFIX}/fatigue_monitoring_score":
float(row["fatigue_monitoring_score"]),
                                               f"{TOPIC_PREFIX}/disruption_likelihood_score":
float(row["disruption_likelihood_score"]),
            f"{TOPIC_PREFIX}/eta_variation_hours": float(row["eta_variation_hours"]),
                                                 f"{TOPIC_PREFIX}/order_fulfillment_status":
float(row["order_fulfillment_status"]),
            f"{TOPIC_PREFIX}/shipping_costs": float(row["shipping_costs"]),
                                                 f"{TOPIC_PREFIX}/customs_clearance_time":
float(row["customs_clearance_time"]),
            f"{TOPIC_PREFIX}/delay_probability": float(row["delay_probability"]),
                                                 f"{TOPIC_PREFIX}/delivery_time_deviation":
float(row["delivery_time_deviation"])
        }


        print("\nSending periodic features:")
        for topic, value in periodic_payloads.items():
            message = json.dumps({"value": value})
            client.publish(topic, message)
            print(f"Published to {topic}: {message}")


        # Build trigger payload
```

```
            trigger_payload = {
              "fixed": {k.replace(f"{TOPIC_PREFIX}/", ""): v for k, v in fixed_payloads.items()},
                              "periodic": {k.replace(f"{TOPIC_PREFIX}/", ""): v for k, v in
periodic_payloads.items()},
                "realtime": realtime_history
            }

            # Send trigger to prediction topic
            client.publish(f"{TOPIC_PREFIX}/trigger_prediction", json.dumps(trigger_payload))
            print(f"\n📣 Published trigger to {TOPIC_PREFIX}/trigger_prediction")

            realtime_history = []
        time.sleep(10)
        simulated_time += timedelta(seconds=10)
        row_count += 1

except KeyboardInterrupt:
    print("\nStreaming stopped by user (Ctrl+C).")
finally:
    client.loop_stop()
    client.disconnect()
```

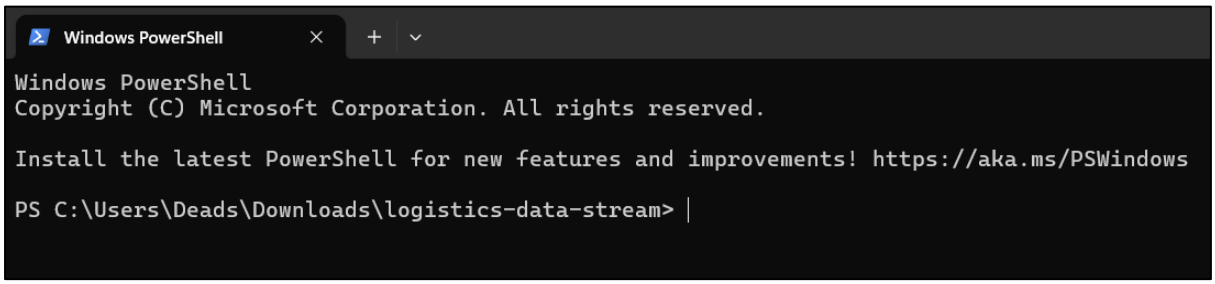
Type Ctrl C to stop the stream.



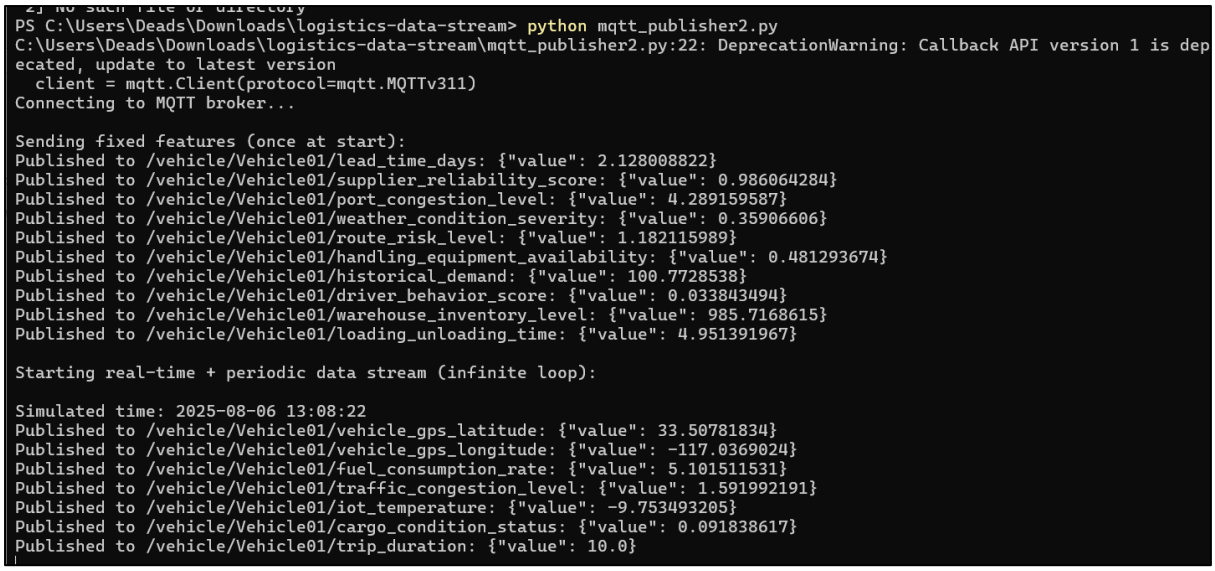
Final Data stream Folder.



At the folder we Open in Terminal



Then we type python mqtt_publisher2.py



Data is sent

See it updates

```
Sending periodic features:
Published to /vehicle/Vehicle01/fatigue_monitoring_score: {"value": 0.986596618}
Published to /vehicle/Vehicle01/disruption_likelihood_score: {"value": 0.499356931}
Published to /vehicle/Vehicle01/eta_variation_hours: {"value": 4.999979063}
Published to /vehicle/Vehicle01/order_fulfillment_status: {"value": 0.002689008}
Published to /vehicle/Vehicle01/shipping_costs: {"value": 296.6356703}
Published to /vehicle/Vehicle01/customs_clearance_time: {"value": 4.159664609}
Published to /vehicle/Vehicle01/delay_probability: {"value": 0.061456799}
Published to /vehicle/Vehicle01/delivery_time_deviation: {"value": 5.135373919}

📣 Published trigger to /vehicle/Vehicle01/trigger_prediction

Simulated time: 2025-08-06 13:09:22
Published to /vehicle/Vehicle01/vehicle_gps_latitude: {"value": 32.60188466}
Published to /vehicle/Vehicle01/vehicle_gps_longitude: {"value": -102.316635}
Published to /vehicle/Vehicle01/fuel_consumption_rate: {"value": 5.474694573}
Published to /vehicle/Vehicle01/traffic_congestion_level: {"value": 4.813078277}
Published to /vehicle/Vehicle01/iot_temperature: {"value": 36.83872787}
Published to /vehicle/Vehicle01/cargo_condition_status: {"value": 0.34449623}
Published to /vehicle/Vehicle01/trip_duration: {"value": 70.0}
```

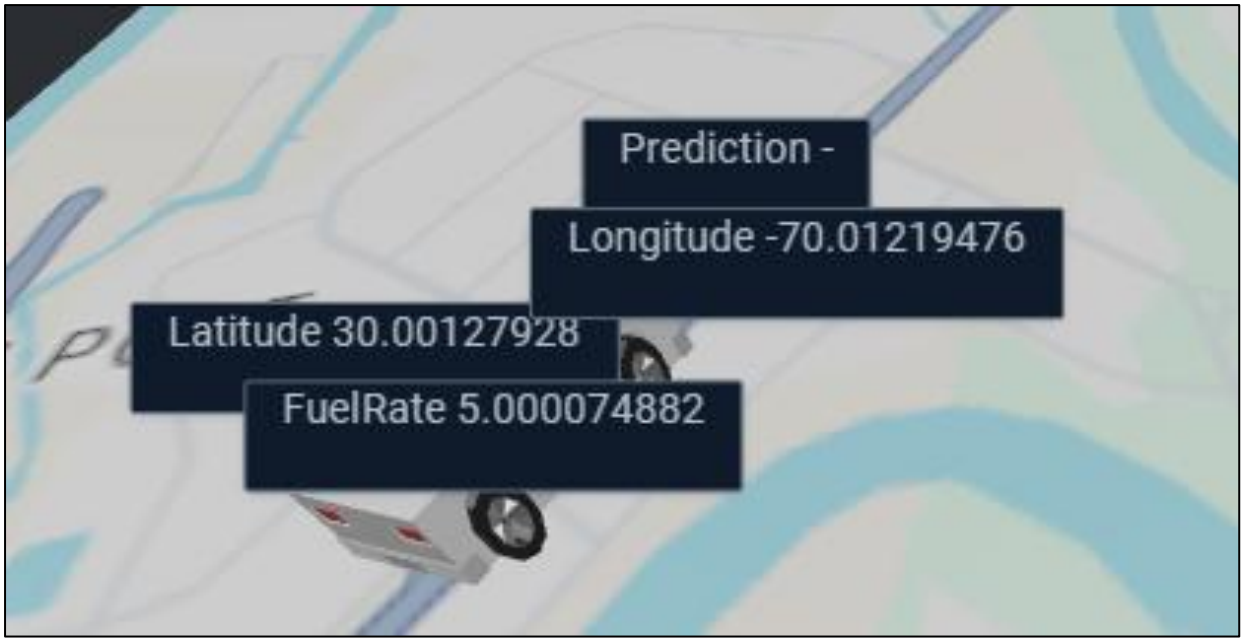
Once it reaches 1 minute the periodic values are sent as well as the trigger which goes to the second lambda to trigger the prediction.

prediction	71f347f2-e4f0-4c6b-a832-3cbf87472b1d	-	/vehicle/Vehicle01/prediction	-	Active	$aws/sitewise/asset-models/d9d4562e-5b6d-43be-8fa5-6254f69b40b3/assets/2be01bb0-c257-4a9e-8d3f-91f570a21557/properties/71f347f2-e4f0-4c6b-a832-3cbf87472b1d	2	August 06, 2025 at 13:09:17 (UTC+8:00)

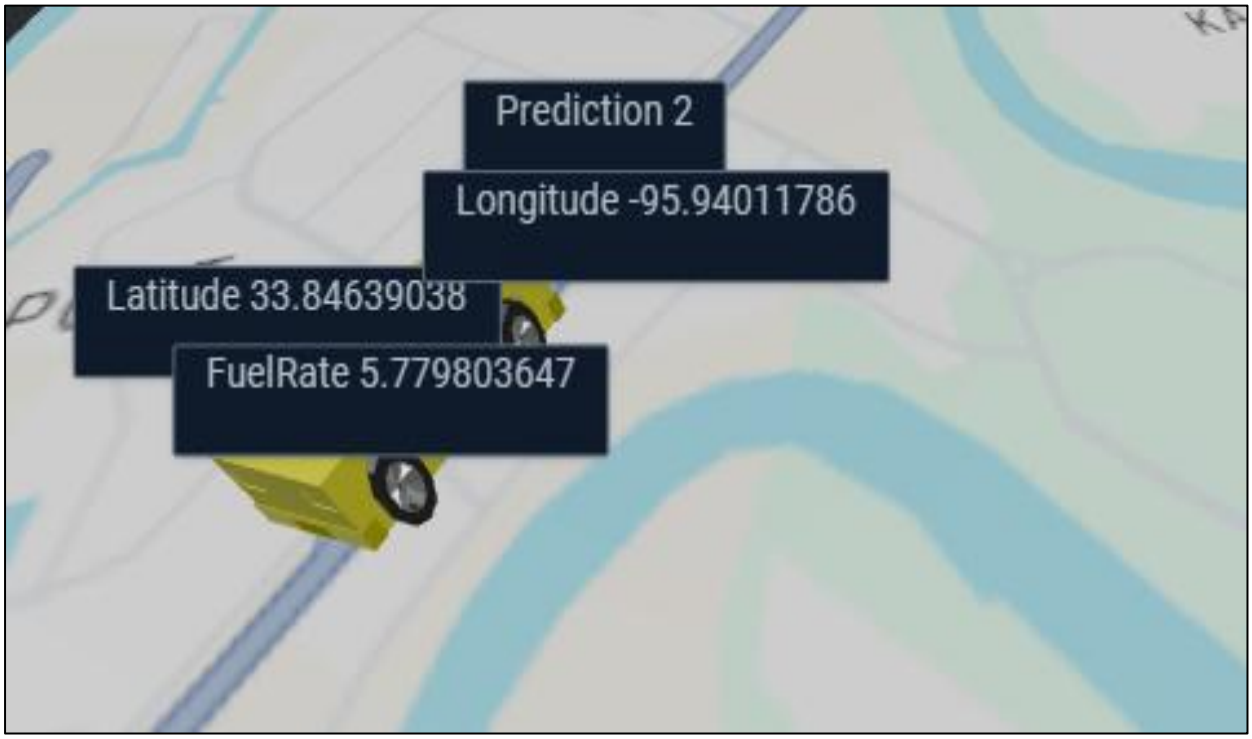
See in sitewise the prediction value is now 2(Moderate Risk)

Now we will show at the same time with Grafana

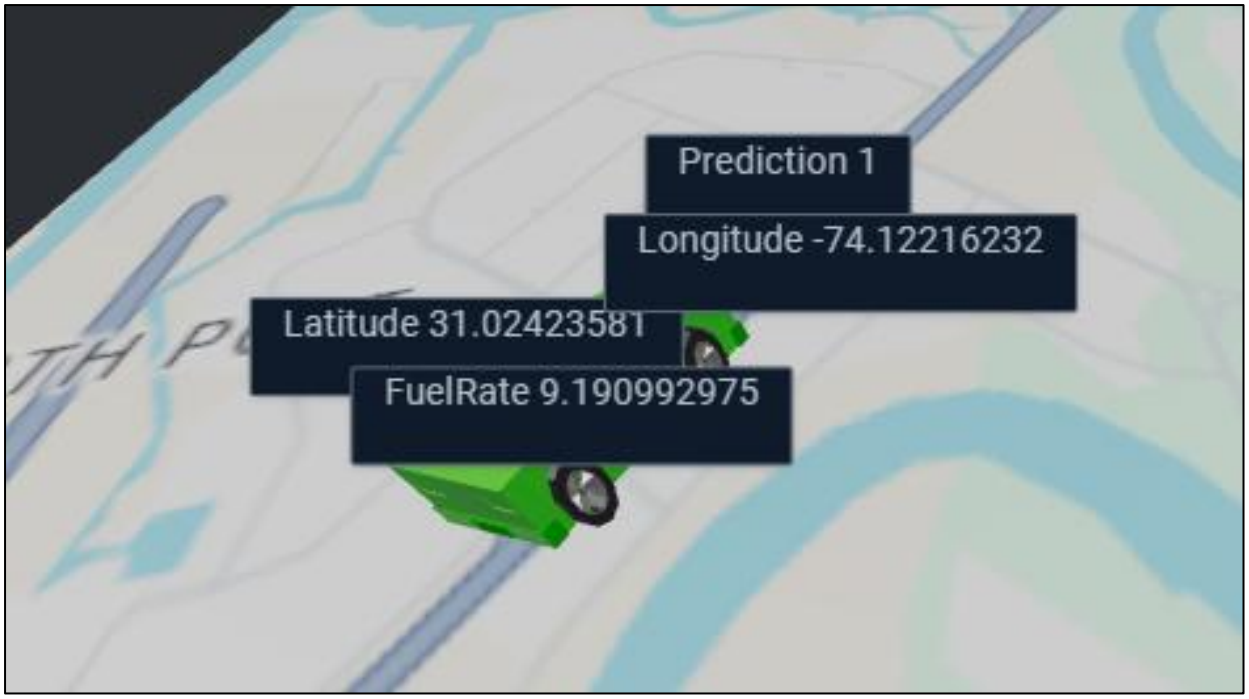
As you see here the values are sent including the fixed value of the route risk(Sorry for the bad spelling)



The values are updating every 10 seconds



The first minute is reached and the prediction is made: 2 which is Moderate Risk.



The next minute the prediction is made: 1 which is Low Risk.

Overall Summary

In this project, we successfully created a Digital Twin system that can simulate and predict the behavior of a logistics vehicle in real-time. We used a real dataset from Kaggle that includes important features like GPS, temperature, fuel rate, traffic level, and driver behavior. This dataset was very suitable because it looks like sensor data and can be streamed easily using AWS IoT Core and SiteWise.

We trained a machine learning model using Random Forest and improved it with SMOTE to handle class imbalance. The model gives predictions such as “Low Risk”, “Moderate Risk”, or “High Risk” for delivery. We deployed the model using Docker and Lambda to make sure it works even if the model file is big.

The whole system runs using AWS services such as IoT Core, Lambda, SiteWise, TwinMaker, and Grafana. We also managed to visualize the data in 3D and in dashboards with minimal cost, since we used free tools like Grafana. The result is a working Digital Twin that can help predict delivery problems before they happen.

This project shows that with the right dataset and proper setup, we can build a smart and useful Digital Twin using cloud tools. It can help logistics companies save time, reduce risk, and plan better in the future.