ASSESSMENT DOCUMENTATION

IOT MIDDLEWARE:

QUESTION: Send real-time telemetry data to AWS IoT Core and visualize in Amazon QuickSight for live asset tracking.

1. INTRODUCTION TO IOT MIDDLEWARE

IoT Middleware is a software layer that connects various IoT devices to cloud platforms and applications, abstracting the complexities of device communication and data management. It facilitates real-time data collection, processing, and analytics, enabling seamless interaction between IoT devices and cloud services.

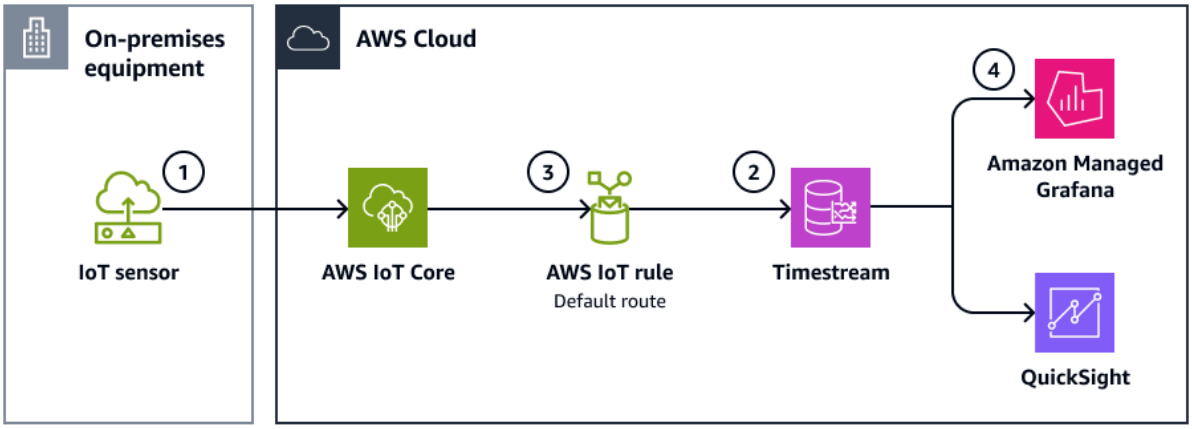
2. Experiment: Real-Time Asset Tracking using AWS IoT Core and Amazon QuickSight

This experiment demonstrates an end-to-end IoT middleware architecture where device telemetry data is transmitted to AWS IoT Core using MQTT protocol, processed and stored using Timestream, and visualized live in Amazon QuickSight. The architecture enables scalable real-time asset tracking with minimal latency.

AWS Services Used and Their Purpose

1. **AWS IoT Core** : Used for device connectivity and secure MQTT communication. Devices publish telemetry messages to topics like device/telemetry.
2. **AWS Timestream**: Time-series database for storing telemetry data. Automatically organizes data by timestamp for querying in real time.
3. **Amazon QuickSight:**BI tool used to build dashboards for live tracking. Connects to Timestream as a data source to visualize asset movement or sensor trends.

2. SYSTEM ARCHITECTURE OVERVIEW



High-level architecture showing the following flow:

- IoT Device(Simulated) → MQTT → AWS IoT Core
- Telemetry App (Python) → AWS IoT Core → Timestream → QuickSight for visualization

Components of the system:

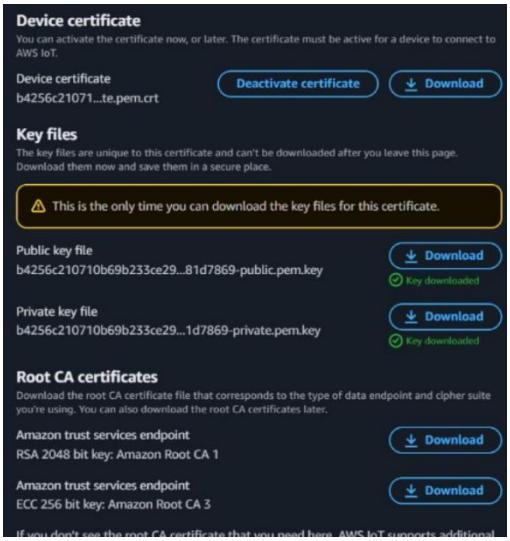
- **IoT Device Emulator**: Sends sensor data such as latitude, longitude, status of device device id ,via MQTT protocol to AWS IoT Core.
- **AWS IoT Core**: Serves as a message broker for securely routing the telemetry data from the IoT device to the cloud.
- **Amazon Timestream**: Stores time-series data, providing an efficient and scalable solution for storing telemetry data over time.
- **Amazon QuickSight**: Visualizes telemetry data by creating interactive dashboards and charts for real-time insights.

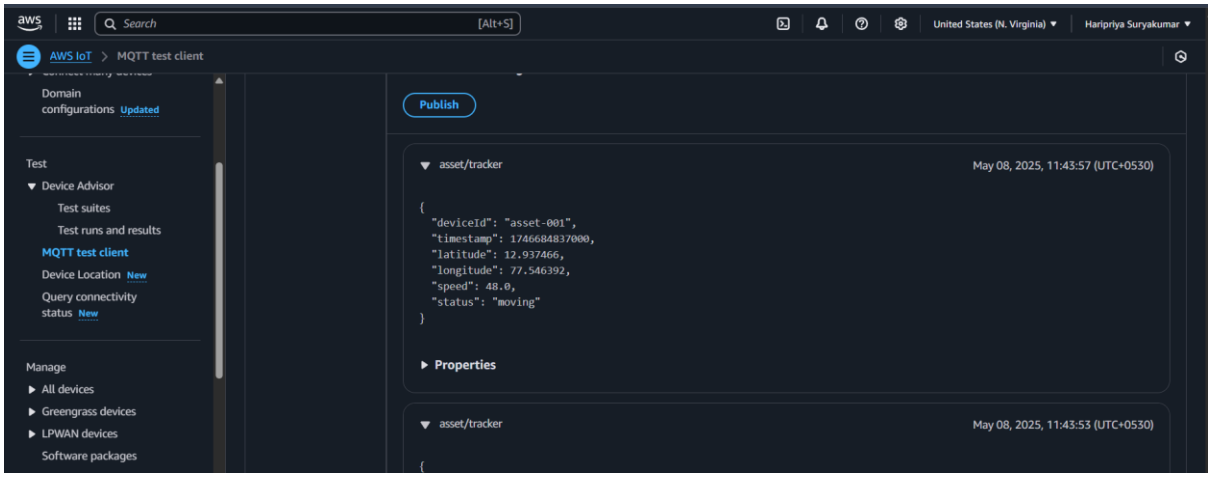
3. AWS IOT CORE SETUP

AWS IoT Core was used to establish secure and scalable communication between the device and the cloud. The following steps were followed:

- Created a Thing in AWS IoT Core and downloaded the security certificates (public key, private key, and root CA).
- Created and attached an IoT Policy with publish and subscribe permissions for topic: device/telemetry.

- Configured the device simulator or Python-based client to connect using MQTT over port 8883 with TLS encryption.
- Published mock telemetry data (location, speed, asset ID) at regular intervals



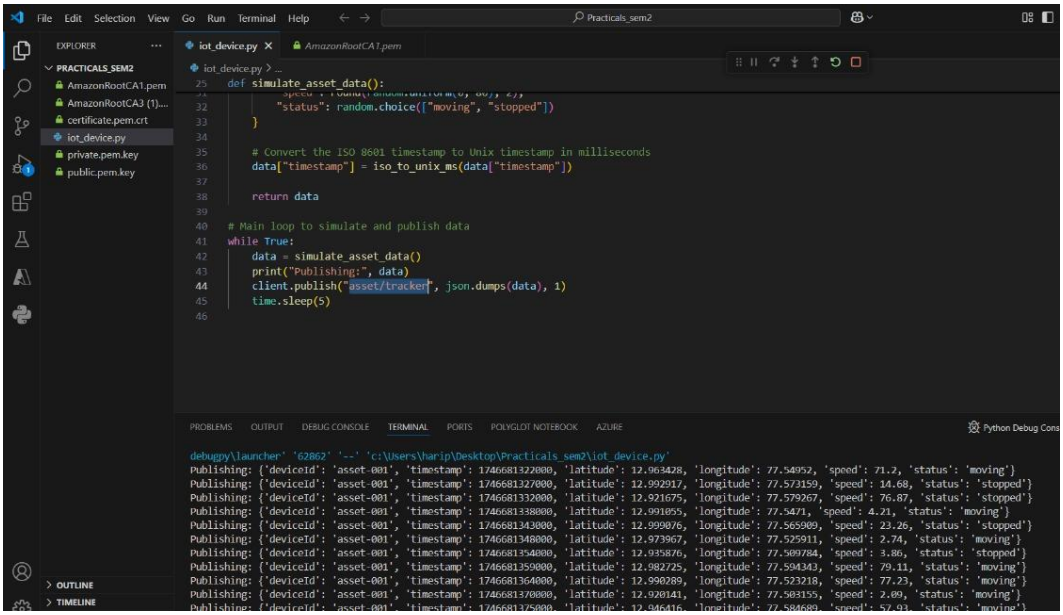


3. PYTHON TELEMETRY

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

import time

import json

import random

import datetime
```

```
# Initialize the MQTT Client

client = AWSIoTMQTTClient("AssetTracker")

client.configureEndpoint("a3dmb30l73pi7o-ats.iot.us-east-1.amazonaws.com", 8883)

client.configureCredentials("AmazonRootCA1.pem",
"private.pem.key","certificate.pem.crt")

cliet.configureOfflinePublishQueueing(-1)

clint.configureDrainingFrequency(2)

client.configureConnectDisconnectTimeout(10)

client.configureMQTTOperationTimeout(5)

client.connect()

def iso_to_unix_ms(iso_timestamp):    dt = datetime.datetime.strptime(iso_timestamp,
'%Y-%m-%dT%H:%M:%SZ')unix_timestamp = nt(dt.timestamp())  # Convert to Unix
timestamp (seconds)

                    unix_timestamp_ms = unix_timestamp * 1000    # Convert to
                milliseconds
                    return unix_timestamp_ms
                # Simulate asset data with timestamp conversion
                def simulate_asset_data():
                    data = {
                        "deviceId": "asset-001",
                        "timestamp": time.strftime("%Y-%m-%dT%H:%M:%SZ"),
                        "latitude": round(random.uniform(12.90, 13.00), 6),
                        "longitude": round(random.uniform(77.50, 77.60), 6),
                        "speed": round(random.uniform(0, 80), 2),
                        "status": random.choice(["moving", "stopped"])
                      }
                        data["timestamp"] = iso_to_unix_ms(data["timestamp"])
                    return data

while True:
```

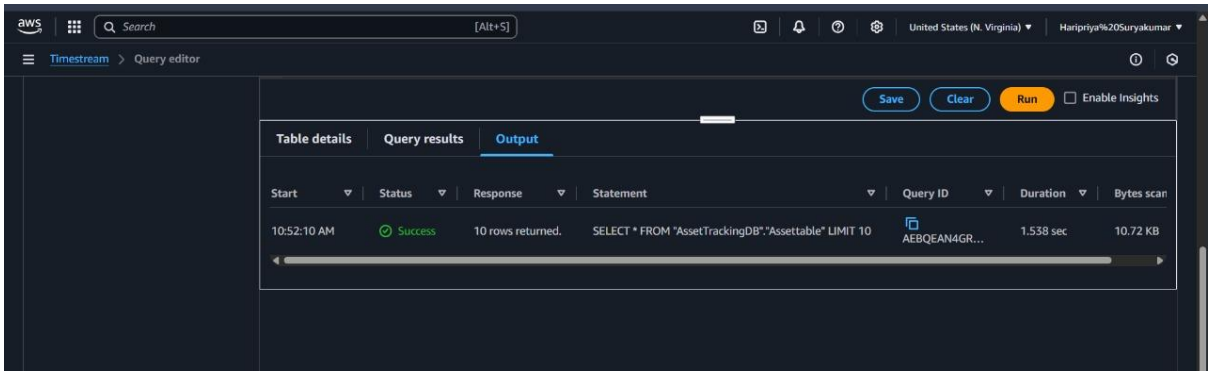
```
data = simulate_asset_data()
print("Publishing:", data)
client.publish("asset/tracker", json.dumps(data), 1)
time.sleep(5)
```



5. AWS TIMESTREAM CONFIGURATION

Timestream was selected as the storage solution for real-time telemetry data due to its optimized performance for time-series data.

- Created a Timestream database named AssetTelemetryDB and a table named TrackingData.
- Created a rule in AWS IoT Core to trigger on data from topic device/telemetry.
- Configured the rule action to insert parsed JSON data directly into the Timestream table using the SQL-based rule engine.



6. INTEGRATION FLOW AND END-TO-END DATA PATH

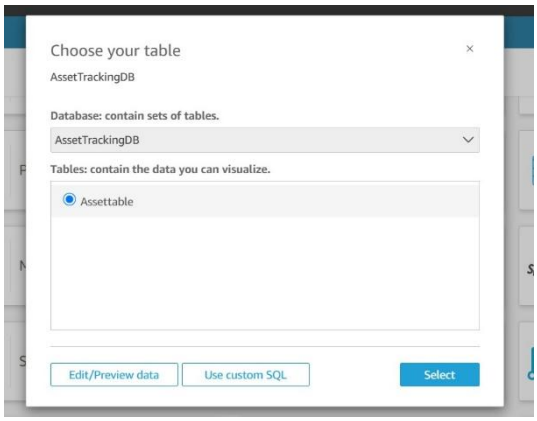
The complete integration is as follows:

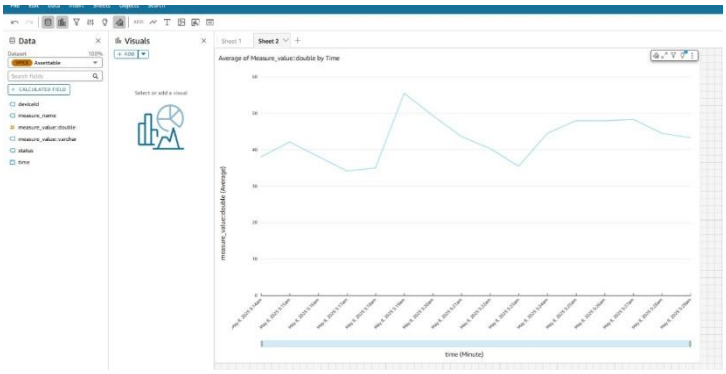
- Device publishes telemetry data (e.g., location, temperature, speed) to MQTT topic.
- AWS IoT Core receives data and routes it through a rule to AWS Timestream.
- Timestream stores the data with timestamps and dimensions (asset ID, region, etc.).
- QuickSight connects to Timestream and refreshes data periodically for live dashboard updates.

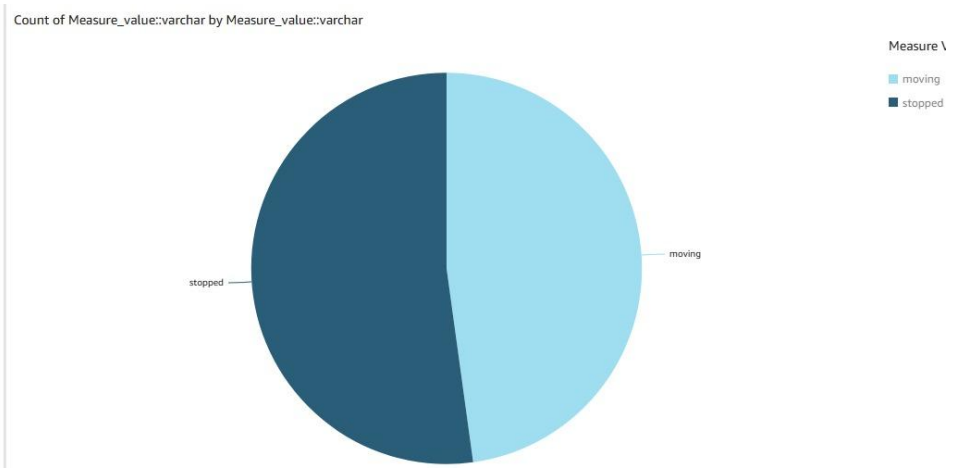
7. AMAZON QUICKSIGHT VISUALIZATION

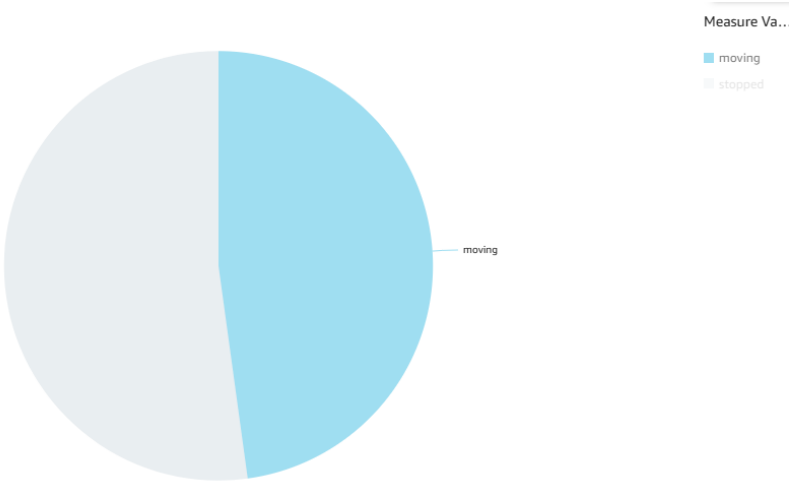
Amazon QuickSight was used to create a live dashboard that visualizes the incoming telemetry data.

- Enabled QuickSight to access AWS Timestream.
- Connected to the AssetTelemetryDB and selected the TrackingData table.









8. **SCALABILITY AND MONITORING**

- AWS IoT Core scales to support a large number of devices and messages, enabling seamless device communication and data ingestion.

- Amazon Timestream is designed for fast, scalable storage and querying of time-series data. It automatically scales as the amount of incoming data grows, ensuring efficient management of historical and real-time telemetry data.
- Amazon QuickSight enables visualization of large datasets by leveraging its powerful scaling capabilities, ensuring that live asset tracking can handle increasing volumes of data.

9. CONCLUSION

- In this implementation, real-time telemetry data was successfully ingested from devices using AWS IoT Core, stored in Amazon Timestream, and visualized in Amazon QuickSight for live asset tracking.
- The architecture supports scalability by leveraging AWS's cloud-native services such as IoT Core, Timestream, and QuickSight, which ensure seamless expansion of resources as data volume increases.
- Automation and monitoring are key highlights, with CloudWatch providing a robust system for tracking performance and alerting for any anomalies.
- The implementation provides a complete solution for real-time IoT data processing, visualization, and monitoring, offering valuable insights into asset status and performance in a scalable and automated manner.