## Project Documentation: Simulating Sensor Behaviour with MQTT, MongoDB, Redis, and FastAPI

**Table of Contents**

1. **Prerequisites**
2. **Project Structure**
3. **MQTT Broker Setup**
4. **MQTT Publisher**
5. **MQTT Subscriber**
6. **Data Storage with MongoDB**
7. **In-Memory Data Management with Redis**
8. **FastAPI Endpoint Design**
9. **Docker Integration with Docker Compose**
10. **Challenges and Solutions**
11. **Conclusion**

**Overview**

This document provides an overview of a project that simulates the behaviour of sensors, monitors their readings, and provides APIs to retrieve data based on specific criteria. The project integrates various technologies, including MQTT, MongoDB, Redis, and FastAPI, all orchestrated using Docker Compose.

**1. Prerequisites**

Before getting started, ensure you have the following prerequisites:

* Docker installed on your system
* Python installed on your system
* Using of Docker, MQTT, MongoDB, Redis, and FastAPI

**2. Project Structure**

**/mqtt-publisher**: Python MQTT client for simulating sensor readings.

**/mqtt-subscriber**: Python MQTT subscriber for storing received messages in MongoDB.

/mongo-redis: Implement Redis using Docker to store the latest ten sensor readings

**/fastapi-app**: FastAPI application with endpoints for retrieving sensor data.

**docker-compose.yml**: Docker Compose file for orchestrating services.

**3. MQTT Broker Setup**

Title: Setting Up MQTT Broker with Docker (Mosquitto)

Description: This section covers the deployment of a Mosquitto MQTT broker using Docker.

Steps:

1. Build and run the Mosquitto Docker container.
2. Verify the MQTT broker is running successfully.

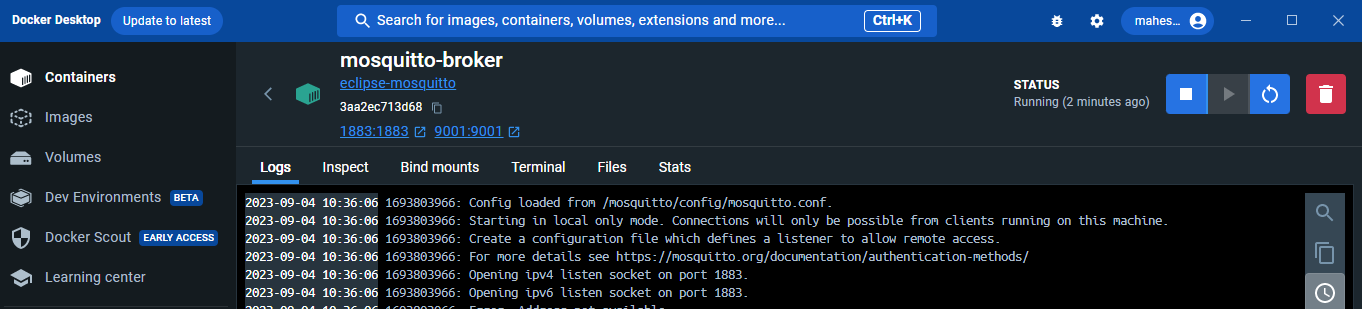
* To deploy the Mosquitto MQTT broker using Docker:

**🡺 docker pull eclipse-mosquitto**

**Docker Run Command**:

**🡺 docker run -d --name mqtt-broker -p 1883:1883 eclipse-mosquitto**

**Output:**



**4. MQTT Publisher**

Title: Creating an MQTT Publisher in Python

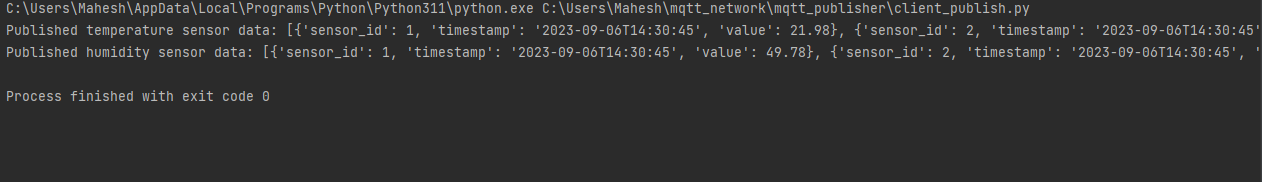
Description: This section covers how to create a Python MQTT client to mimic multiple sensor readings and publish them to MQTT topics.

Steps:

1. Install the necessary Python libraries (e.g., paho-mqtt).
2. Implemented the MQTT publisher script.
3. Configure the JSON payload structure.
4. Publish simulated sensor readings to MQTT topics.

🡺 **python client\_publisher.py (publishing data into MQTT broker)**

**Publishing data output:**



**5. Data Storage with MongoDB**

Title: Setting Up MongoDB with Docker

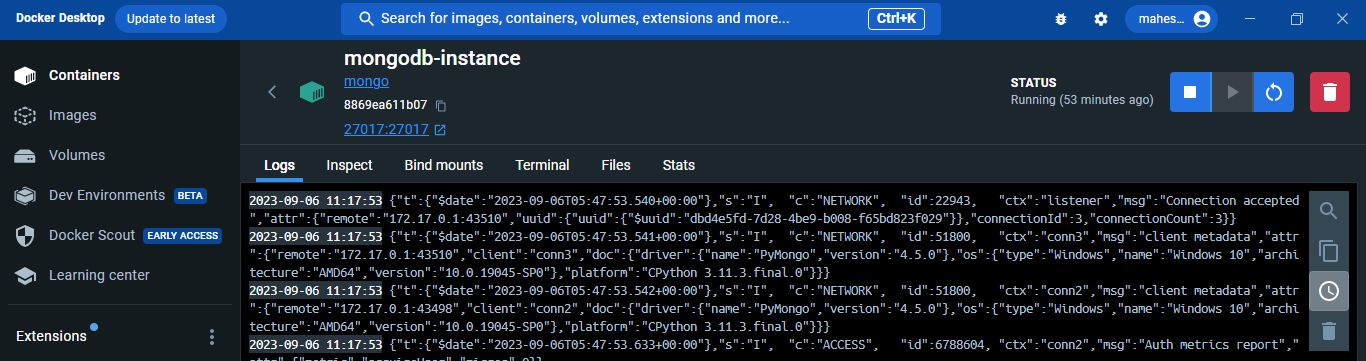
Description: This section explains how to initiate a MongoDB instance using Docker and save incoming MQTT messages to it.

Steps:

1. Create a Dockerfile for MongoDB configuration if needed.
2. Build and run the MongoDB Docker container.
3. Verify MongoDB is running and accessible.

🡺 **docker run -d --name mongo-instance -p 27017:27017 mongo**

**MongoDB docker container creating Output:**



**6. MQTT Subscriber**

Title: Creating an MQTT Subscriber in Python

Description: This section covers how to create a Python MQTT subscriber to receive and store MQTT messages in a MongoDB collection.

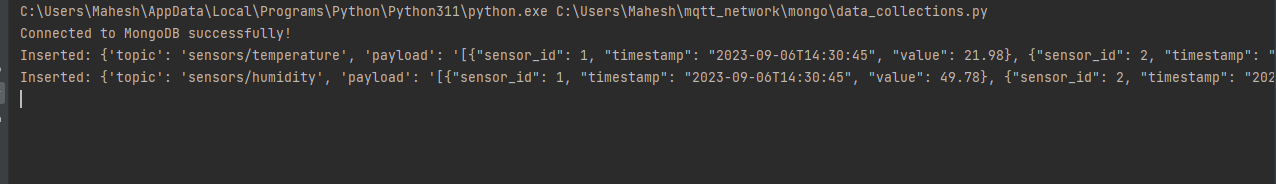
Steps:

1. Install the required Python libraries (e.g., paho-mqtt, pymongo).
2. Implemented the MQTT subscriber script.
3. Configure MongoDB connection and storage logic.
4. Receive and store MQTT messages in MongoDB.

🡺 **python consumer.py (Checking data receiving or not to MQTT broker)**

**🡺python data\_collection.py (Loading data into mongodB collections)**

**Inserted message to MongodB output:**



**7. In-Memory Data Management with Redis**

Title: Setting Up Redis with Docker

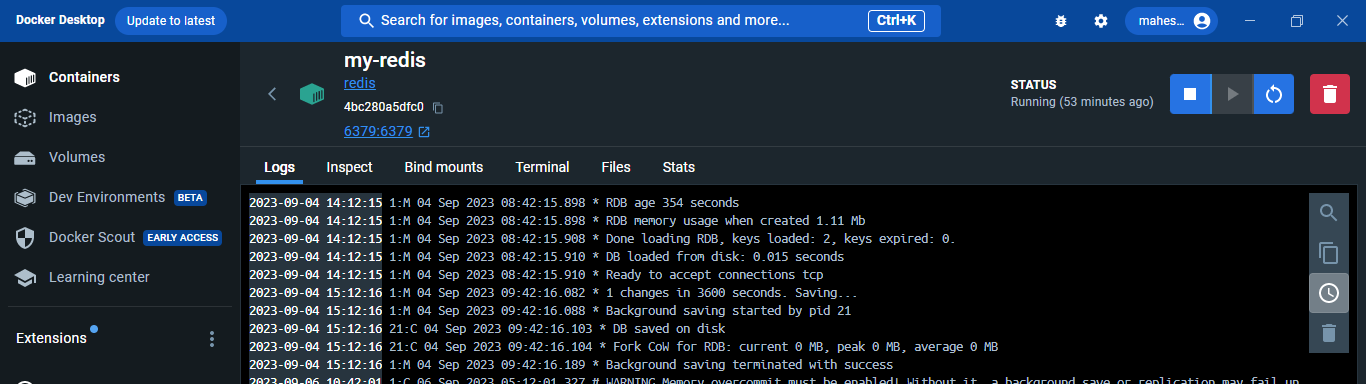
Description: This section covers how to implement Redis using Docker to store the latest ten sensor readings in-memory.

Steps:

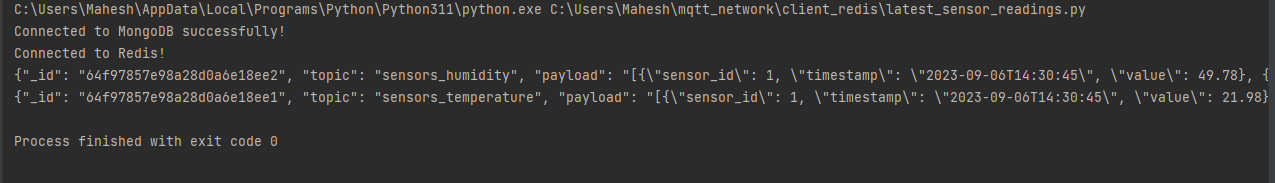
1. Create a Dockerfile for Redis configuration if needed.
2. Build and run the Redis Docker container.
3. Verify Redis is running.

🡺 **docker run -d --name my-redis- -p 6379:6379 redis**

**Redis Docker container creating Output:**



**Last ten sensor reading from Redis output:**



**8. FastAPI Endpoint Design**

Title: Designing FastAPI Endpoints

Description: This section outlines the design of FastAPI endpoints for interacting with sensor data.

Endpoints:

1. An endpoint that allows users to fetch sensor readings by specifying a start and end range.
2. An endpoint to retrieve the last ten sensor readings for a specific sensor.

🡺 **uvicorn main:app –reload (Run your FastAPI application using uvicorn)**

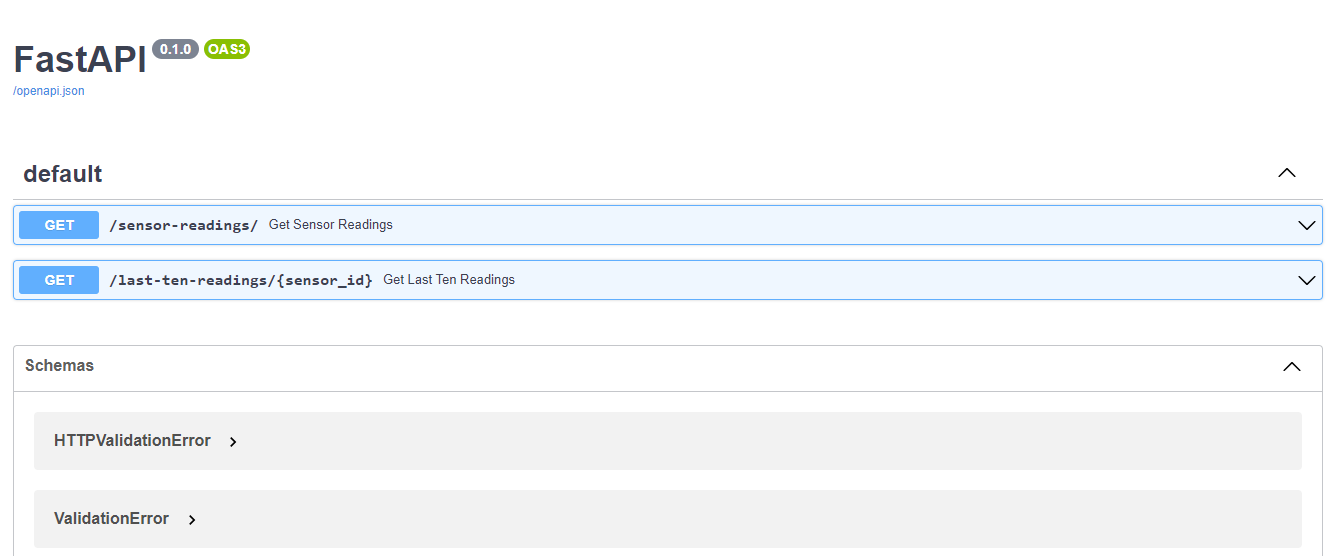
🡺 **Uvicorn running on** [**http://127.0.0.1:8000**](http://127.0.0.1:8000) **(access running port)**

**Testing endpoints:**

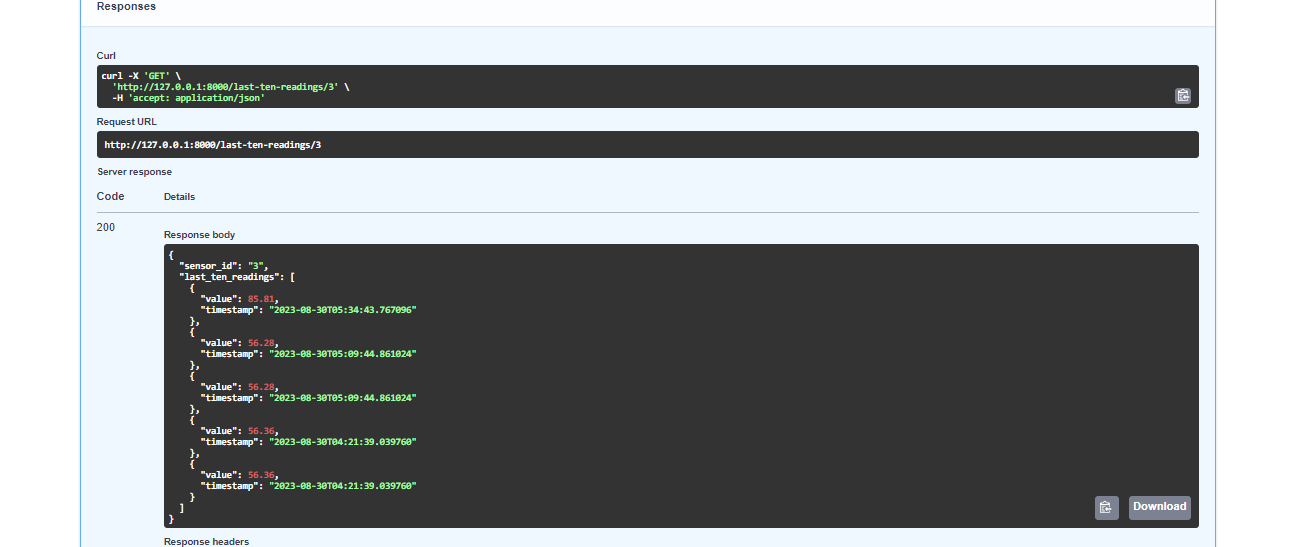
<http://localhost:8000/sensor-readings?start=2023-01-01T00:00:00Z&end=2023-12-31T23:59:59Z>

<http://localhost:8000/last-ten-readings/unique_sensor_id>

**FastAPI UI**



**sensor reading response**



**9. Docker Integration with Docker Compose**

Title: Integrating Services with Docker Compose

Description: This section explains how to integrate all services (MQTT broker, Python publisher/subscriber, MongoDB, Redis, FastAPI) using Docker Compose.

Steps:

1. Create a Docker Compose YAML file.
2. Define services, network configurations, and dependencies.
3. Build and start the entire project using Docker Compose.

* To start all services using Docker Compose:

🡺 **docker-compose up**

**10. Challenges and Solutions**

**Challenges**

1. Setting up the Mosquitto MQTT broker using Docker may involve issues with configuring ports, network settings, or security.
2. Initiating a MongoDB instance using Docker and saving incoming MQTT messages may involve challenges with configuration and data modelling.
3. Designing and implementing FastAPI endpoints can be challenging in terms of request parsing, data retrieval, and response generation.
4. Integrating all services using Docker Compose may involve issues with dependencies, networking, and debugging.

**Solutions**

1. Ensure that you've specified the correct ports and network settings in your Docker Compose file.
2. Make sure that no other service is using the same ports to avoid conflicts.
3. Consider securing your MQTT broker by configuring username and password authentication, as well as SSL/TLS encryption if needed.
4. Use the official MongoDB Docker image and provide the necessary configuration options (e.g., data volume, authentication) in your Docker Compose file.
5. Design your MongoDB schema to efficiently store sensor readings. Consider indexing fields used for queries.
6. Follow the FastAPI documentation to create your endpoints, including request and response models.
7. Implement validation and error handling for input parameters.
8. Test your API endpoints using tools like **curl** or Postman to ensure they work as expected.
9. Organize your Docker Compose file logically, ensuring that services are started in the correct order (e.g., MQTT broker before MQTT publisher).
10. Use custom Docker network bridges or link containers as needed to ensure they can communicate.

**11. Conclusion**

In conclusion, this project demonstrates how to simulate sensor behaviour, capture and store sensor data using MQTT, MongoDB, and Redis, and expose the data through a FastAPI-based API. Docker Compose is used to orchestrate the various services, making it easy to set up and manage the project environment.