Title -: Exalens Coding Challenge

Owner -: Nalla Vikram

This coding challenge is all about having sensors that measure things like temperature and humidity. Below is a simple system setup to make all integrations.

MQTT Broker Setup: Mosquitto MQTT broker is deployed using Docker. This component enables efficient communication between sensors and our processing pipeline.

MQTT Publisher: UsingPython, Programmed MQTT client capable of generating multiple sensor readings. These readings, meticulously crafted in JSON format, are published to designated topics such as sensors/temperature and sensors/humidity.

MQTT Subscriber and MongoDB Integration: To collect the valuable data stream, we constructed a Python MQTT subscriber. This subscriber dutifully channels incoming messages and securely stores them within a MongoDB collection. Each entry in the collection is structured with critical sensor information, including a unique sensor ID, reading value, and timestamp.

Redis for In-Memory Data Management: Redis, deployed seamlessly through Docker, takes center stage in our in-memory data handling strategy. Redis manages the ten most recent sensor readings, ensuring rapid access and responsiveness.

FastAPI Endpoint: This API provides two crucial endpoints: one to fetch sensor readings within a specified time range, and another to retrieve the latest ten readings for a particular sensor. This streamlined access to data empowers users with actionable insights.

Docker Compose Integration: Bringing it all together, Docker Compose orchestrates the deployment of our entire system. This unified containerization approach simplifies development, testing, and deployment, ensuring a seamless experience across environments.

Instructions for setting up and interacting with the system using the docker-compose command-:

- It is a simple system setup using docker-compose.
- Command -: docker-compose up -d
- Docker compose command file builds our entire system.

FastApi, Mosquitto publisher and Mosquitto subscriber are integrated in one docker container, All the three services are started simultaneously and continuously capturing of data is streamlined When docker compose command is executed.

A detailed overview of each service is in the docker-compose.yml file.

Mosquitto (MQTT Broker):

- Description: Mosquitto acts like a traffic cop for our sensors. It helps them communicate with each other. Making sure all the messages get where they need to go.
- Purpose: Mosquitto listens on a special port (1883) and makes sure all the sensor messages are delivered correctly.

MongoDB (Database):

- Description: MongoDB is like a database to stores all the readings from our pretend sensors so we can look at them later. We've set it up with a special username and password for security. Which is hard coded there it self
- Purpose: MongoDB keeps track of all the sensor data, making sure it's safe and organised.

Redis (In-Memory Data Storage):

- Description: Redis is like a super-fast memory booster. It remembers the ten most recent sensor readings so we can get to them really quickly.
- purpose: Redis helps us keep track of the latest sensor data for speedy access.

Demo App (Application):

- Description: This is the main program that makes everything work together. It's like the conductor of an orchestra, making sure all the instruments play in harmony. This app is built from the files in the 'app' folder using a special Dockerfile.
- Purpose: The Demo App coordinates the communication between Mosquitto, MongoDB, and Redis, and lets users ask for specific sensor data through a friendly interface.

Api Information-:

Latest sensor readings Api curl-:

```
curl --location 'http://localhost:8000/latest/temperature'
```

Data Reading sensor readings Api curl-:

```
curl --location 'http://localhost:8000/readings' \

--header 'Content-Type: application/json' \

--data '{

   "from_date": "2023-09-09",

   "to_date" : "2023-09-10",

   "from_time":"16:34:21",

   "to_time":"16:34:51"

}'
```