**Syyclops Take-Home Assignment**

Welcome to the Syyclops take-home assignment! We’d like to see your ability to design, implement, and document an **end-to-end data workflow** in the context of **smart buildings**. This assignment focuses on **data infrastructure**, **integration**, and **analytics**, using **Docker**, **MQTT**, **Postgres**, and an optional **analytics or ontology layer** (like a semantic model or a dashboard/API). We expect this to be a challenging but manageable project for about **3–4 days**.

**Project Description**

You will build a mini data pipeline around a **smart building scenario**, in which building sensors publish data via **MQTT**. These sensors can represent anything from temperature, humidity, and energy usage to occupancy or access control events. The assignment involves:

1. **Setting up MQTT** to receive simulated building sensor data.
2. **Processing** those incoming messages and storing them in a **Postgres** database.
3. **Integrating building metadata** (i.e., floors, rooms, sensor types) to showcase how data relates to the physical structure of a building.
4. (Optional) **Applying a building ontology** (e.g., Brick Schema, Project Haystack, or a simplified custom ontology) to classify sensor data.
5. Optionally providing an **analytics/data visualization** layer (a small dashboard, reporting feature, or an API) that retrieves and aggregates data from Postgres.

All services should be containerized using **Docker**.

**Core Requirements**

1. **MQTT Broker**: Spin up an MQTT broker (e.g., Eclipse Mosquitto) via Docker.
2. **Sensor Data Generation**: Publish simulated data messages to the broker. Consider multiple sensor types (temperature, occupancy, or energy usage). The payload can be JSON or a simple string but should include relevant metadata (sensor ID, timestamp, reading value, etc.).
3. **Data Consumer**: Implement a service (in Python, Node.js, Go, or a language of your choice) that subscribes to the MQTT topic(s), processes the incoming messages, and inserts them into Postgres.
4. **Postgres Database**:
   * Define a schema to store sensor readings.
   * Include a reference table for **building metadata** (e.g., floors, rooms, sensor types) so each sensor reading can be associated with the correct location and device type.
5. **Docker Compose**:
   * Provide a Docker Compose setup for building and running the entire stack (MQTT broker, consumer service, Postgres, optional analytics service, etc.) with one command.

**Advanced Requirements (Optional / Bonus)**

1. **Ontology / Semantic Modeling**:
   * Show how you might align your building metadata with an existing ontology (e.g., Brick or Project Haystack) or a simplified variant. This could be as simple as a table referencing sensor classes (TemperatureSensor, OccupancySensor, etc.) mapped to rooms and floors.
   * Demonstrate how this ontology could support extensibility or advanced queries.
2. **Analytics / Dashboard**:
   * Develop a small analytics service that queries Postgres for aggregated metrics (e.g., average temperature, occupancy counts, energy usage) and presents them via:
     + A basic REST API endpoint, **or**
     + A minimal web dashboard with charts or tables.
   * Provide instructions or code for generating data visualizations if you choose a dashboard approach.
3. **Data Transformations**:
   * Perform data cleaning or transformations before inserting into Postgres. For instance:
     + Normalizing units of measurement.
     + Checking for out-of-range values and flagging anomalies.
     + Summaries or roll-ups in the database (daily averages, peak occupancy, etc.).
4. **Testing / CI**:
   * Include a simple automated test suite to verify critical paths (e.g., that messages are received properly, transformations are correct, data is stored in Postgres).
   * (Optional) If you want to showcase DevOps, integrate a CI pipeline.
5. **Scalability / Reliability**:
   * Configure the MQTT broker or consumer for handling higher message volumes.
   * Optionally demonstrate a strategy for scaling your consumer services or your database.

**Task Guidelines**

1. **Code Organization**: Keep your code maintainable, well-documented, and logically structured. Provide a concise README that describes your design choices.
2. **Building Ontology / Metadata**: Show how you handle building structure (floors, rooms, etc.) and how sensors map to that structure. Even if you don’t fully implement a known ontology, outline how it could be expanded.
3. **MQTT Topics**: Choose or define topic structures that reflect building hierarchies, sensor types, or device IDs. For example: building/roomID/sensorType.
4. **Schema Design**:
   * At minimum, store sensor ID, sensor type, room/floor reference, timestamp, and reading value.
   * If using an ontology, show how you store references to classes or relationships.
5. **Documentation**:
   * Provide instructions on starting/stopping the project (docker-compose up / docker-compose down).
   * Show how to publish test messages (e.g., using mosquitto\_pub, a custom script, or a test client).
   * Explain how to access any analytics endpoints or dashboards if implemented.

**Deliverables**

1. **Source Code & Docker Files**:
   * All code for the data consumer, optional analytics service, and any scripts.
   * Docker Compose file(s) and Dockerfiles for each service.
   * **A link to your final public GitHub repository** containing the complete solution.
2. **Database Schema & Setup**:
   * SQL statements or migrations to create tables (including building metadata and sensor readings).
   * Setup instructions for seeding the database with sample metadata if needed.
3. **README**:
   * Summarize your approach, including any ontology or advanced logic.
   * Setup steps for running the containers and publishing messages.
   * Guidance for testing or exploring your analytics.
4. **(Optional) Documentation on Ontology**:
   * If you implement an existing or custom ontology, briefly describe the classes, relationships, or design rationale.

**Evaluation Criteria**

1. **Correctness**: Does the data flow from MQTT to Postgres accurately? Is the building metadata integrated properly?
2. **Code Quality**: Is the solution well-structured and documented?
3. **Docker Proficiency**: Can the entire stack be launched smoothly with Docker?
4. **Data Handling & Ontology**: How effectively are sensor data and building metadata managed? Are transformations or ontologies used?
5. **Analytics / Visualization** (if implemented): Are meaningful metrics presented? Is the interface clear?
6. **Scalability / Reliability** (Optional): Are there any architectural considerations for handling larger data volumes or future expansion?

**Submission Instructions**

* **Public Repository**: Host your solution on a public repo (GitHub).
* **README**: Make it easy for us to build and run your solution. Clearly document your approach, assumptions, and any additional setup.
* **Showcase**: If you implement a dashboard or advanced analytics, provide instructions or screenshots for how to view it. If you use an ontology, explain how it’s structured or could be extended.

**We look forward to your solution!**

This assignment will let you demonstrate your skill in data integration, containerization, and a bit of domain modeling within the smart building context. We encourage creativity and robust design—feel free to experiment with an ontology, advanced data transformations, or a polished analytics UI if time permits.

Good luck with your implementation, and we’re excited to see what you build!