

Senior IoT Application Developer Challenge

Task

Design and implement a robust IoT Event Processing System using MQTT and Docker. The system should include the following features:

1. MQTT Broker and Listener:

- Set up a Mosquitto MQTT broker inside a Docker container.
- Create a Python-based MQTT client using the `gmqtt` library to:
 - Subscribe to a topic `/devices/events`.
 - Process incoming messages containing JSON payloads with the format:

```
{
    "device_id": "<string>",
    "sensor_type": "<string>",
    "sensor_value": <float>,
    "timestamp": "<ISO8601 formatted timestamp>"
}
```

2. Message Validation:

- Implement a robust validation mechanism to ensure the payload adheres to the specified schema.
- Log invalid messages to a file with the error reason.

3. Data Storage:

- Store valid messages in a lightweight SQLite database with appropriate indexing. Include tables for:
 - **Devices:** `device_id`, `last_seen`.
 - **Events:** `event_id`, `device_id`, `sensor_type`, `sensor_value`, `timestamp`.

4. Real-Time Monitoring:

- Create a REST API (using Flask or FastAPI) to:
 - Retrieve the latest events for a given device.
 - List all registered devices and their last active timestamps.
- Containerize the REST API separately from the MQTT listener and broker.

5. Dockerization:

- Provide Dockerfiles for:
 - The Mosquitto MQTT broker.
 - The Python MQTT client application.
 - The REST API service.
- Use Docker Compose to orchestrate the setup.

6. Testing and Documentation:

- Manually publish test messages to the `/devices/events` topic using `mosquitto_pub` and verify that:
 - Valid messages are processed and stored correctly.
 - Invalid messages are logged.
- Document the setup and usage in a comprehensive `README.md`.

Help/Specifications

- Use the `gmqtt` library for MQTT communication.
- Design the system with scalability and modularity in mind.
- Log events and errors to a file using Python's `logging` module.
- Ensure the REST API follows RESTful principles and includes proper error handling.
- Use Docker Compose for easier deployment and to manage inter-container communication.
- Include an example `.env` file for any environment variables required by your application.

Submission

- Push your solution to a public GitHub/GitLab repository.
- Include a `README.md` with:
 - Detailed setup instructions.
 - System architecture overview.
 - API documentation (for the REST API endpoints).
 - Test cases and example usage.
- Share the repository link with us.

Evaluation Criteria

- **Architecture:** The solution's modularity and scalability.
- **Code Quality:** Readability, organization, and adherence to best practices.
- **Validation:** Robustness of data validation and error handling.
- **Documentation:** Clarity and comprehensiveness of the `README.md`.
- **Testing:** Inclusion and effectiveness of test cases.
- **Dockerization:** Proper containerization and seamless deployment using Docker Compose.