Tôi sẽ mô tả chi tiết cách **Listener** (listener.py) trong hệ thống của bạn hoạt động, bao gồm cách nó kết nối với NanoMQ, nhận dữ liệu từ các Gateway, xử lý dữ liệu và lưu trữ vào Redis.

**1. Tổng quan về Listener**

* **Vai trò**: listener.py là thành phần lắng nghe (listener) dữ liệu từ các Gateway thông qua MQTT broker (NanoMQ). Nó nhận dữ liệu thô từ các topic MQTT, xử lý và đẩy dữ liệu vào Redis để các thành phần khác (như Processor) sử dụng.
* **Input**: Dữ liệu từ NanoMQ qua topic bluetooth/+/data.
* **Output**: Dữ liệu thô và log được lưu vào Redis (queues beacon\_data và logs).
* **Thư viện**: Sử dụng paho-mqtt (phiên bản đồng bộ).

**2. Cách Listener hoạt động**

**2.1. Khởi tạo và cấu hình**

* **Load biến môi trường**:
  + Listener lấy thông tin cấu hình từ file .env bằng python-dotenv:

python

CollapseWrapCopy

load\_dotenv()

BROKER\_HOST = os.getenv("BROKER\_HOST", "localhost")

BROKER\_PORT = int(os.getenv("BROKER\_PORT", 1883))

QUEUE\_HOST = os.getenv("QUEUE\_HOST", "localhost")

QUEUE\_PORT = int(os.getenv("QUEUE\_PORT", 6379))

* + Mặc định: NanoMQ chạy trên localhost:1883, Redis trên localhost:6379.
* **Kết nối Redis**:
  + Tạo kết nối tới Redis để lưu dữ liệu:

python

CollapseWrapCopy

redis\_client = redis.Redis(host=QUEUE\_HOST, port=QUEUE\_PORT, db=0)

* **Tạo client MQTT**:
  + Khởi tạo một đối tượng MQTT client bằng paho-mqtt:

python

CollapseWrapCopy

client = mqtt.Client()

**2.2. Định nghĩa callback**

Listener sử dụng hai hàm callback để xử lý kết nối và tin nhắn từ NanoMQ:

1. **on\_connect**:
   * Được gọi khi kết nối tới NanoMQ thành công hoặc thất bại:

python

CollapseWrapCopy

def on\_connect(client, userdata, flags, rc):

if rc == 0:

print(f"Connected to {BROKER\_HOST}:{BROKER\_PORT}")

client.subscribe("bluetooth/+/data")

else:

print(f"Connection failed with code {rc}")

* + Nếu rc == 0 (kết nối thành công):
    - In thông báo kết nối.
    - Subscribe topic bluetooth/+/data để nhận dữ liệu từ tất cả Gateway (dùng wildcard + thay cho gateway\_id).
  + Nếu rc != 0: In mã lỗi (ví dụ: 1 = sai giao thức, 5 = không được ủy quyền).

1. **on\_message**:
   * Được gọi mỗi khi nhận được tin nhắn từ NanoMQ:

python

CollapseWrapCopy

def on\_message(client, userdata, msg):

topic = msg.topic

payload = json.loads(msg.payload.decode())

gateway\_id = topic.split("/")[1] *# Lấy gateway\_id từ topic*

beacon\_id = payload["beacon\_id"]

rssi = payload["rssi"]

ts = payload["timestamp"]

redis\_client.rpush("beacon\_data", json.dumps({

"beacon\_id": beacon\_id,

"gateway\_id": gateway\_id,

"rssi": rssi,

"timestamp": ts

}))

redis\_client.rpush("logs", json.dumps({

"time": ts,

"service": "Listener",

"message": f"Pushed: {beacon\_id}, {gateway\_id}"

}))

print(f"Pushed to queue: {beacon\_id}, {gateway\_id}")

* + **Xử lý tin nhắn**:
    - Lấy gateway\_id từ topic (ví dụ: bluetooth/GW1/data → GW1).
    - Giải mã payload JSON để lấy beacon\_id, rssi, timestamp.
    - Lưu dữ liệu thô vào queue beacon\_data trong Redis.
    - Lưu log hoạt động vào queue logs trong Redis.
    - In thông báo lên console để debug.

**2.3. Kết nối và chạy**

* **Gán callback**:
  + Gắn các hàm callback vào client:

python

CollapseWrapCopy

client.on\_connect = on\_connect

client.on\_message = on\_message

* **Kết nối tới NanoMQ**:
  + Thử kết nối tới NanoMQ:

python

CollapseWrapCopy

try:

client.connect(BROKER\_HOST, BROKER\_PORT, 60)

* + 60: Timeout 60 giây nếu không kết nối được.
  + Nếu thất bại (NanoMQ không chạy),抛出 ConnectionRefusedError.
* **Chạy vòng lặp**:
  + Dùng loop\_forever() để giữ client chạy liên tục, lắng nghe tin nhắn:

python

CollapseWrapCopy

client.loop\_forever()

* + Vòng lặp này xử lý tất cả sự kiện MQTT (kết nối, ngắt kết nối, nhận tin nhắn).
* **Xử lý ngoại lệ**:
  + Bọc trong try-except để bắt lỗi:

python

CollapseWrapCopy

except ConnectionRefusedError:

print(f"Failed to connect to {BROKER\_HOST}:{BROKER\_PORT}. Ensure NanoMQ is running.")

except Exception as e:

print(f"Error: {e}")

**3. Luồng hoạt động chi tiết**

1. **Khởi động**:
   * Listener đọc .env, kết nối tới Redis và tạo client MQTT.
2. **Kết nối NanoMQ**:
   * Gọi client.connect() để kết nối tới NanoMQ (localhost:1883).
   * Nếu thành công, on\_connect được gọi, subscribe bluetooth/+/data.
3. **Nhận dữ liệu từ Gateway**:
   * Khi Gateway publish dữ liệu tới topic như bluetooth/GW1/data, NanoMQ chuyển tiếp tới Listener.
   * on\_message được gọi với msg chứa:
     + msg.topic: Ví dụ bluetooth/GW1/data.
     + msg.payload: Chuỗi JSON như {"beacon\_id": "Beacon1", "rssi": -70, "timestamp": "2025-03-11T10:00:00"}.
4. **Xử lý dữ liệu**:
   * Trích xuất gateway\_id từ topic.
   * Parse payload để lấy beacon\_id, rssi, timestamp.
   * Tạo bản ghi dữ liệu thô và log.
5. **Lưu vào Redis**:
   * Dữ liệu thô → beacon\_data (queue).
   * Log → logs (queue).
6. **Lặp lại**:
   * Listener tiếp tục lắng nghe tin nhắn mới từ NanoMQ.

**4. Ví dụ minh họa**

* **Gateway gửi dữ liệu**:
  + Topic: bluetooth/GW1/data
  + Payload: {"beacon\_id": "Beacon1", "rssi": -70, "timestamp": "2025-03-11T10:00:00"}
* **Listener xử lý**:
  + Nhận tin nhắn → on\_message chạy.
  + Lưu vào Redis:
    - beacon\_data: {"beacon\_id": "Beacon1", "gateway\_id": "GW1", "rssi": -70, "timestamp": "2025-03-11T10:00:00"}
    - logs: {"time": "2025-03-11T10:00:00", "service": "Listener", "message": "Pushed: Beacon1, GW1"}
  + In console: Pushed to queue: Beacon1, GW1