University of Messina

Department of

Mathematical and Computer Sciences,

Physical Sciences and Earth Sciences

Computer Science - Data Analysis Study Program

Database mod. B Course

Integration of MQTT Server and Python for Big Data Analysis and Storage in Different Database Platforms

Professor: Armando Ruggeri

Mohammadjavad Qanati

Academic Year 2024-2025

**📄 Introduction**

**Context and Motivation**

The rapid proliferation of Internet of Things (IoT) devices in modern infrastructure — from smart homes to industrial sensors — has led to an exponential growth in real-time data generation. Managing this volume, variety, and velocity of data presents a major technological challenge, particularly in systems that require fast, flexible, and scalable data handling.

As organizations increasingly rely on sensor-driven data for decision-making, it becomes essential to build systems that are not only capable of receiving and analyzing large streams of heterogeneous data, but also of storing it in appropriate formats depending on its structure and usage context.

**Project Objective**

This project aims to develop a lightweight, modular system capable of:

* Receiving data from simulated IoT sensors using the MQTT protocol,
* Processing that data in real time using Python,
* Storing the processed data across three different types of databases:
  + SQL for structured records,
  + MongoDB for semi-structured or evolving data,
  + Neo4j for graph-based relationships between entities.

The integration of these technologies enables efficient data storage, flexible schema support, and powerful relationship analysis, covering a wide range of real-world data requirements in IoT ecosystems.

**Problem Statement**

Traditional data pipelines are often designed around a single type of database, which can be a limiting factor when dealing with diverse IoT data. Some data fits neatly into tables, while other data is better represented as documents or graphs. A one-size-fits-all solution compromises either performance or clarity.

This project addresses that problem by designing a multi-database pipeline — one that dynamically routes incoming MQTT messages to the appropriate storage platform based on message type or topic.

**Development Workflow**

The project will be developed in the following stages:

**1. MQTT Server Setup**

We will install and configure an MQTT server (e.g. Eclipse Mosquitto) locally to act as a message broker. This server will be responsible for accepting incoming messages published by simulated IoT devices and forwarding them to subscribed clients.

**2. Python Client Development**

Using the paho-mqtt library in Python, we will create a client application that:

* Connects to the MQTT broker
* Subscribes to specific topics (e.g. iot/temperature/room1)
* Listens continuously for new messages
* Parses and processes incoming data based on the topic

The Python client will serve as the central processing hub, routing messages to the correct database platform.

**3. Database Design and Integration**

The system will store data in three different types of databases, chosen based on the structure of each message:

* **SQL Database (e.g., MySQL or MariaDB)**  
  Structured data such as temperature, timestamps, and sensor IDs will be stored in relational tables.
* **MongoDB (Document Store)**  
  Semi-structured data — such as device health logs, custom sensor payloads, or variable sensor formats — will be stored as JSON documents in a MongoDB collection.
* **Neo4j (Graph Database)**  
  Data describing relationships (e.g., proximity between devices, network topologies) will be modeled as nodes and relationships in Neo4j.

Each database will be accessed via its appropriate Python library:

* mysql-connector-python for SQL
* pymongo for MongoDB
* py2neo for Neo4j

**4. Topic-Based Routing**

Incoming messages will be processed according to their MQTT topic. For example:

* Topics under iot/temp/ may be routed to SQL
* Topics under iot/device\_status/ may go to MongoDB
* Topics under iot/network/ may be used to create or update relationships in Neo4j

This routing logic will be written in the Python client using conditional topic parsing.

**5. Testing and Validation**

We will simulate incoming data using Python scripts or MQTT clients like mosquitto\_pub. Each component (message reception, processing, and storage) will be tested independently and in integration.

Functional and performance tests will be conducted to:

* Confirm data integrity in all databases
* Measure message processing latency
* Evaluate scalability for higher message throughput

**Project Tools and Technologies**

* **Programming Language**: Python 3.11+
* **Messaging Protocol**: MQTT (via Eclipse Mosquitto)
* **Database Platforms**: MariaDB/MySQL, MongoDB, Neo4j
* **Python Libraries**: paho-mqtt, pymongo, py2neo, mysql-connector-python
* **Simulation Tools**: custom Python scripts or command-line publishers