Multi-Sensor Data Collection and Storage with InfluxDB

PROJECT DOCUMENTATION

August 25, 2023

Shalini Priya

1 Introduction

The project focuses on building a scalable system for collecting and processing different types of simulated data streamed from three IoT devices such as cars, heart rate sensors and thermostats, and store that in database using different measurements for each device.

1.1 Technologies Used

The system utilizes Flask, Kafka, InfluxDB, and microservices using python language.

- **Python** : For coding and development
- **Flask** : Is used for building API's, managing routes, handling requests and responses and interaction with the InfluxDB database
- **Kafka** : Is used for IoT data ingestion to handle high frequency data streams
- **InfluxDB** : For storing timeseries and continuous streaming data and can scale horizontally to handle larger amount of data

1.2 Dataset

The data of the 3 IoT devices mentioned above consists of simulated readings from each device type. Each device generates specific data based on its type, and the data is then sent as JSON payloads to the API. Here's a breakdown of the data structure for each device.

- **Car Fuel Reading**
  - Device Type : Car1
  - Fuel Reading : "45.07"
  - Timestamp : "2023-08-25 09:44:00"
- **Heart Rate Sensor:**
  - Device Type : Patient1
  - Heart Rate : "78"
  - Timestamp : "2023-08-25 09:44:00"
- **Thermostat:**
  - Device Type : Home1
  - Temperature :"23.29"
  - Timestamp : "2023-08-25 09:44:00"

1.3 Features of the Project

- Simulates data for 3 IoT devices as per project needs
- Publishes data to Kafka topic for real-time processing
- Stores data to InfluxDB for time-series analysis
- Provides APIs for querying statistics (e.g average/median/max/min values) of specific vehicle or groups of vehicles for a specific timeframe from InfluxDB. Only one parameter (mean) for querying has been considered in the project
- Utilizes micro services

2 Architecture

The project consists of several interconnected components that collectively handles IoT vehicle data. Each component serves a distinct purpose in the data pipeline.The architecture illustrated in the Figure 1 shows the flow of data and interactions among the components

- **IoT Data Simulation**: Simulates data generated by three IoT devices
- **API Microservices**: This segment provides a robust API that accepts incoming payloads and supports querying the stored data from InfluxDB
- **Kafka Producer**: The Kafka Producer plays an important role in the project by publishing the simulated IoT data to a designated Kafka topic. This data, resembling real-time sensor readings, is made available for consumption by kafka consumer
- **Kafka Consumer**: The Kafka Consumer is responsible for retrieving data from the Kafka topic and subsequently storing it in the InfluxDB database
- **InfluxDB**: InfluxDB serves as the data repository. It stores the real-time vehicle data streamed from the Kafka topic. It's designed to efficiently manage and retrieve time-series data, making it a suitable choice for IoT applications

3 Project Structure

Figure 2 shows the structure of the project which has been developed in PyCharm IDE. It has following components:

- **api.py**: Contains Flask API code that accepts payloads, publishes them to Kafka and GET API to query the database for statistics
- **client_simulation.py** : Contains code to generate the data every one second for three IoT devices

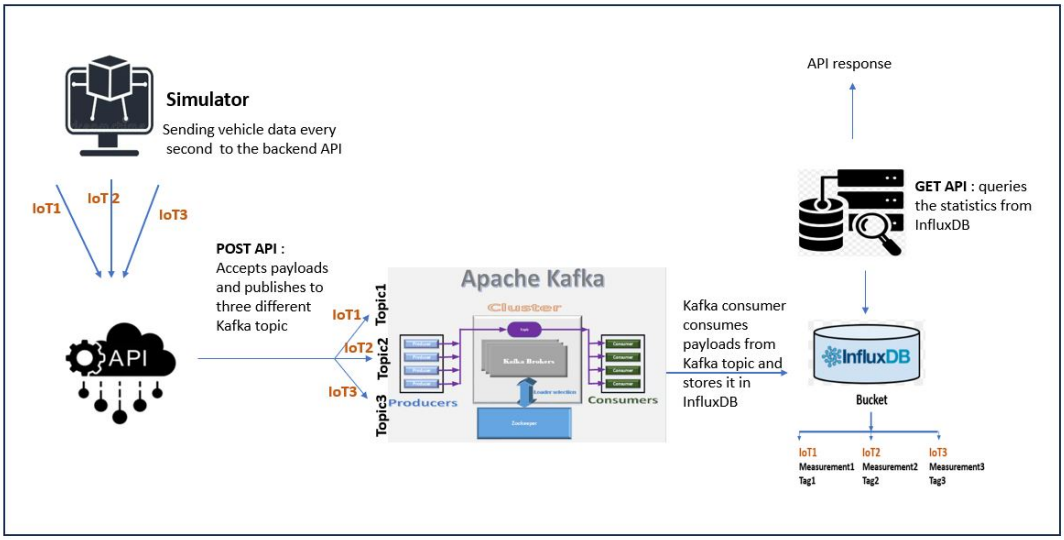

FIGURE 1: Project Architecture

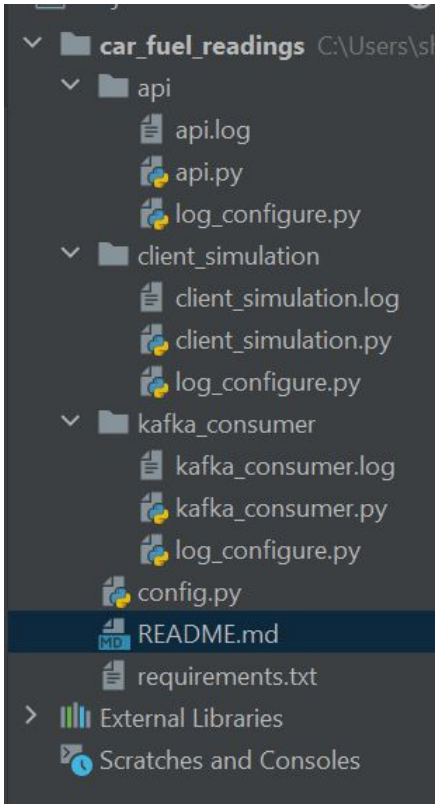

FIGURE 2: Project Structure

- **kafka_consumer.py**: contains Kafka consumer code which consumes messages from the Kafka topic and stores them in InfluxDB
- The root directory contains the project's documentation **README.md**, **requirements.txt** file consisting of libraries that were used in the project with relevant version and **config.py** file which stores the configuration details of the project

During execution, logs of each project directory will be placed under the directory path.

4 Configuration and Instructions

4.1 Configuration

- **Set Up Environment** Install required libraries and softwares. Below comand is used to install dependencies from requirements.txt file:
  pip install -r requirements.txt
- **Kafka and InfluxDB Configuration** Install and set up Kafka and InfluxDB instances using below comands:
  - Open a command prompt and navigate to the Kafka directory, start Zookeeper server by running the comand shown in Figure 3

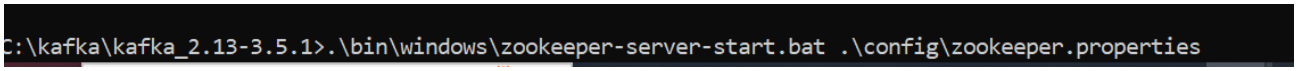

FIGURE 3: Zookeeper Server Comand

  - Open another comand prompt and Start Kafka server by running comad shown in Figure 4

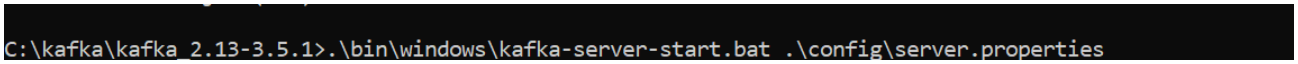

FIGURE 4: Kafka Server Comand

  - Open powershell and start Influx database instance using comand mentioned in the Figure 5

FIGURE 5: InfluxDB Comand

4.2 Instructions

- Start Kafka, InfluxDB and Zookeeper servers
- Run api.py file. API services can be accessed by below links:
  - `http://127.0.0.1:5000/add-readings` : for reading payloads shown in Figure 6
  - `http://127.0.0.1:5000/get-statistics?start_time=2023-08-22T00:00:00Z&end_time=2023-08-25T07:14:24Z` : for getting the mean value of temperature (Iot device=thermostat) for a particular timeframe shown in Figure 7

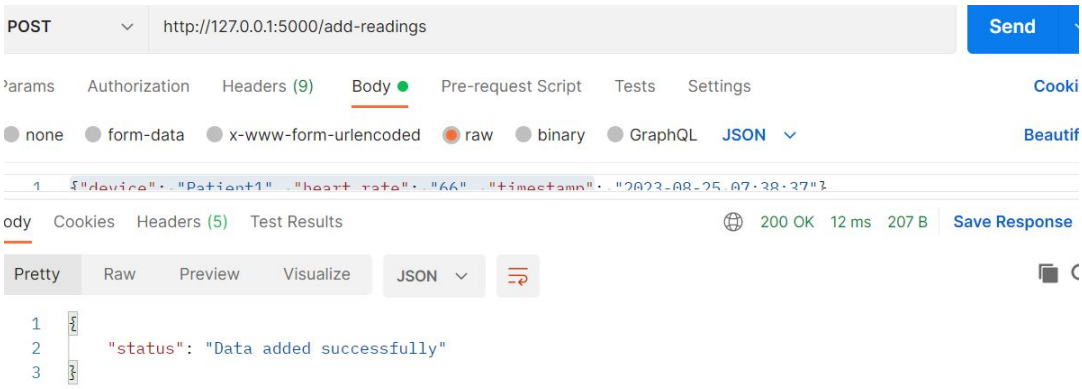

FIGURE 6: POST API Response in Postman

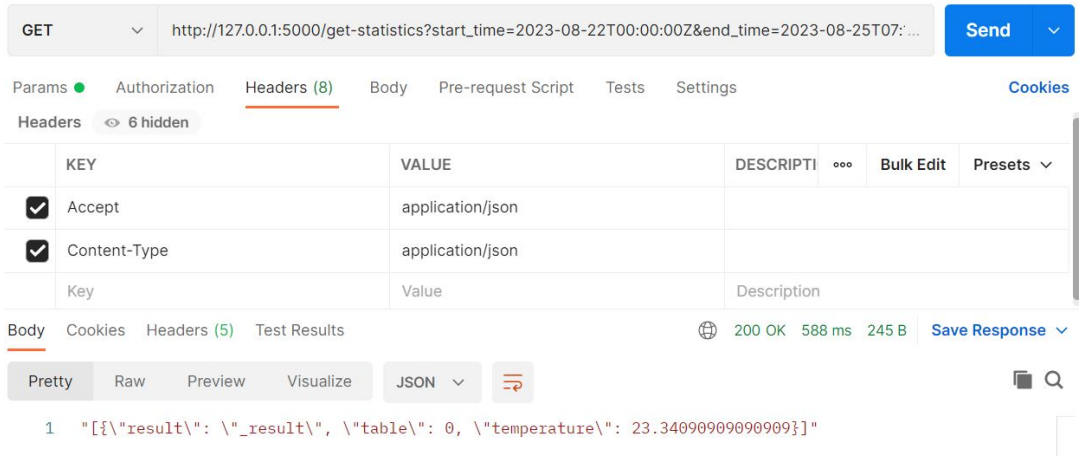

FIGURE 7: GET API Response in Postman

- Run client_simulation.py for generating data for three IoT devices every 1 second
- run kafka_consumer.py file to store the payloads in database

5 Limitations

- Security mechanisms like data encryption and authentication need to be implemented to prevent unauthorized access to the system
- Consumer consumes messages from only one of the kafka topics (one IoT device), the code needs to be enhanced further for the other two IoT devices
- Current project includes limited dataset with less parameters, which will affect the ability to train machine learning models accurately
- Currently single executable file (driving the project) has not been included due to time constraint

6 Future Work for Scalability

- Choose an appropriate deployment method such as containerization (using Docker) and orchestration using Kubernetes
- Set up load balancer to distribute incoming requests
- Machine learning model optimization

7 Output

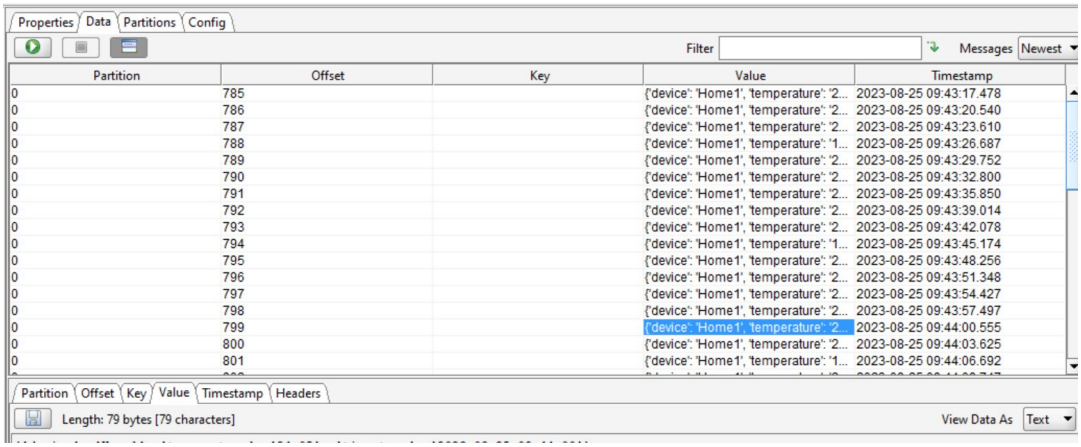

FIGURE 8: Data Storage in Kafka topic

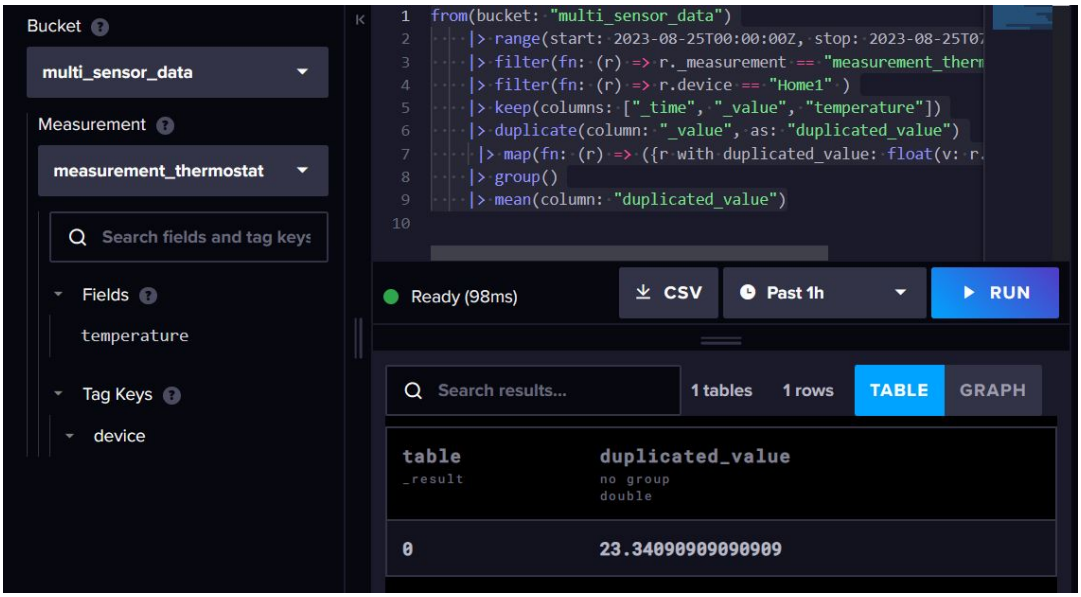

FIGURE 9: InfluxDB