**Pre-Interview Task: Advanced Real-Time Data Pipeline and Analytical Processing**

*Yashwanth Gopal*

*[Yashwanth.0968@gmail.com](mailto:Yashwanth.0968@gmail.com)*

**DATA DESCRIPTION**

Source : <https://www.kaggle.com/datasets/hmavrodiev/sofia-air-quality-dataset?select=2017-07_bme280sof.csv>

2017-07 Weather data of **Sofia** (Capital city of bulgaria). The data has temperature, pressure, and humidity readings from multiple outdoor sensors, captured in a specific timeframe. Therefore, using this timeseries data for building real time data pipeline suits our requirement for the task.

A screenshot of a computer

Description automatically generated

**ARCHITECTURE & DESIGN**

A diagram of a flowchart

Description automatically generated

**INSTRUCTIONS FOR SETTING UP & EXECUTION *(attached dependency.txt)***

1. **Visual Studio Code** as IDE
2. **SQLite** as database engine
3. The dependencies are attached in the file – ***“requirements.txt”***
4. ***“monitor.py”***  is the pipeline which monitors the folder data in real time. Therefore, enter the local path and filenames in the global variables.

**SCALABILITY**

In order to scale this pipeline for handling millions of files per day, a robust approach is necessary.

1. **APACHE KAFKA** = can be used as distributed event streaming platform.

Basically it has producer, queue and consumer.

Instead of suing monitor.py to set up file monitoring, file producer will act as kafka producer which will publish events such as filepath, timestamp, etc., to kafka topics. As per kafka topics, each file is processed **independently** and **parallelly** therefore **distributing the workload** across multipe processers. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability and more fault-tolerance. [1]. This is an example for **Horizontal scaling** because of distributed processing systems.

1. **USING CLOUD-BASED DATA SERVICES** = AWS Lambda is a cloud based data service which is driver by event and can scale automatically without needing manual intervention.
2. **PARTITIONING** = To optimize the storage of large datasets, partition the data by date, sensor ID, or other relevant criteria.

**HOW TO HANDLE DUPLICATE DATA, DELTA LOADS**

Handling duplicate data and **managing delta loads** is essential in a real-time pipeline. For efficient handling of delta loads (incremental data), ensure that only new data or modified data is processed and stored which can be done by maintaining the timestamp of last processed record. This also helps to avoid loading of duplicate data.

**IMPROVE EFFICIENCY USING INDEXES, PARTITIONS, ERROR LOGGING**

1. **ERROR LOGGING** = Implement a logging system that captures all errors and provides detailed error messages (e.g., file validation errors, database connection errors). Logs should be stored in a file or database for easy debugging.
2. **INDEXING** = to use **clustered** and **column-store indexes** for efficient storage and retrieval.
3. **PARTITIONING** = Partitioning involves dividing the data in a table into smaller segments based on a key column, which can be a range of values, a list of values. For **example** = we can use timestamp for range partitioning.

**RESULTS**

The pipeline continuously monitors the "data" folder for incoming CSV files **every 5 seconds**

*Output of Monitor.py*   
A black and white text on a black background

Description automatically generated

Raw data

*SELECT \* FROM sensor\_data LIMIT 10*

A screen shot of a computer

Description automatically generated

The aggregated data calculates the averages for each sensor for a particular day.

*SELECT \* FROM aggregated\_metrics LIMIT 10*

A screen shot of a computer

Description automatically generated

References

1. <https://kafka.apache.org/documentation/#:~:text=In%20Kafka%2C%20producers%20and%20consumers,to%20process%20events%20exactly%2Donce>.