**Shirli home assessment**

1. **Pipeline design**:

*Requirements:*

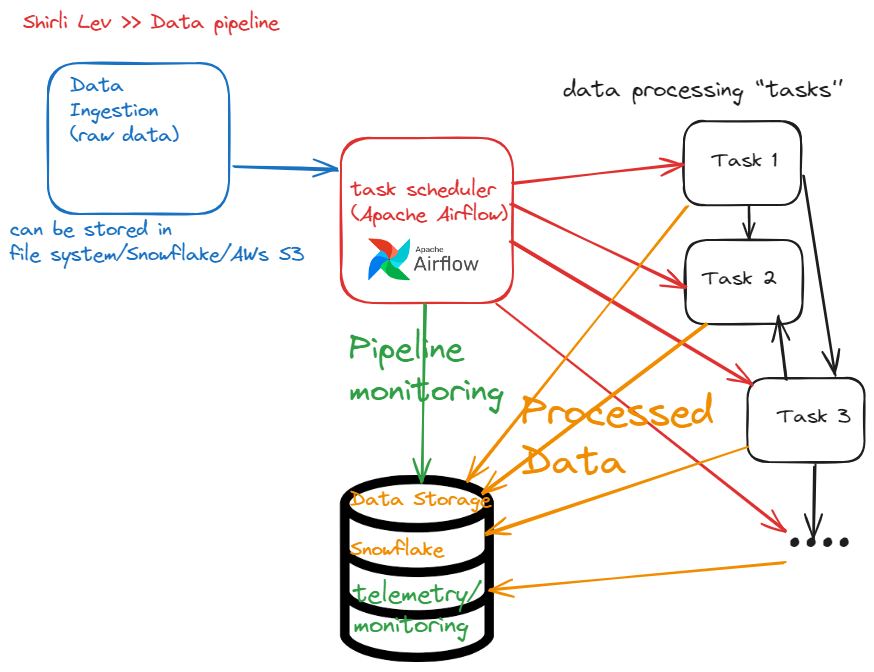
*• The assignment is to develop a data processing pipeline that will run different data processing “tasks” (a “task” is a code that can be run) on raw data. This system should process the tasks and store the processes’ results in a database or a data lake.*

*• The system should be run on demand and process different predefined “tasks”.*

*• One task can invoke other tasks.*

*• The system also produces telemetry on its runs (stores/updates the status of the runs).*

*•* ***Pipeline monitoring****: Each execution should be monitored and store its* ***state*** *and* ***results****. This should be ready to be queried at any time.*

* 1. *• You may use known orchestration services/frameworks.*
  2. a. Provide a diagram of the system (components/flows)
  3. 

b. Provide the data model (including db. schema. Which database will you use? Why?)

* 1. **data model**
  2. A screenshot of a computer

     Description automatically generated
  3. The bold fields are the primary keys.
  4. The red fields are foreign keys referencing tasks(task\_id).
  5. The green fields are foreign keys referencing executions(execution\_id).
  6. I believe Snowflake is an excellent choice for data storage and processing, especially for large scale data pipelines, due to its scalability, performance, ease of use, and strong integration with various data processing tools like DBT and storage services like AWS S3.
  7. As a cloud platform, Snowflake also supports semi structured data formats such as JSON, Avro, and Parquet, making it suitable for handling raw data.
  8. PostgreSQL is also a great solution, known for its robust feature set, compliance, and extensive support for complex queries and transactions. However, Snowflake offers several advantages for large-scale pipelines:
  9. **Scalability**: Independently and elastically scales compute and storage resources.
  10. **Performance**: Automatic query optimization and massive parallel processing.
  11. **Managed Service**: Fully managed, eliminating dba tasks.
  12. **Semi-Structured Data Support**: Natively supports formats like JSON, Avro, ORC, and Parquet.
  13. **Integration**: Seamlessly integrates with ETL tools, BI tools, and data processing frameworks.
  14. These features make Snowflake the better choice for large-scale data processing.
  15. For orchestration, I have chosen Apache Airflow because it simplifies scheduling and monitoring workflows. It is great for batch processing, ETL pipelines, and data engineering workflows.

Airflow's task-based management, rich user interface for monitoring, and extensive integration options (DBT, AWS, Snowflake, PostgreSQL, MySQL) make it an ideal orchestration framework.

Additionally, Airflow fulfills the requirements of running the system on demand, processing different predefined tasks, and allowing one task to invoke other tasks.

* 1. **Snowflake SQL Scripts**
  2. -- Create a database for our pipeline
  3. CREATE DATABASE data\_pipeline\_db;
  4. USE DATABASE data\_pipeline\_db;
  5. -- Create a schema for our tables
  6. CREATE SCHEMA pipeline\_schema;
  7. USE SCHEMA pipeline\_schema;
  8. -- Tasks table
  9. CREATE TABLE tasks (
  10. task\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  11. task\_name VARCHAR(255) NOT NULL,
  12. task\_description TEXT,
  13. created\_at TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP()
  14. );
  15. -- Task dependencies
  16. CREATE TABLE task\_dependencies (
  17. dependency\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  18. parent\_task\_id INTEGER,
  19. child\_task\_id INTEGER,
  20. UNIQUE (parent\_task\_id, child\_task\_id),
  21. FOREIGN KEY (parent\_task\_id) REFERENCES tasks(task\_id),
  22. FOREIGN KEY (child\_task\_id) REFERENCES tasks(task\_id)
  23. );
  24. -- Executions table
  25. CREATE TABLE executions (
  26. execution\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  27. task\_id INTEGER,
  28. start\_time TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),
  29. end\_time TIMESTAMP\_LTZ,
  30. status VARCHAR(50),
  31. error\_message TEXT,
  32. FOREIGN KEY (task\_id) REFERENCES tasks(task\_id)
  33. );
  34. -- Results table (for small results, larger results stored in Snowflake stage)
  35. CREATE TABLE results (
  36. result\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  37. execution\_id INTEGER,
  38. result\_data VARIANT, -- can store semi-structured data such as JSON, Avro, Parquet, XML.
  39. stage\_file\_path VARCHAR(255),
  40. FOREIGN KEY (execution\_id) REFERENCES executions(execution\_id)
  41. );
  42. c. An example of a query syntax to retrieve specific telemetry of a task that was run

SELECT t.task\_name,

e.execution\_id,

e.status,

e.start\_time,

e.end\_time,

e.error\_message,

r.result\_data,

r.stage\_file\_path

FROM tasks t

JOIN executions e ON t.task\_id = e.task\_id

LEFT JOIN results r ON e.execution\_id = r.execution\_id

WHERE t.task\_name = 'Task1'

AND e.start\_time > '2024-07-18 00:00:00';

**2*. Coding skills assessment*** *(not related to the above)*

*a. You are asked to write a process that checks for the arrival of new files and loads them into their corresponding tables in the DB.*

*b. There are two types of files in the folder (See appendix below):*

*i. Objects\_detection*

*1. The format of the file name - objects\_detection\_[timestamp].json*

*2. These files will hold streaming detection events that are sent from Mobileye's cars.*

*3. Each file can hold one or more events.*

* 1. *ii. Vehicles\_status*

*1. The format of the file name - vehicles\_status\_[timestamp].json*

*2. These files will hold the latest status of each vehicle.*

*c. You can choose whatever DB you wish to hold the received information, but you should take care of common queries that users can perform based on that information. You should also provide the code that configures the DB (creates table/defines scheme and so).*

I am debating between PostgreSQL and Snowflake. If I **assume** the data is not large-scale and I need a cost effective, open-source solution with strong JSON support, I choose PostgreSQL.

If the data is large scale, I prefer Snowflake because its performance is better, and it offers more scalability for handling large datasets efficiently.

I ran the code locally without connecting to the database, and it works well.

**All the code exists here:**https://github.com/shirlilev/mobileye-data-processor.git

This is the readme file:

**# Mobileye Data Processing System**

This system processes JSON files containing object detection events and vehicle status updates, storing the data in a PostgreSQL database.

**## Prerequisites**

- Python 3.7+

- PostgreSQL 12+

- pip (Python package manager)

**## Installation**

1. Clone this repository:

git clone https://github.com/your-username/mobileye-data-processor.git

cd mobileye-data-processor

2. Create a virtual environment and activate it:

python -m venv venv

source venv/bin/activate  # On Windows, use venv\Scripts\activate

3. Install the required packages:

pip install -r requirements.txt

4. Set up the PostgreSQL database: Create a new database named `mobileye\_data`

5. Update the `config.py` file with your database credentials and the folder to watch for new files.

**## Usage**

1. Run the script:

python main.py

2. The script will now watch for new files in the specified directory and process them automatically.

3. To stop the script, press Ctrl+C.

**## Design Considerations**

- Code reuse: The Handler, Watcher, and process\_file functions are designed to be reusable and easily extensible.

- Flexibility for changes: The modular design allows for easy modifications to file processing logic or database operations.

- Cloud readiness: The solution can be deployed to cloud environments by updating the DB\_CONFIG and WATCH\_FOLDER accordingly.

- Resilience: The script uses a file watcher to ensure no files are missed, and processed files are removed to prevent duplication.

- Scalability: While PostgreSQL can handle moderate volumes of data efficiently, for extremely large datasets, consider using a more scalable solution like Snowflake

            (A cloud native data warehousing solution that offers excellent scalability, performance, and support for complex queries.)

**## Performance Considerations**

- CPU usage: The script uses minimal CPU when idle and processes files efficiently.

- Memory usage: Files are processed one at a time to minimize memory consumption.

- Execution time: Bulk inserts are used to minimize database roundtrips and improve performance.

**## Assumptions**

- Files are valid JSON and follow the specified format.

- The watch directory is accessible and has appropriate permissions.

- The PostgreSQL database is properly configured and accessible.

- The system has sufficient disk space to handle incoming files.