**Shirli home assessment**

1. **Pipeline design**:

*Requirements:*

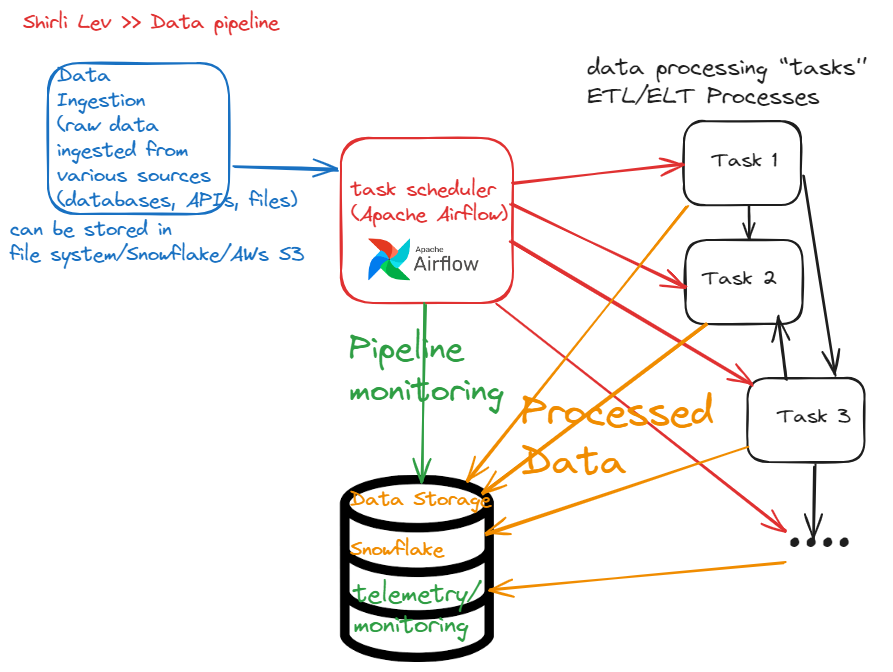
*• The assignment is to develop a data processing pipeline that will run different data processing “tasks” (a “task” is a code that can be run) on raw data. This system should process the tasks and store the processes’ results in a database or a data lake.*

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

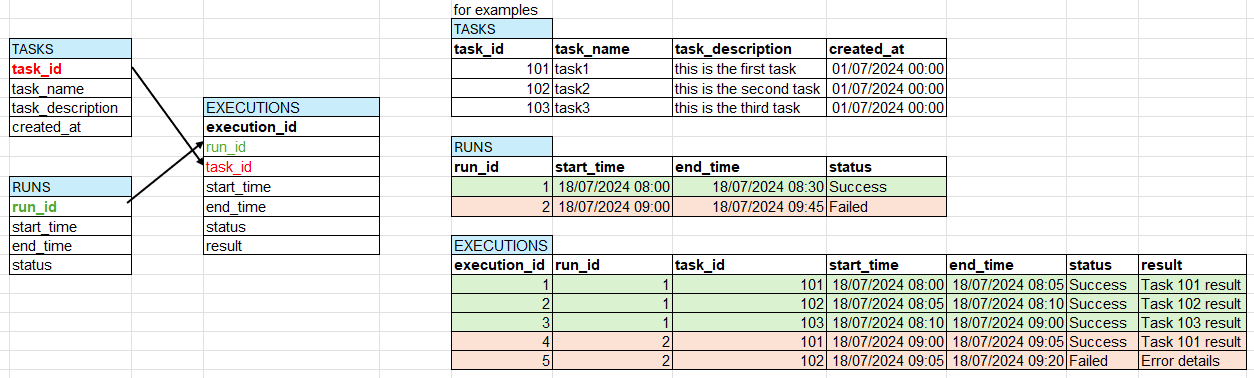
*• One task can invoke other tasks.*

*• 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. 
  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 runs(run\_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. CREATE TABLE pipeline\_schema.Tasks (
  9. task\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  10. task\_name STRING NOT NULL,
  11. description STRING,
  12. created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
  13. );
  14. CREATE TABLE pipeline\_schema.Runs (
  15. run\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  16. start\_time TIMESTAMP NOT NULL,
  17. end\_time TIMESTAMP,
  18. status STRING,
  19. created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
  20. );
  21. CREATE TABLE pipeline\_schema.Executions (
  22. execution\_id INTEGER AUTOINCREMENT PRIMARY KEY,
  23. run\_id INTEGER,
  24. task\_id INTEGER,
  25. start\_time TIMESTAMP NOT NULL,
  26. end\_time TIMESTAMP,
  27. status STRING,
  28. result STRING,
  29. created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,
  30. FOREIGN KEY (run\_id) REFERENCES pipeline\_schema.Runs(run\_id),
  31. FOREIGN KEY (task\_id) REFERENCES pipeline\_schema.Tasks(task\_id)
  32. );
  33. c. An example of a query syntax to retrieve specific telemetry of a task that was run

SELECT te.execution\_id, te.run\_id, t.task\_name, te.start\_time, te.end\_time, te.status, te.result

FROM pipeline\_schema.Executions te

JOIN pipeline\_schema.Tasks t ON te.task\_id = t.task\_id

WHERE te.run\_id = <specific\_run\_id>

AND t.task\_name = 'task1'

AND te.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 PostgreSQL and Snowflake for the database solution.

- PostgreSQL: If the data is not large-scale and I need a cost-effective, open-source solution with strong JSON support, PostgreSQL is the preferred choice.

- Snowflake: For large-scale data, Snowflake is preferable due to its better performance and greater scalability for handling large datasets efficiently.

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

\*\*\*\*While this solution can also be implemented on AWS (with files uploaded to S3 and a Lambda function triggered by S3 events), this is a code assessment and not an AWS exam.

**Therefore, I assume the files are in the local file system, and I use the `watchdog` library to implement it.**

All the code and readme files are available here:

https://github.com/shirlilev/mobileye-data-processor.git