Project

Tool Selection

Apache Airflow was used for orchestration, and MinIO was chosen as the data lake. Both run inside Docker containers to facilitate replication.

Three DAGs were built to perform the complete ETL process and save the data at three different levels following the **Medallion Architecture**.

All variables were added directly in Airflow via the UI and are only referenced at the beginning of each DAG.

First ETL

In **`api_etl`**, the data is extracted from the API and stored in its original JSON format in the **bronze** bucket. The idea was to keep the data as raw as possible.

Initially, the data is extracted via API, and a validation process checks whether the file is empty or missing the `"brewery_type"` and `"state"` columns. If any of these validations fail, the ETL process is aborted.

If the extraction is successful, the data is saved **locally** and then read again and stored in the **bronze bucket** in MinIO.

The DAG is configured to:

- Retry **3 times** in case of failure, waiting **2 minutes** between attempts.
- Send an **email alert** if all 3 attempts fail.
- Run **every 5 minutes** for testing purposes. The `"timedelta(minutes=5)"` argument could be replaced by `"@daily"`.
- Set an **SLA of 10 minutes** per task. If a task takes longer than 10 minutes, an **email alert** is sent.

At the end of the process, the next ETL process is triggered.

Second ETL

In **`parquet_etl`**, the data is extracted from **bronze**, converted to **Parquet format**, and stored in **silver**, partitioned by the `"state"` column.

The first task **clears the silver bucket** to ensure a **full overwrite** and avoid keeping outdated states that were present in previous JSON files but are no longer included.

Here, the data is **kept in memory** instead of being saved locally (as in the previous DAG). The goal is to demonstrate that both approaches are possible, especially for small files like this. If the files were larger, in-memory processing would not be recommended..

The data is **partitioned by state**, converted to **Parquet**, and then stored in the **silver bucket**.

The DAG is configured to:

- Retry **3 times** in case of failure, waiting **2 minutes** between attempts.
- Send an **email alert** if all 3 attempts fail.
- Set an **SLA of 10 minutes** per task. If a task takes longer than 10 minutes, an **email alert** is sent.

At the end of the process, the **final ETL process is triggered**.

Third ETL

In **`agg_etl`**, the data is extracted from **silver**, aggregated, and stored as **CSV** in the **gold bucket**. This step performs a **distinct count aggregation** by `"state"` and `"brewery_type"`.

The data is stored in CSV format because it is a very small dataset and to facilitate readability at the end of the process.

The **silver** bucket is read, and, once again, the data is processed **in memory** for simplicity. A loop reads each **Parquet file**, merging them into a single Pandas DataFrame.

At the end of the process, the data is **written to the gold bucket**, aggregated by `"state"` and `"brewery_type"`, with a **distinct count** applied.

The DAG is configured to:

- Retry **3 times** in case of failure, waiting **2 minutes** between attempts.
- Send an **email alert** if all 3 attempts fail.
- Set an **SLA of 10 minutes** for the task. If the task takes longer than 10 minutes, an **email alert** is sent.