# **Brewery Data Pipeline Documentation**

This project demonstrates a data pipeline solution using the **Open Brewery DB API**. The pipeline is structured using the **Medallion Architecture** and processes data through three layers: Bronze, Silver, and Gold. The pipeline is containerized using **Docker**, orchestrated with **Apache Airflow**, and developed primarily in **Python** with **PySpark scripts** for transformations and aggregations.

## **Project Overview**

The **Brewery Data Pipeline** performs the following steps:

1. **Ingests data** from the Open Brewery DB API.
2. **Stores raw data** in the Bronze layer (JSON format).
3. **Transforms data** for the Silver layer, partitioned by location and stored in Parquet format.
4. **Aggregates data** for analytical purposes in the Gold layer, producing a count of breweries by type and location.
5. **Orchestrates tasks** using an Airflow DAG with task dependencies.

### ****Key Components****

* **API Ingestion**: Fetches brewery data from the Open Brewery DB API.
* **Data Lake Architecture**: Follows the Medallion Architecture for structuring the data.
* **Airflow Orchestration**: Uses Airflow to schedule and run data pipeline tasks.
* **Docker Containerization**: Packages the application in Docker for easy deployment and modularity.

## **Project Structure**

brewery-data-pipeline/

├── README.md

├── git.sh

├── docker/

│ ├── Dockerfile

│ └── requirements.txt

├── airflow/

│ └── brewery\_data\_pipeline\_dag.py

├── src/

│ ├── 0\_fetch\_breweries.py

│ ├── 1\_store\_raw\_data.py

│ ├── 2\_transform\_data.py

│ └── 3\_aggregate\_data.py

└── test/

└── test\_breweries.py

## **Step-by-Step Installation and Setup Guide**

### ****Prerequisites****

* **Docker**: Make sure Docker is installed. You can download it from Docker's official site.
* **Docker Compose**: Ensure Docker Compose is available, as it simplifies container management.
* **Python**: Although not mandatory, it’s recommended for testing and running scripts locally if needed.
* **Airflow CLI**: For troubleshooting or debugging, having Airflow locally can be beneficial.

### ****Step 1: Clone the Repository****

Clone the project repository from GitHub.

git clone https://github.com/adolfoeliazat/brewery-data-pipeline.git

cd brewery-data-pipeline

### ****Step 2: Build Docker Image****

Navigate to the docker/ directory and build the Docker image with all dependencies.

docker build -t brewery-pipeline .

This command builds the Docker image based on the Dockerfile and installs necessary Python packages.

### ****Step 3: Set Up Airflow Configuration****

1. **Initialize Airflow**: To initialize Airflow’s configurations and database, use the following command:

docker-compose up airflow-init

1. **Start Airflow Services**: Once initialized, run Airflow and its scheduler in detached mode:

docker-compose up -d

**Step 4: Run the Airflow DAG**

After Airflow is running, access the **Airflow UI** by visiting http://localhost:8080 in your browser.

* **Enable DAG**: Find the brewery\_data\_pipeline DAG, switch it on, and trigger it manually or wait for it to execute on schedule.

**Step 5: View the Results**

The pipeline processes the data through the following steps:

1. **Raw Data** stored in /app/bronze/raw\_breweries.json.
2. **Transformed Data** stored in Parquet format in /app/silver/breweries.parquet.
3. **Aggregated Data** stored in /app/gold/aggregated\_breweries.parquet.

You can check the /app directory within the Docker container or configure these to output to a data lake on the cloud.

## **Explanation of Each File**

### ****Docker Configuration****

* **Dockerfile**: Configures the image with dependencies (Python, PySpark, Airflow).
* **requirements.txt**: Lists all necessary Python packages.

### ****Source Code****

* **0\_fetch\_breweries.py**: This script fetches data from the Open Brewery DB API.
* **1\_store\_raw\_data.py**: Stores raw data in JSON format for the Bronze layer.
* **2\_transform\_data.py**: Transforms data to Parquet, partitioned by location for the Silver layer.
* **3\_aggregate\_data.py**: Aggregates data by brewery type and location, producing analytics in the Gold layer.

### ****Airflow DAG****

* **brewery\_data\_pipeline\_dag.py**: The Airflow DAG orchestrates the entire workflow, defining task dependencies from ingestion to aggregation.

### ****Testing****

* **test\_breweries.py**: A simple unit test to validate the data fetched from the API.

## **Running the Tests**

1. **Run Tests**: From the root project directory, execute the unit tests using Python's unittest framework.

python -m unittest discover test

1. **Review Test Results**: The tests should verify the data fetched from the API and other functions as needed.

## **Monitoring and Alerting**

To implement monitoring and alerting, you could use the following steps:

1. **Airflow Monitoring**:
   * **Task Logging**: Each task in Airflow logs its execution details, which can be viewed in the Airflow UI.
   * **Retries and Alerts**: Configure email alerts and retries within the DAG’s default arguments to receive notifications on failures.
2. **Data Quality Checks**:
   * **Column Validations**: Add data quality checks within transform\_data.py to ensure each record meets schema requirements.
   * **Aggregated Counts**: Check that the Gold layer output includes values for each expected type and location.

## **Common Troubleshooting**

* **Docker Permission Errors**: Ensure Docker has the necessary permissions. You may need to use sudo with Docker commands if encountering permission issues.
* **Airflow Scheduling Issues**: If Airflow tasks aren’t running as expected, verify the DAG is enabled and try reloading the Airflow webserver.

## **Future Enhancements**

1. **Cloud Data Lake Integration**: Extend the current storage paths to point to an S3 bucket, Google Cloud Storage, or Azure Data Lake.
2. **Additional Transformations**: Enhance the Silver layer transformations by performing data cleaning and adding derived columns.
3. **Advanced Monitoring**: Integrate with a monitoring service like **Prometheus** or **CloudWatch** for additional metrics and alerts.
4. **Kubernetes Deployment**: Deploy the pipeline in Kubernetes for greater scalability and better resource management.

## Derivables

* **GitHub Repository**: Codebase, Apache Airflow Dags Scripts, Docker files , bash files, test scripts and README documentation.
* **Documentation**: Diagrams of data flow, explanation of transformations, and automation approach.

This pipeline solution provides a scalable, automated data pipeline capable of handling growing data volumes, making it easy to generate insights and reports for the e-commerce business.