BEES Data Engineering – Breweries Case

Tech Challenge - Data Engineer - AB-InBev | Bees

Autor: Aldenis França

Data: 21/08/2025

1. Visão Geral

O projeto Open Brewery Medallion implementa um pipeline de dados baseado no modelo de arquitetura Medallion (Bronze, Silver e Gold). Ele consome dados da API pública Open Brewery (https://www.openbrewerydb.org/), realiza processamento e aplica validações de qualidade antes de armazenar os dados em camadas organizadas.

O Open Brewery DB é um conjunto de dados e API gratuitos com informações públicas sobre cervejarias, sidrerias, cervejarias artesanais e lojas de bebidas de diversos países do mundo.

Neste projeto, o pipeline é orquestrado pelo **Apache Airflow**, com transformações em **PySpark** e persistência em formatos otimizados para análise, como **json** e **parquet**. Os testes são realizados com a biblioteca **pytest**.

2. Estrutura do Projeto

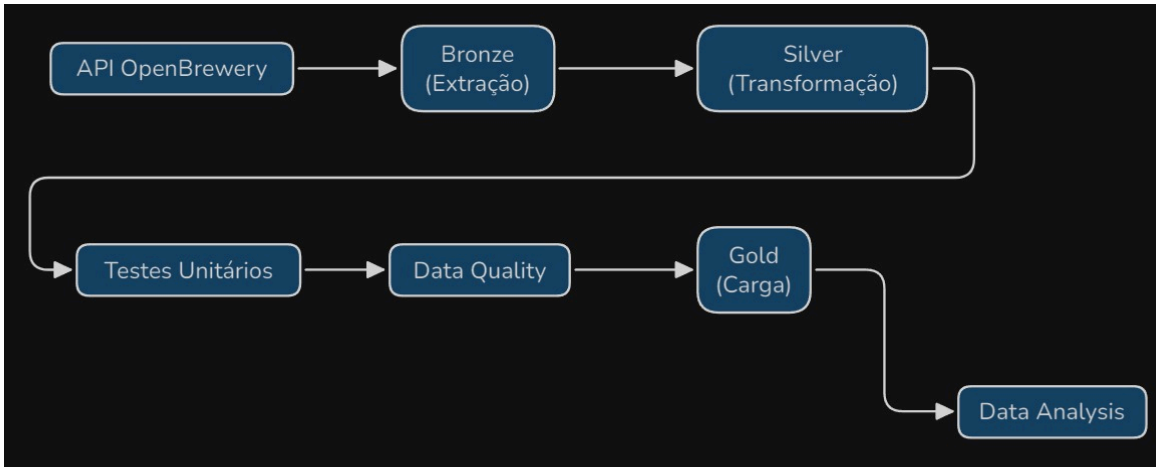
O repositório deste projeto possui a seguinte estrutura de arquivos e pastas. A seguir, há o detalhamento sobre os processos que cada arquivo desse executa.

```
openbrewery-medallion/
├── requirements.txt             # Dependências do projeto
├── dags/
│   └── brewery_etl_dag.py       # DAG do Airflow para orquestração do pipeline
├── logs/
│   └── pipeline.log             # Arquivo de logs da execução
├── scripts/
│   ├── data_analysis.py         # Script para análise exploratória dos dados
│   ├── extract.py               # Extração de dados da API OpenBrewery
│   ├── load.py                  # Carregamento para camada destino
│   ├── quality.py               # Validações de qualidade dos dados
│   ├── transform.py             # Transformações e limpeza dos dados
│   └── utils.py                 # Funções utilitárias (logger, conexões, etc.)
├── tests/
│   └── test_transform.py        # Testes unitários das transformações
```

a. **dags/brewery_etl_dag.py** → Define a DAG do Apache Airflow que orquestra as etapas do pipeline.
b. **scripts/extract.py** → Responsável pela extração de dados da API OpenBrewery.
c. **scripts/transform.py** → Realiza transformações, limpeza e padronização dos dados.
d. **scripts/quality.py** → Aplica validações de qualidade e consistência.
e. **scripts/load.py** → Carrega os dados tratados nas camadas alvo.
f. **scripts/data_analysis.py** → Exibe uma análise exploratória dos dados extraídos.
g. **scripts/utils.py** → Funções auxiliares (logger, conexões, etc.).
h. **tests/test_transform.py** → Testes unitários garantindo a consistência das transformações.

3. Fluxo do Pipeline

O pipeline de dados deste projeto segue o seguinte fluxo:



a. **Bronze (Extração):** Coleta dados brutos da API OpenBrewery.
b. **Silver (Transformação):** Limpeza, padronização e tratamento de nulos.
c. **Testes Unitários:** Verificação do correto funcionamento das funções.
d. **Data Quality:** Validação de schema e de consistência dos dados.
e. **Gold (Carga):** Armazenamento dos dados tratados para análise.
f. **Data Analysis:** Scripts auxiliares permitem exploração e relatórios.

4. Arquitetura Medallion (Medalhão)

A Arquitetura Medalhão é um padrão para organizar dados em um lago de dados (Data Lake), estruturando-os em três camadas progressivas de qualidade e refinamento. Neste projeto em específico, os dados foram armazenados desta forma nas camadas:

- Bronze (raw): Dados brutos em JSON.
- Prata (curated): Dados normalizados em Parquet, particionados por país e estado (localização), com validações de qualidade.
- Ouro (analytical): Agregações analíticas por tipo de cervejaria e por país e estado (localização).

O objetivo é melhorar gradualmente a qualidade dos dados à medida que eles fluem pelas camadas, permitindo maior governança, escalabilidade e o uso de dados para análise, relatórios e machine learning.

5. Data Quality

As seguintes regras foram implementadas em `scripts/quality.py`:

a. **Schema esperado:** o adequado tipo de dado para cada coluna do conjuntos de dados.
b. **Não-nulos:** as colunas-chaves id e name não podem ser nulas.
c. **Ranges válidos:** latitude e longitude dentro dos intervalos corretos → latitude ∈ [-90, 90], longitude ∈ [-180, 180].
d. **Duplicatas:** garantir a unicidade do id de cada linha.
e. **Dataset não vazio:** verificar se existem dados no dataset da camada curated.

Se ocorrer de dar erro em algum desses itens, é gerado um alerta persistente, o qual interrompe a execução do pipeline no Airflow.

6. Dependências

O arquivo **requirements.txt** contém as bibliotecas necessárias para execução do projeto. Para instalá-las, utilize o comando:

```
python pip install -r requirements.txt
```

7. Logs

Os logs de execução do pipeline são armazenados em `logs/pipeline.log`, permitindo auditoria e depuração. Eles são gerados utilizando a biblioteca logging do Python, com suas diversas funcionalidades.

8. Testes Unitários

Os Testes Unitários estão localizados no diretório `tests/`. Eles garantem que as transformações de dados funcionem corretamente, executando validações dessas funções. Foram adicionados os seguintes testes unitários com Pytest para garantir as validações:

a. Identificação e remoção de colunas com dados iguais.
b. Detecção de coordenadas inválidas.
c. Regras de consistência geográfica.

Para executar os testes unitários, utilize o comando:

```
pytest tests/
```

9. Execução no Airflow

A DAG do Airflow (`dags/brewery_etl_dag.py`) possui as seguintes tasks:

- extract_bronze
- transform_silver
- run_transform_tests
- run_quality_tests
- load_gold
- run_data_analysis

Trecho adicionado na DAG:

```
run_transform_tests = BashOperator(
    task_id='run_transform_tests',
      bash_command='pytest  ../tests/test_transform.py  >>  /opt/airflow/logs/test_transform.log
2>&1'
)
extract_bronze >> transform_silver >> run_transform_tests >> run_quality_tests >> load_gold >>
run_data_analysis
```

Desta forma, se qualquer teste unitário falhar, a DAG interrompe a execução antes de publicar os dados na camada Gold.

10. Monitoramento e Alerta

O Data Quality foi aplicado à camada Silver, com tarefas de validação no PySpark, já que é o ponto único de verdade para a camada Gold.

Na DAG Airflow, caso ocorra falha na execução, foram aplicadas *retries* para executar novamente o código antes de realizar as notificações, as quais são enviadas via e-mail, enviando os Logs Airflow por task e as métricas de duração.

11. Autor

Projeto desenvolvido por Aldenis França.