Documentação do Pipeline Airflow - Open Brewery

🔁 Fluxo Geral do Pipeline

O pipeline segue a arquitetura em camadas Bronze → Silver → Gold. Utiliza uma DAG no Airflow para orquestrar três tarefas principais:

1. Bronze: Extração de dados da API Open Brewery.
2. Silver: Transformação e padronização em Parquet particionado por estado.
3. Gold: Agregações e geração de três arquivos CSV com insights.

🥉 Camada Bronze - Extração

A função `extract_bronze_data` coleta todos os dados da API Open Brewery usando paginação. Primeiro consulta o endpoint

`https://api.openbrewerydb.org/v1/breweries/meta`

para descobrir o total de registros. Em seguida, percorre todas as páginas usando os parametros e página, e usando `per_page=200` para limitar o número de chamadas na API, respeitando o limite da API. Há um delay de 1 segundo entre as requisições.

Se alguma chamada falhar, a URL é armazenada para tentativa posterior, serão realizadas uma segunda tentativa apenas das URL que falharam para otimização. Ao final, todos os dados são salvos em um único arquivo JSON na pasta `/opt/airflow/data/bronze`.

🥈 Camada Silver - Transformação

A função `transform_to_silver` lê o JSON bruto da camada Bronze e transforma em DataFrame com colunas padronizadas. Realiza validação das colunas esperadas, conversão de tipos e tratamento de campos nulos.

Os dados são salvos em formato Parquet particionado por estado na pasta `/opt/airflow/data/silver/breweries`. Cada partição é criada como `state=NomeDoEstado`.

🥇 Camada Gold - Agregações

A função `aggregate_gold` é responsável por consolidar e agregar os dados que foram transformados e particionados na camada Silver. Esses dados estão armazenados no formato Parquet e organizados por estado na pasta `data/silver/breweries`.

O processo inicia localizando todos os arquivos `.parquet` dentro da estrutura particionada (`state=XX/breweries.parquet`). Cada um desses arquivos representa os dados de cervejarias de um determinado estado. A função então realiza a leitura desses arquivos em um loop, convertendo cada um em um `DataFrame` pandas. Todos os DataFrames lidos com sucesso são concatenados em um único `DataFrame` unificado.

Com os dados unificados em memória, são executadas três agregações distintas:

1. **Por tipo de cervejaria**
   Agrupa os dados pela coluna `brewery_type` e conta quantas cervejarias existem de cada tipo.
   Resultado salvo em: `brewery_count_by_type.csv`
2. **Por estado**
   Agrupa os dados pela coluna `state` para contar quantas cervejarias existem por estado.
   Resultado salvo em: `brewery_count_by_state.csv`
3. **Por tipo de cervejaria e estado**
   Agrupamento duplo por `brewery_type` e `state`, retornando a contagem de cada tipo de cervejaria por estado.
   Resultado salvo em: `brewery_count_by_type_and_state.csv`
4.

Todos os arquivos gerados são salvos na pasta `data/gold`, completando assim a camada Gold do pipeline.

Além disso, a função possui tratamento de erros detalhado: se algum arquivo Parquet não puder ser lido, o erro é logado e o processo segue com os arquivos restantes. O mesmo vale para problemas nas etapas de agregação e escrita dos arquivos. Esse controle garante a resiliência do pipeline, mantendo a execução mesmo que ocorram falhas pontuais.

🚨 Tratamento de Erros

Todas as funções estão protegidas por blocos `try/except`. Em caso de falha, logs são gerados com `logging.error()` e mensagens detalhadas são armazenadas usando `traceback.format_exc()`. Isso garante que falhas em partes isoladas do processo não interrompam a execução completa.

Exemplos de erros tratados:
- Falha na chamada de API
- Problemas na leitura/escrita de arquivos Parquet
- Dados ausentes ou colunas inconsistentes

🧩 DAG do Airflow

A DAG é definida com o ID `brewery_pipeline_bronze_silver_gold`, desativando `catchup` e com agendamento automático 1 vez ao dia. (`schedule_interval=@daily`).
As tasks são:

- `extract_bronze_data`
- `transform_to_silver`
- `aggregate_breweries`

A ordem de execução é sequencial: Bronze → Silver → Gold.

📘 Anexo Técnico - Projeto Breweries Case

📘 Documentação Técnica - Projeto Breweries Case

🚀 Como Iniciar o Projeto

1. Clone o repositório:
```
git clone git@github.com:mateusf97/breweries_case.git
cd breweries_case
```

2. Dê permissão de leitura e escrita para as pastas essenciais:
```
chmod -R 777 ./logs ./data ./dags
```

3. Suba o projeto com Docker Compose (no Linux):
```
docker compose up --build
```

4. Acesse a página web em:
```
http://localhost:8088
```

Login padrão:
```
Usuário: Admin
Senha: admin
```

📂 Arquitetura do Pipeline

1️⃣ Bronze Layer
- Coleta os dados da API Open Brewery com paginação.
- Armazena os dados brutos em JSON.
- Se falhar, tenta novamente ao final.

Trecho de código:
```python
for url in failed_urls:
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        all_data.extend(data)
        logging.info(f"🔁 Retry bem-sucedido para: {url}")
    except Exception as e:
        logging.error(f"❌ Falha permanente em: {url}")
```

2️⃣ Silver Layer
- Converte o JSON em Parquet.
- Particiona por estado.
- Tipagem e limpeza de dados.

3️⃣ Gold Layer
- Agrega os dados da Silver.
- Gera 3 arquivos CSV:
  - brewery_count_by_type.csv
  - brewery_count_by_state.csv
  - brewery_count_by_type_and_state.csv

📊 Página Web
Criei um mapa utilizando HTML, JavaScript e CSS que consome os dados processados e gera uma visualização interativa. Ele está disponível dentro da pasta `html_view`, no arquivo `.html`.

Acesso:
```
html_view
```

📬 Sistema de Monitoramento

Airflow gera logs e envia e-mails de falha.

Exemplo de log:
```bash
❌ Erro ao ler arquivo:
/opt/airflow/data/silver/breweries/state=XX/breweries.parquet
```

🧠 Tecnologias Utilizadas

- Airflow
- Python
- PySpark / Pandas
- Parquet
- Docker
- Streamlit
- Git

📈 Camadas Adicionais no GOLD

Além das agregações principais, temos também:
- brewery_count_by_type.csv
- brewery_count_by_state.csv
- brewery_count_by_type_and_state.csv

🗂️ Estrutura de Diretórios

```
.
├── data/
│   ├── bronze/
│   ├── silver/
│   └── gold/
├── dags/
├── html_view/
└── docker-compose.yml
```