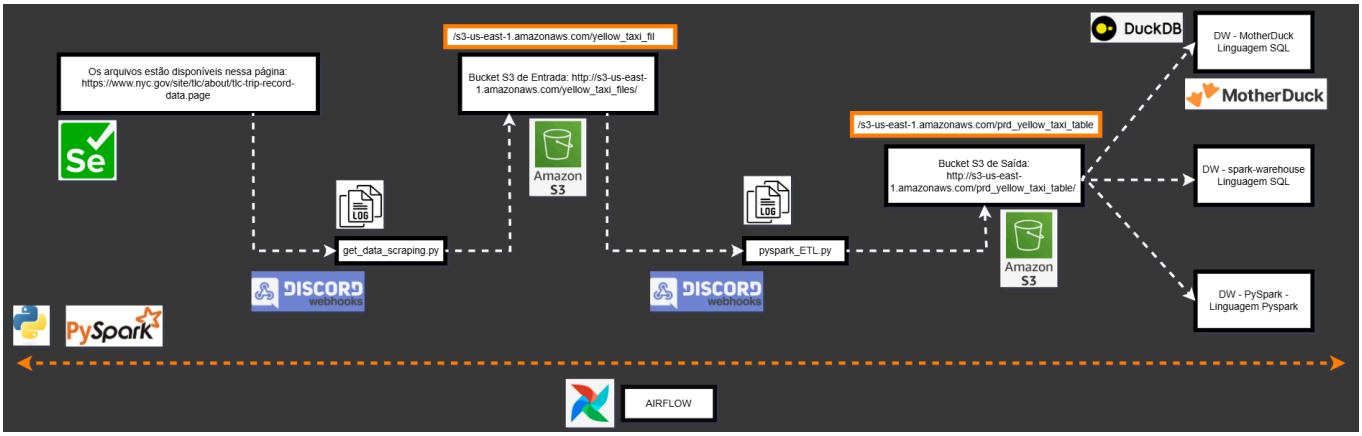
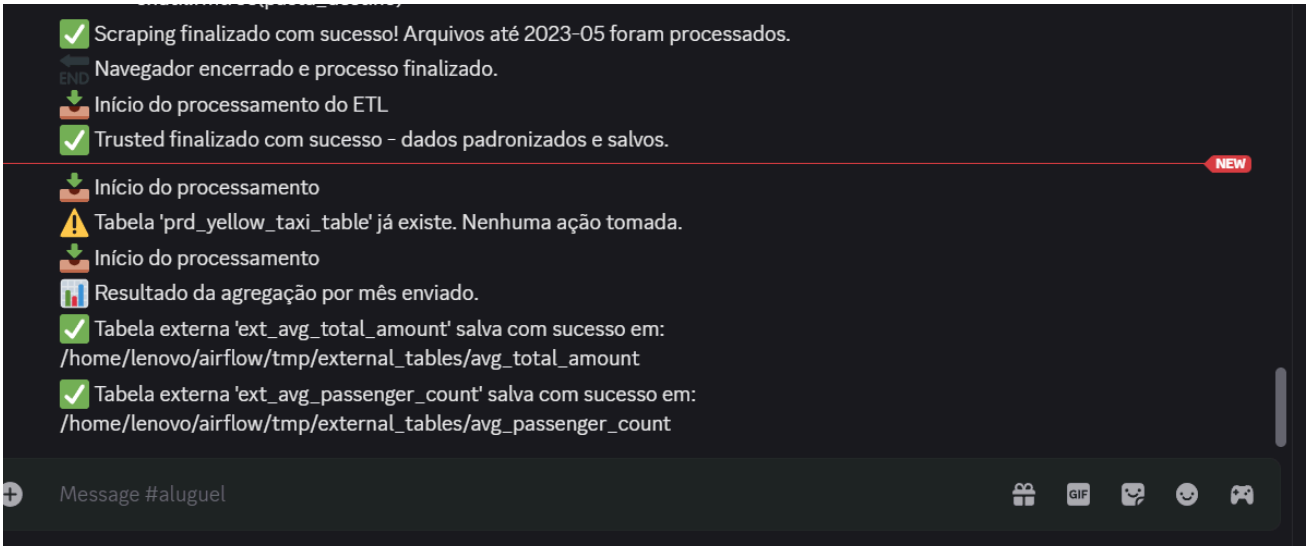
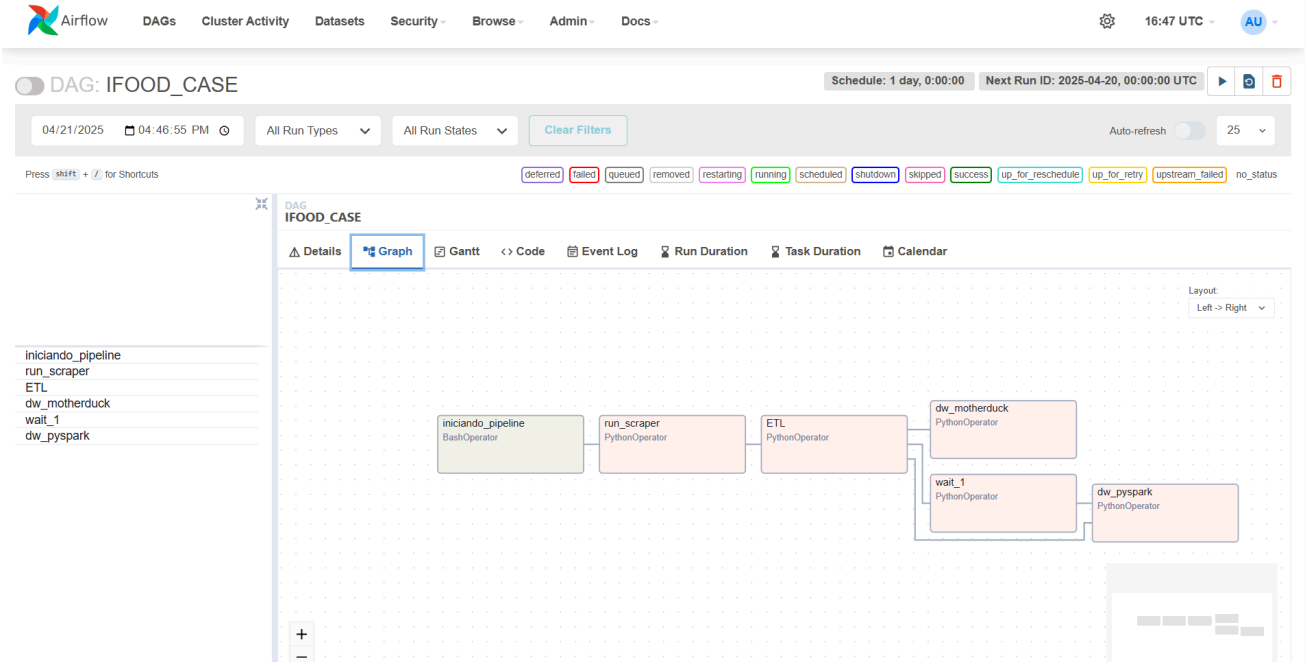


- Para esse caso, foram utilizadas as linguagens PySpark e SQL.
- Para fins de monitoramento, utilizou-se **logging** e **webhook** (via Discord), permitindo acompanhar os pipelines e detectar possíveis falhas.
  Exemplo de monitoramento pelo webhook no discord:



- O Airflow foi utilizado como orquestrador.



  Foram implementados três casos de Data Warehouse (DW):
  1. Um DW utilizando **MotherDuck**, com linguagem SQL.
  2. Um DW utilizando **spark-warehouse**, com linguagem SQL, dentro do próprio metastore do Hive.
  3. Um DW como **tabela externa fora do Hive metastore**, utilizando linguagem PySpark.
- Para fins de **reaproveitamento de código** e **modularização**, foram implementados e utilizados os seguintes módulos, com importações seletivas de funções específicas:

```
from jobs.log_utils.logging_function import *
(Funções de notificação, logging e webhook do discord)

from jobs.s3.s3_functions import *
(Funções de leitura e de salvamento dos dados para dentro dos buckets do S3)

Arquivo .env com o TOKEN do motherduck
```

Esses módulos centralizam funcionalidades reutilizáveis, como logging customizado, Webhook (via Discord), operações com o S3 e integração com PySpark, promovendo maior organização e reutilização do código no projeto.

✅ Explicação da Solução Implementada

Os dados foram obtidos a partir do site oficial da prefeitura de Nova York: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Para realizar o download automático dos arquivos **yellow_tripdata** no formato **Parquet**, foi desenvolvido um **bot de scraping** utilizando a biblioteca **Selenium**. Os arquivos foram

armazenados diretamente em um **bucket S3 simulado (mocking)**, na pasta: `s3://s3-us-east-1.amazonaws.com/yellow_taxi_files/` (região `us-east-1`).

O processo de ingestão foi monitorado por meio da biblioteca **logging** e de um **webhook** (via Discord), permitindo o acompanhamento em tempo real. Toda a orquestração do pipeline foi realizada utilizando o **Apache Airflow**.

Com os dados armazenados no bucket, foi realizado um processo de **ETL (Extração, Transformação e Carga)** para tratamento e organização dos registros. Durante a etapa de transformação, foram extraídos os campos de **mês e ano** a partir da coluna `tpep_dropoff_datetime`.

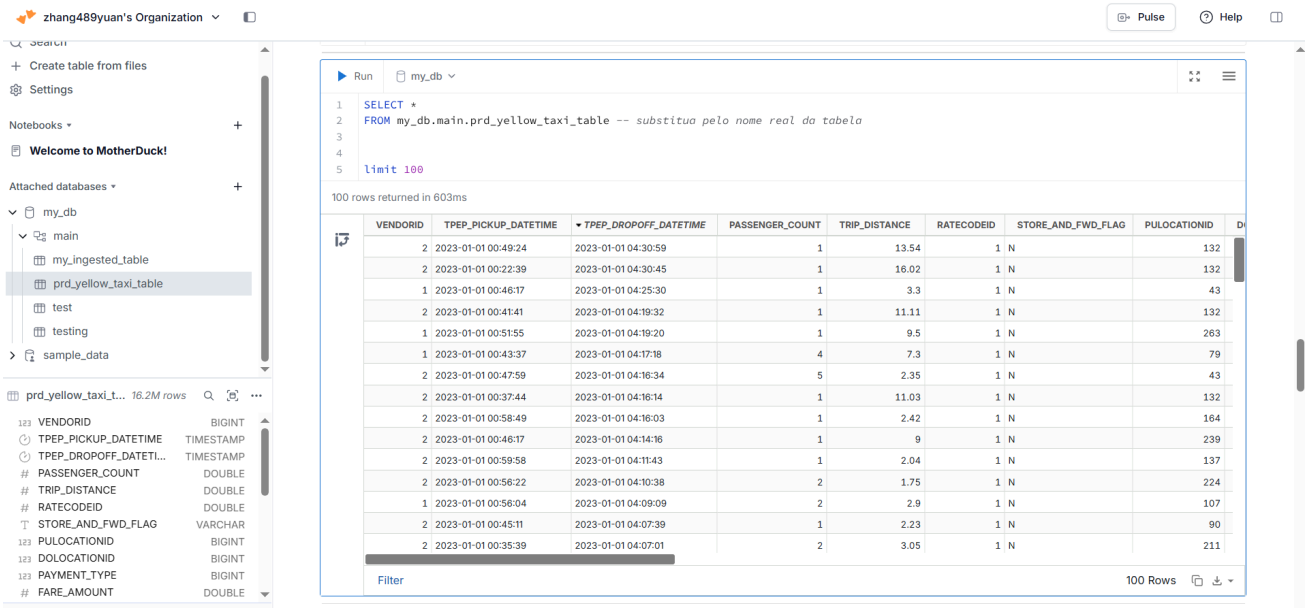
Durante a análise inicial, identificou-se a presença de dados **inconsistentes**, como registros com anos incorretos (ex.: 2008, 2009), que não fazem parte do escopo atual do projeto. Por esse motivo, foi aplicado um **filtro** para considerar **apenas os dados do ano de 2023**, garantindo maior qualidade e relevância na análise.

Com o **sistema de monitoramento implementado via `logging` e webhook (Discord)**, e com a **orquestração feita pelo Apache Airflow**, os dados tratados foram direcionados para o ambiente de **produção** em um bucket S3 simulado (mocking), no seguinte caminho: `s3://s3-us-east-1.amazonaws.com/prd_yellow_taxi_table/` (região `us-east-1`), simulando uma estrutura real de armazenamento em nuvem.

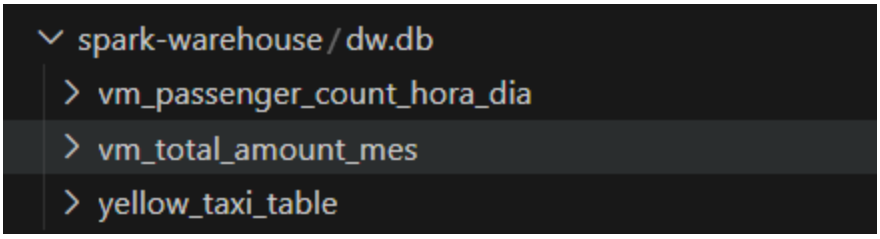
Com os dados armazenados no bucket `s3://s3-us-east-1.amazonaws.com/prd_yellow_taxi_table/`, visando maior **versatilidade na análise e exploração dos dados**, foram criadas três estruturas distintas de **Data Warehouse (DW)**:

1. **DW com MotherDuck (DuckDB):**
   Os dados foram inseridos em um banco **MotherDuck**, utilizando o **DuckDB** como

mecanismo de consulta, aproveitando sua leveza e compatibilidade com SQL.



2. **DW gerenciado pelo metastore do Hive (via PySpark):**
   Criação de uma tabela gerenciada diretamente no metastore do Hive, utilizando comandos SQL via PySpark. Essa abordagem permite integração nativa com outras ferramentas que utilizam o Hive como catálogo de metadados.



3. **DW com tabela externa (external table via PySpark):**
   Criação de uma tabela externa em PySpark, apontando diretamente para os arquivos no bucket S3. Essa abordagem oferece flexibilidade ao permitir leitura dos dados sem depender do metastore do Hive.