Projeto - Engenharia de Dados

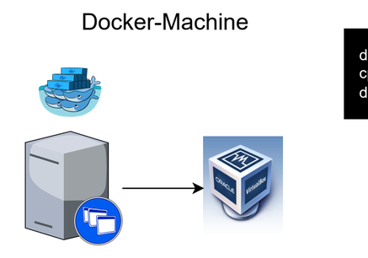
Objetivo do projeto: desenvolver uma estrutura de dados, que possibilidade a entrega dos dados com maior velocidade e assertividade para a equipe de ciência de dados.

Data: 22/12/2022

Arquitetura dos Dados

1 - Infraestrutura

Para o projeto piloto da arquitetura de dados, foi utilizado a solução Docker-machine e Docker para agilizar o desenvolvimento do laboratório de dados e experimentação do ambiente.



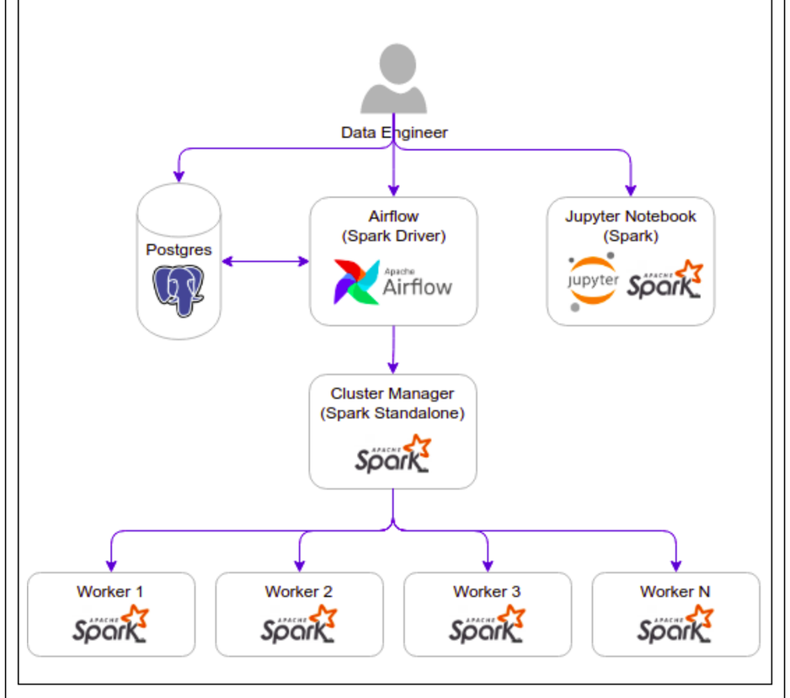

```
docker-machine create --driver virtualbox --virtualbox-cpu-count 4 --virtualbox-memory "4096" --virtualbox-disk-size "12000" engdados
```

```
Oscar@DESKTOP-S4VUVQM MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
engdados   -        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.12
```

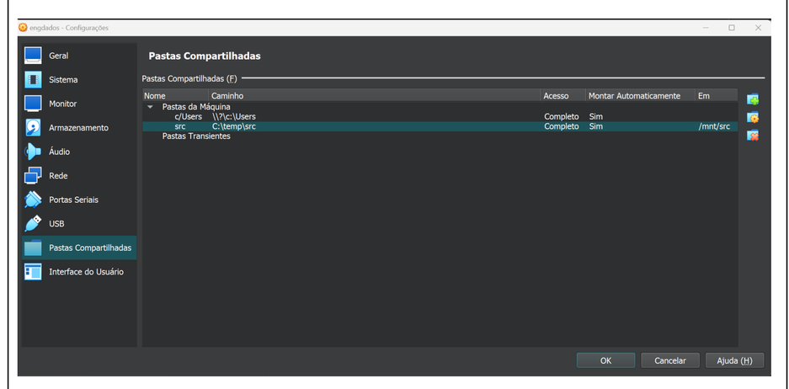
Docker - Airflow / Spark / Postgres

Para agilizar o processo de instalação dos softwares, procurei um projeto Docker com a infra que necessitava para o projeto piloto, gostei muito do projeto criado pelo "Thiago Cordon Rodrigues", onde ele traz a arquitetura ideal para o projeto com Airflow, spark e Postgres.

Github: https://github.com/cordon-thiago/airflow-spark



Criei uma pasta em meu PC e compartilhei via VirtualBOX, para deixar os dados permanentes nesta pasta.



Montei a pasta em: /mnt/src

sudo mount -t vboxsf -o defaults,uid=`id -u docker`,gid=`id -g docker` src /mnt/src

Clonando projeto

```
docker@engdados:/mnt/src$ git clone https://github.com/cordon-thiago/airflow-spark
Cloning into 'airflow-spark'...
remote: Enumerating objects: 1479, done.
remote: Counting objects: 100% (99/99), done.
remote: Compressing objects: 100% (64/64), done.
remote: Total 1479 (delta 49), reused 35 (delta 35), pack-reused 1380
Receiving objects: 100% (1479/1479), 221.17 MiB | 9.79 MiB/s, done.
Resolving deltas: 100% (534/534), done.
Checking out files: 100% (1090/1090), done.
```

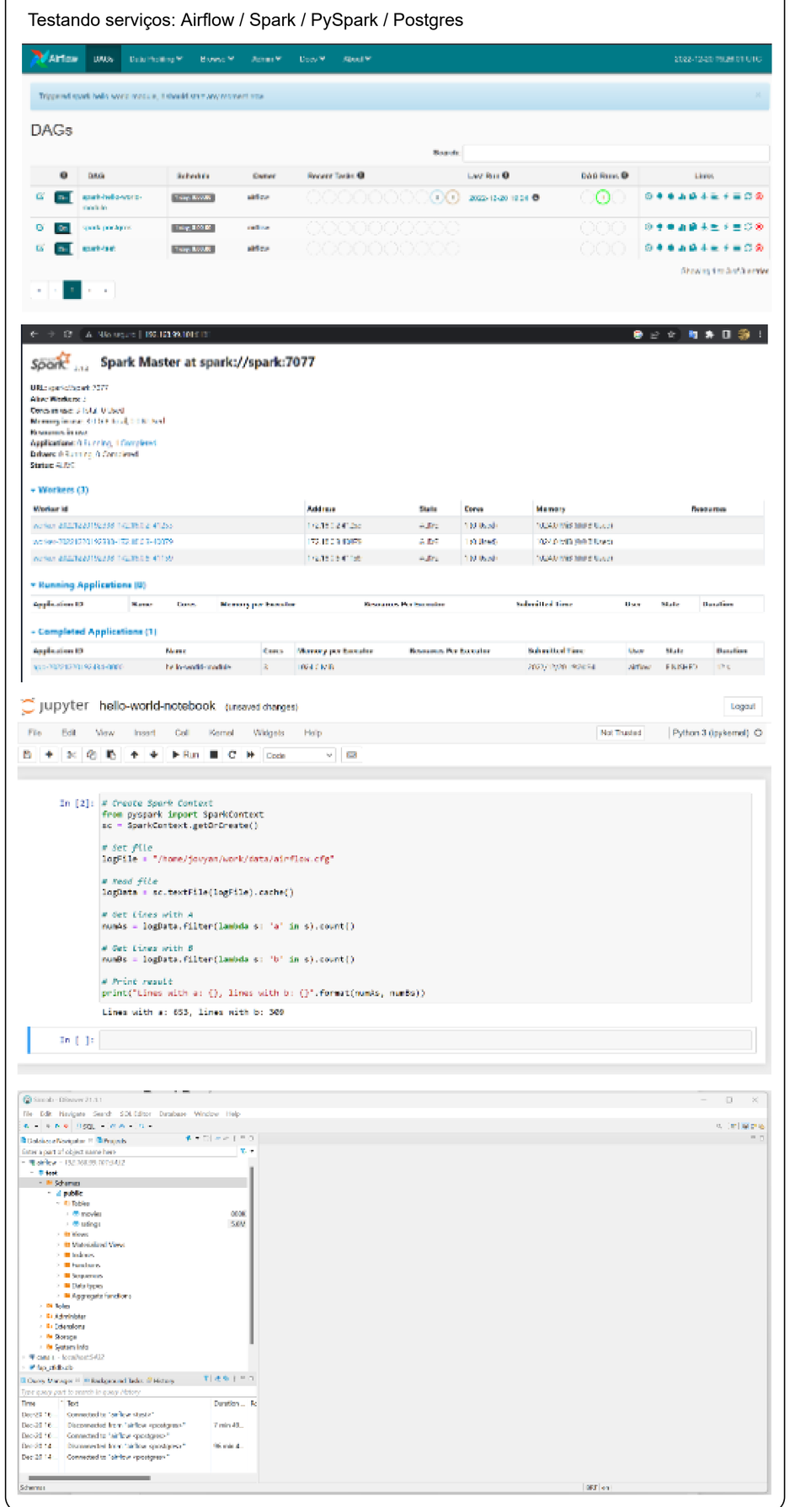
Build

```
docker@engdados:/mnt/src/airflow-spark/docker/docker-airflow$ sudo docker build --rm --force-rm -t docker-airflow-spark:1.10.7_3.1.2 .
Sending build context to Docker daemon  261.8MB
Step 1/34 : FROM python:3.6-stretch
3.6-stretch: Pulling from library/python
08224db8ce18: Already exists
abd3caf86f5b: Already exists
71c316554a55: Already exists
721081de66bf: Already exists
239fb482263d: Already exists
ec2643d613a1: Already exists
8973a3f2f69f: Already exists
```

docker-compose up

Setando variáveis do Docker-machine (engdados)

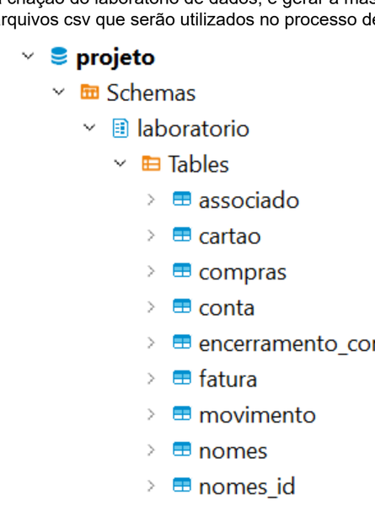
Testando serviços: Airflow / Spark / PySpark / Postgres



2 - Laboratório de dados

Para o desenvolvimento do projeto, foi criado um laboratório de dados no postgres, para geração da massa de dados que será utilizado no projeto.

O objetivo da criação do laboratório de dados, é gerar a massa de dados, para posteriormente exportar os arquivos csv que serão utilizados no processo de carga.



Obs: Como o objetivo do exercício não é criar um laboratório de dados, e sim focar no processo de carga dos dados, criei a massa de dados utilizando scritps SQL.

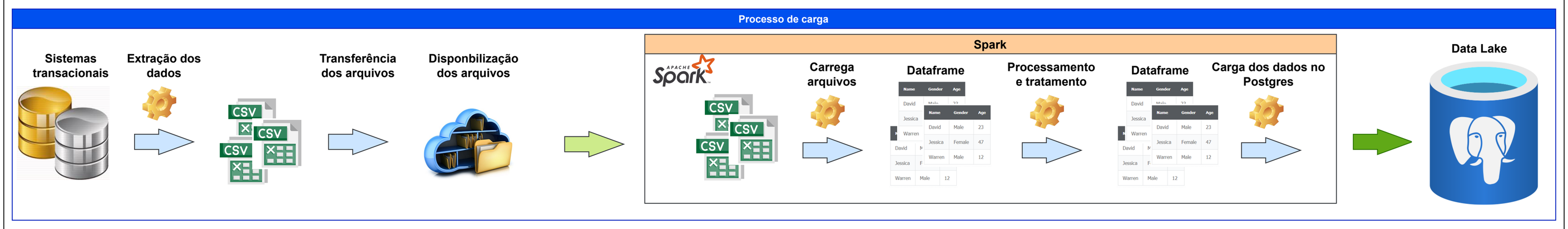
Todos os scripts SQL que foram utilizados para o desenvolvimento do laboratório de dados, está disponível no github do projeto.

3 - Arquitetura do processo de carga dos dados

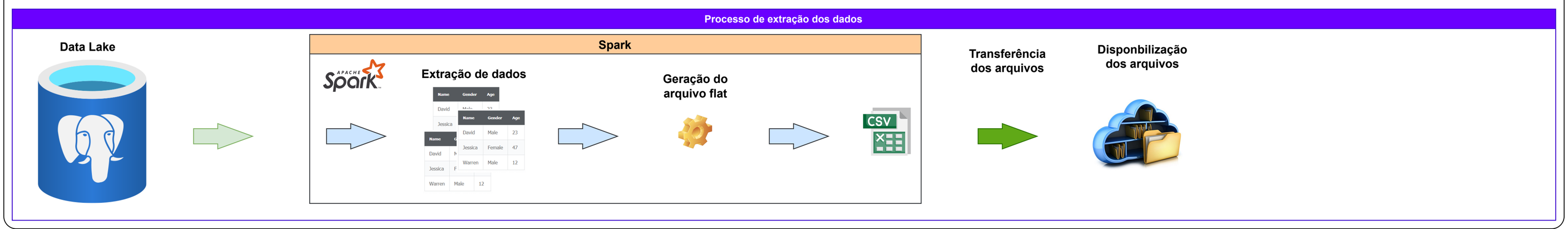
A arquitetura do processo de carga dos dados escolhido será o modelo ETL.

No cenário atual, onde os dados relacionais são armazenados em um banco de dados relacional como o postgres, talvez o modelo ideal seria o ELT, onde extrairíamos os dados em CSV, carregaríamos os dados diretamente no postgres utilizando os utilitários do postgres e todo o tratamento seria dentro do postgres.

A escolha do modelo ETL, é para exercitar o processamento de dados utilizando o motor de processamento do Spark.



O objetivo do exercício, é exercitar o processamento no Spark. A extração dos dados e relacionamentos entre os dados, serão processados no Spark.
Por essa razão, os dados não serão relacionados no postgres.

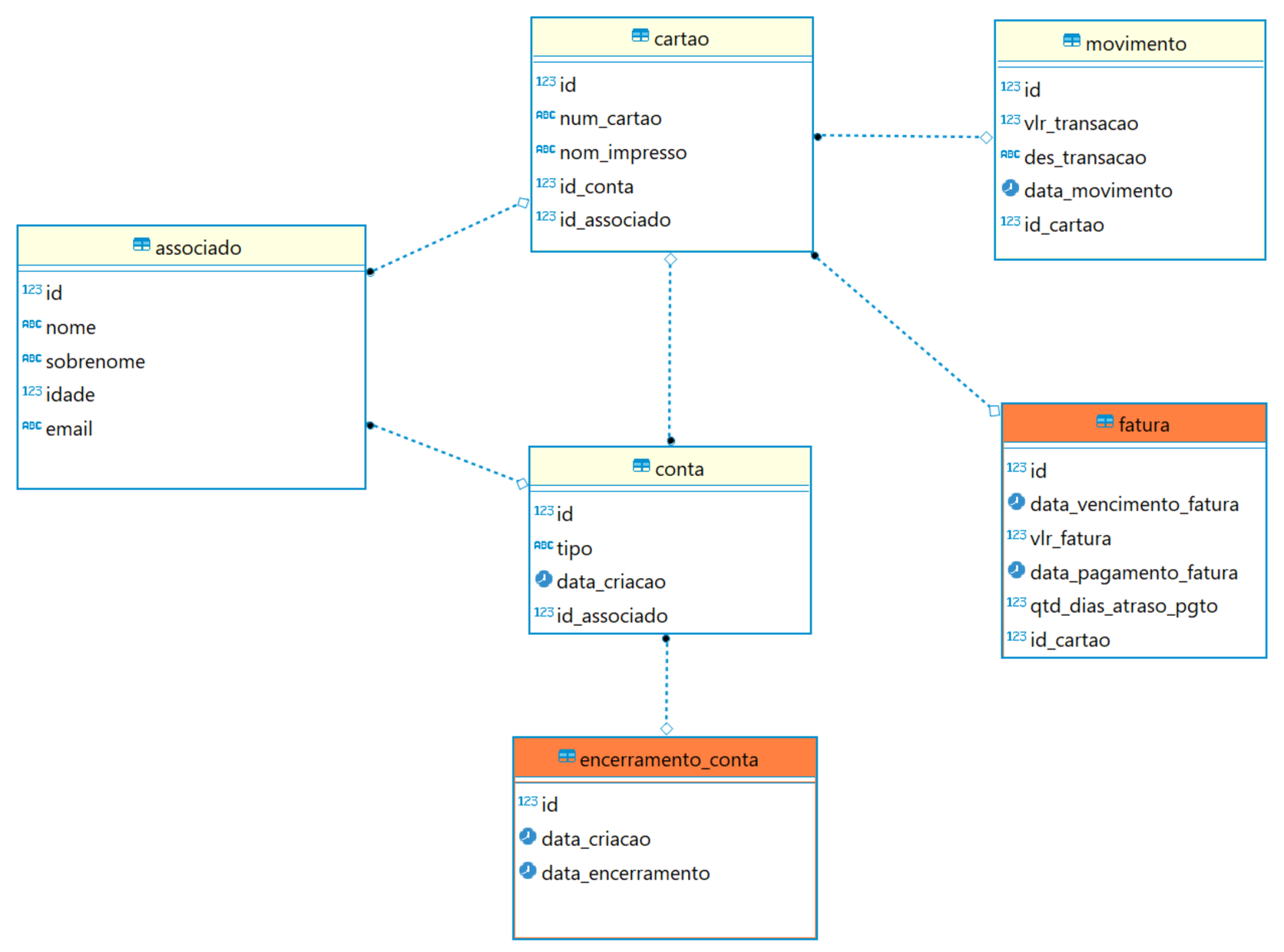


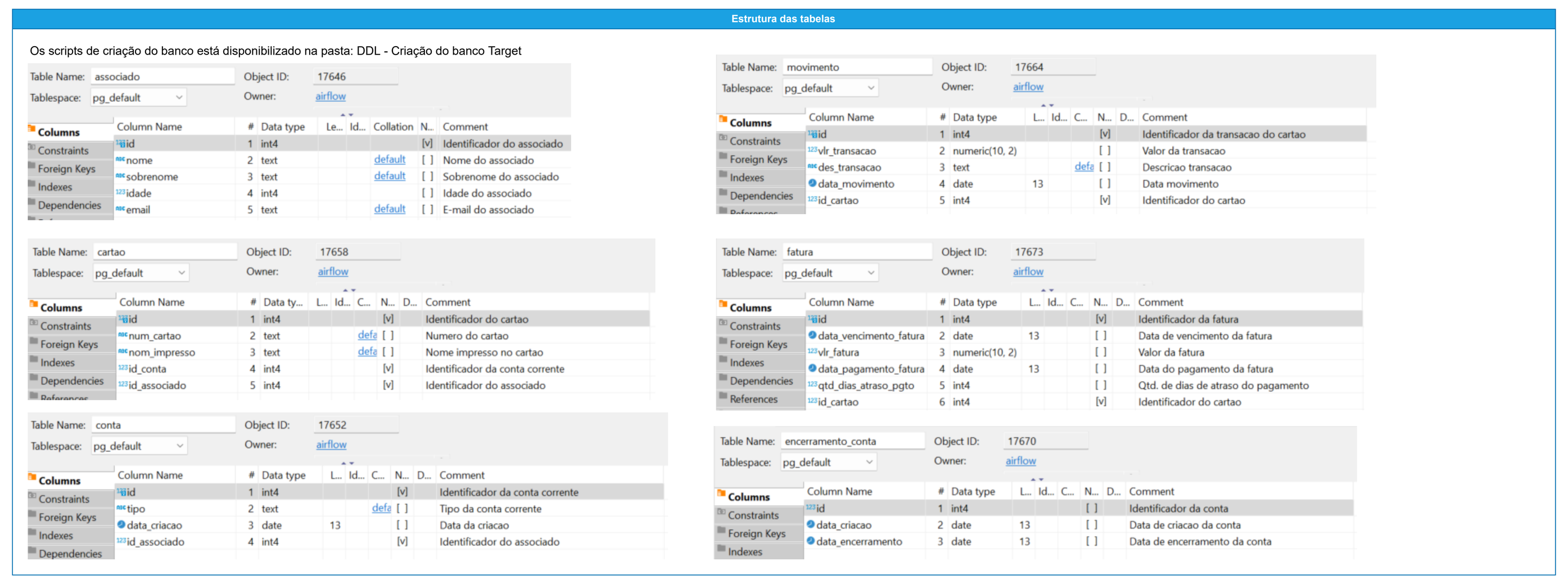
4 - Modelo dos dados no Data Lake (Postgres)

Para o desenvolvimento do projeto, foi desenhado o seguinte modelo de dados:

Obs: foi adicionado duas tabelas auxiliares (fatura e encerramento de conta), para acrescentar 2 cenários para o projeto.

Tabelas em amarelo: tabelas propostas no projeto.
Tabelas em laranja: tabelas para enriquecimentos da análise de inadimplência e análise de churn.

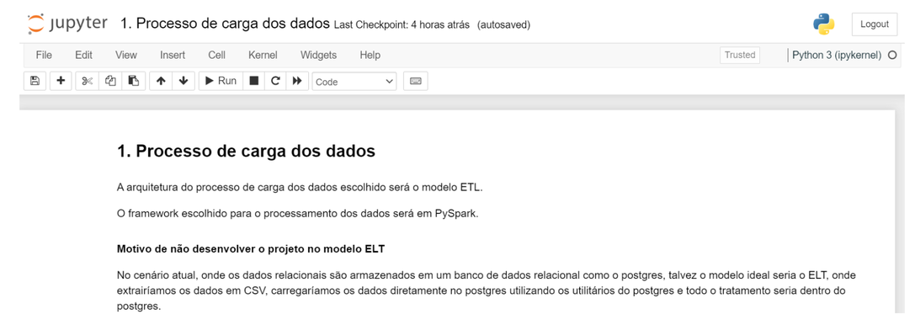




5 - Fluxo do processo de carga

O fluxo do processo de carga, apresenta de forma visual as etapas do processo de carga, facilitando o entendimento do funcionamento do processo de carga.

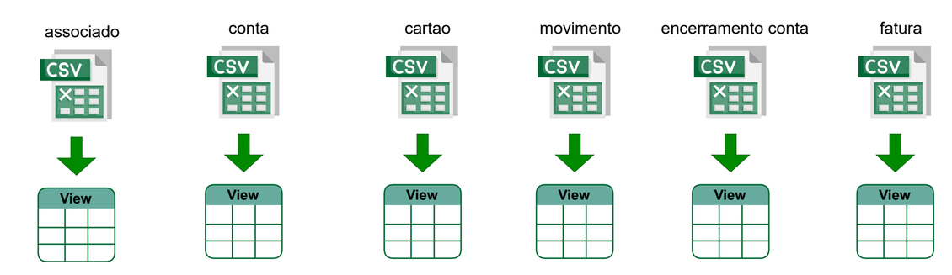
Obs: para facilitar o entendimento do código de carga, criei um notebook via Jupyter, que explica cada etapa do processo de carga, por essa razão resolvi não utilizar o airflow. O arquivo está disponível no github.



Processamento dos dados utilizando PySpark.

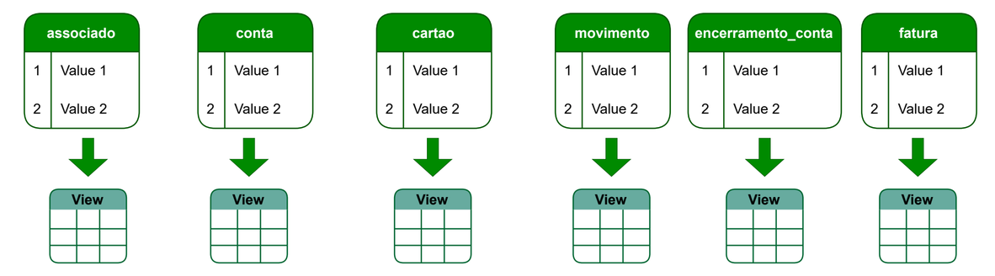
Passo 1 - Carregar e formatar arquivos CSV

Carregar os arquivos csv em dataframes, formatar os tipos dos dados e criar views dos dataframes para poder utilizar o spark.sql no tratamento de dados.



Passo 2 - Trazer dados do Data Lake (target)

Carregar tabelas do postgres em dataframes e criar views para disponibilizar os dados para tratamento e verificação se o registro já existe na tabela.

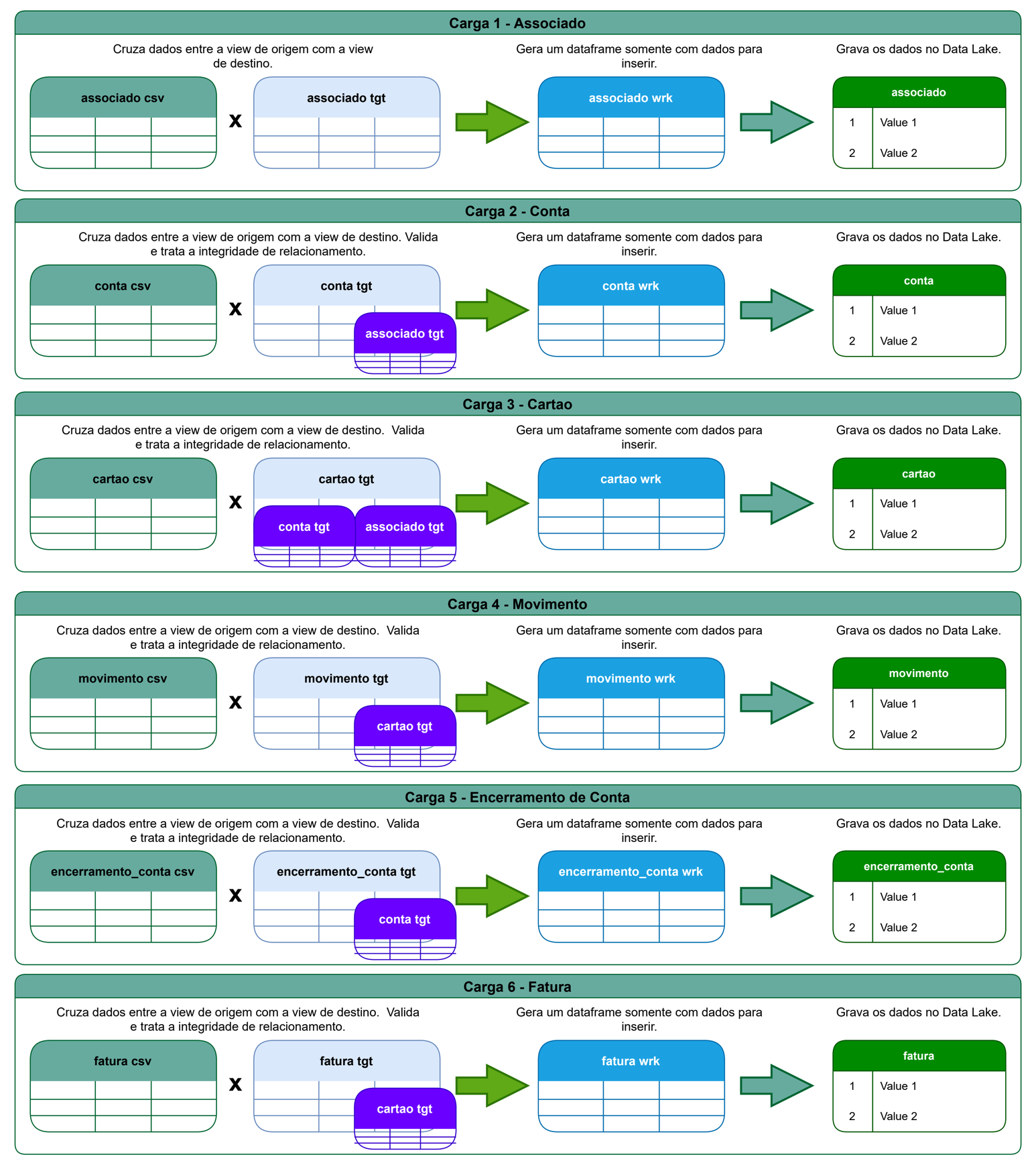


Passo 3 - Tratamento de dados e inserção dos dados no Data Lake

Cruza os dados entre as views dos dados das origens com os dados do target (origem Data Lake), verificando problemas de integridade de relacionamento.

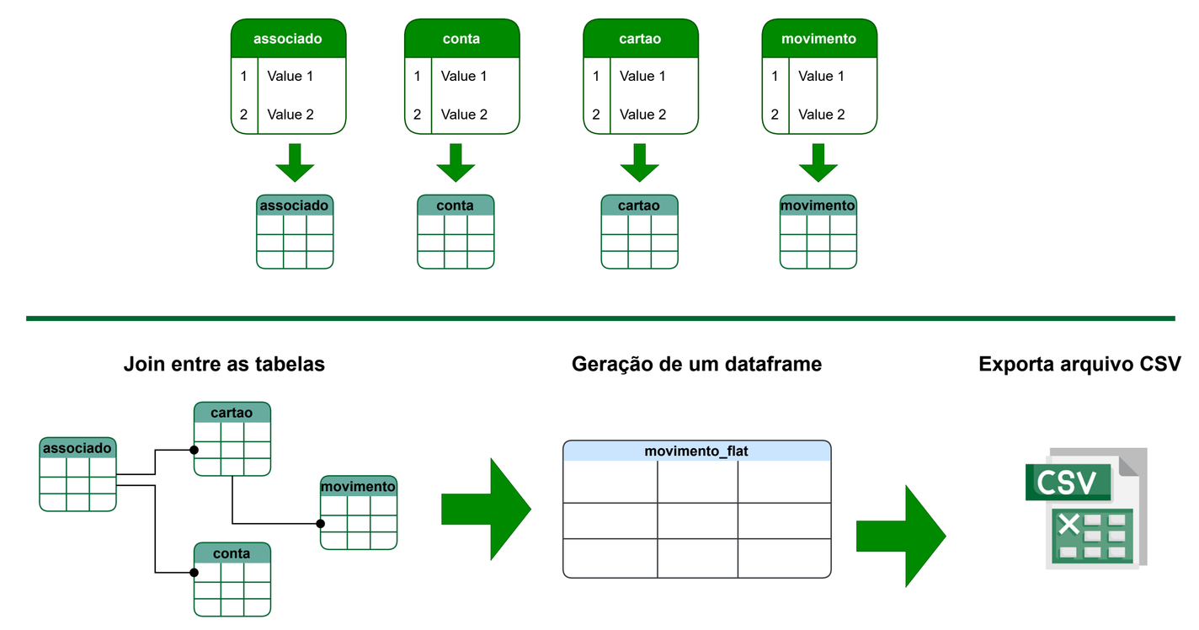
A estratégia de carga escolhido para este projeto, é inserir caso o dado no Data Lake caso o dado não exista, caso exista, ignorar a inserção do dado.

Obs: foi definido essa estratégia, devido ao curto prazo para entrega do projeto, um processo de atualização dos dados demandaria mais tempo de desenvolvimento. Devido ao tempo de desenvolvimento, os processos de carga ficaram em ordem sequencial.



Passo 4 - Geração do arquivo Flat (movimento_flat)

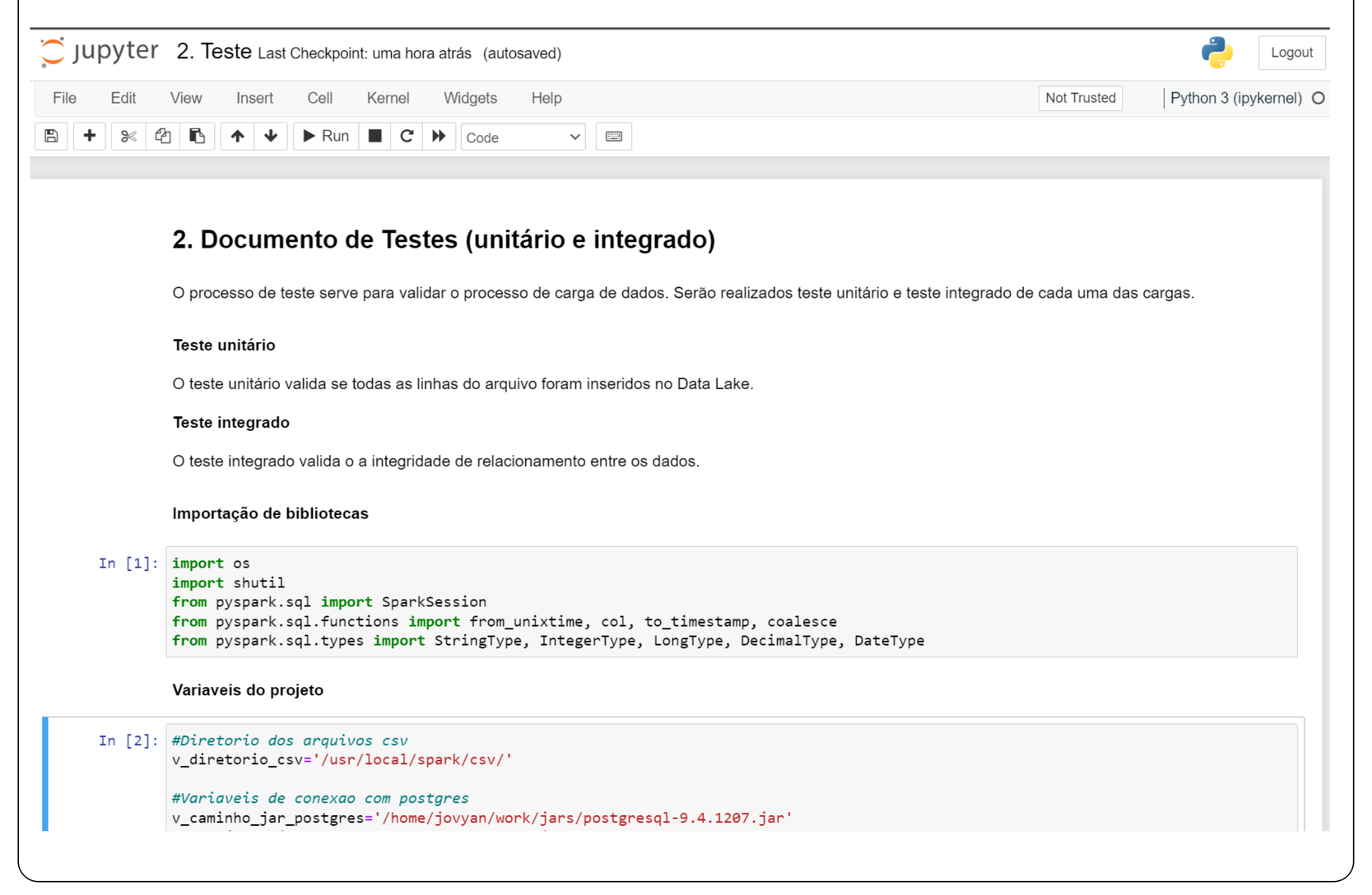
Carregar tabelas do postgres em dataframes e criar views para disponibilizar os dados para cruzamento de dados e criação de um dataframe para exportar o arquivo em csv.



6 - Testes

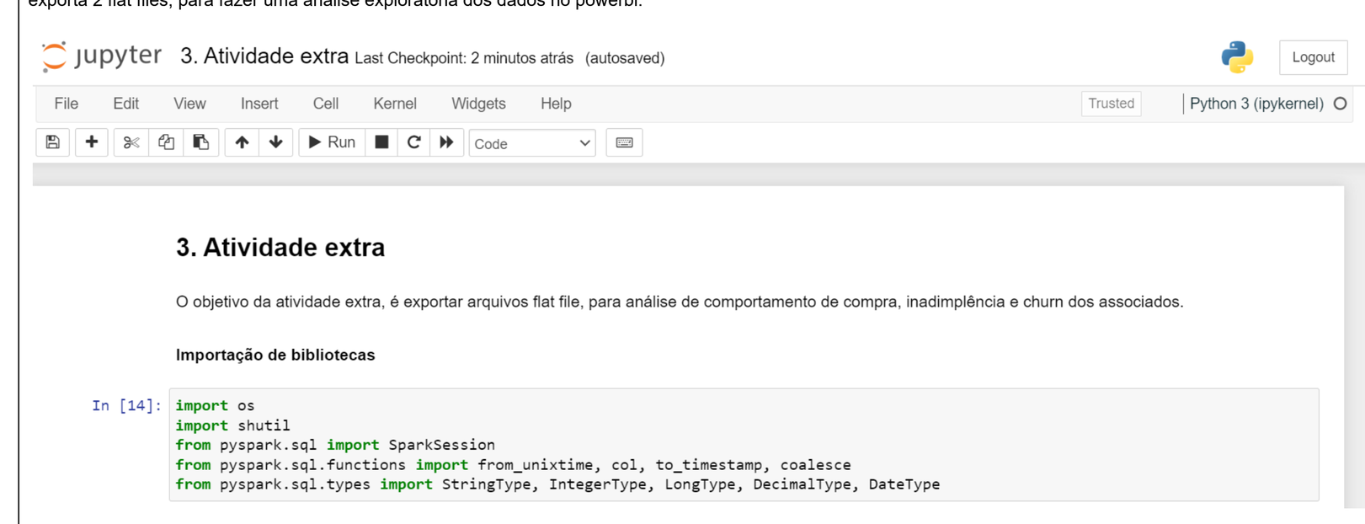
Foi realizado testes unitários e integrados para validar o processo de carga dos dados.

Obs: para facilitar o entendimento do código de carga, criei um notebook via Jupyter, que explica cada etapa do processo de teste. O arquivo está disponível no github.



Atividade Extra - Análise exploratória

Como atividade extra, para exercitar o processamento no datalake e utilizando o spark somente para gera o dataframe e gerar o arquivo csv, criei um processo que exporta 2 flat files, para fazer uma análise exploratória dos dados no powerbi.



Dificuldades do Projeto

Em relação à dificuldade técnica, normalmente utilizo o Docker em minha máquina pessoal Windows para testes e criação de pequenos projetos. Rapidamente montei um ambiente Docker na minha máquina com spark e postgre.

Encontrei um grande desafio para provisionar o ambiente em Docker. Como sempre trabalhei com servidores appliances ou aplicações que tem que fazer a instalação direto no Linux (Redhat/Suse), nunca me preocupei em provisionar uma solução Docker.

Tive que relembrar muitos conceitos de Docker e ler muitas documentações para provisionar uma máquina Docker. Depois de provisionar, passei por muitos situações, como perder todas as configurações quando reinicia a máquina, alocar um volume permanente e falta de espaço no ambiente. Até deixar o ambiente 100%, levei 1 dia para configurar e deixar o ambiente apto para o desenvolvimento.