



Bootcamp: Engenharia de Dados

Enunciado do Desafio Final

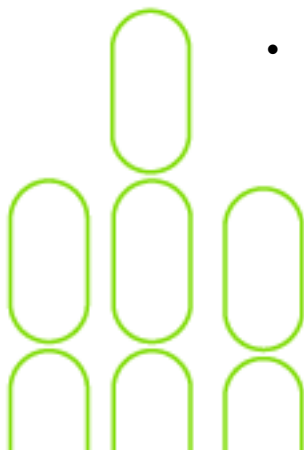
Módulo: Desafio Final

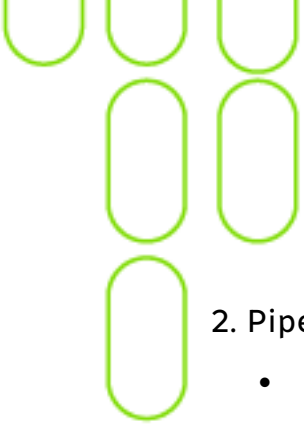
Objetivos de Ensino

1. Construção de Pipelines ETL com integração do Kafka com uma database (postgresql) usando kafka connect e entrega em data lake com kafka connect. Todos os serviços que compõem o kafka e o database PostgreSQL que servirá de fonte serão implantados com docker-compose.
2. Desenvolver uma solução prática de Engenharia de Dados que implemente a criação de pipelines ETL utilizando o modelo bronze, silver e gold, processados com Apache Spark SQL API e integrados a um datalake no Amazon S3 via Kafka Connect.

Requisitos

1. Pipeline Bronze (Ingestão Bruta)

- Fonte de Dados: Arquivo JSON fornecido (com sujeiras e duplicações).
 - Ferramenta: Spark SQL para carregar os dados e criar uma tabela temporária ou persistente (formato Parquet ou Delta).
 - Processamento:
 - Carregar dados brutos para a camada Bronze, sem transformação além da validação do esquema em um banco de dados (por exemplo, PostgreSQL).
- 



2. Pipeline Silver (Limpeza e Transformação)

- Fonte de Dados: Tabela Bronze.
- Ferramenta: Spark SQL para limpeza e transformações.
- Processamento:
 - Remover duplicações.
 - Tratar dados ausentes (ex.: preencher valores nulos ou descartar registros inválidos).
 - Ajustar colunas para um formato consistente (ex.: normalizar nomes).
 - Salvar os dados limpos em uma tabela Silver em um banco de dados (por exemplo, PostgreSQL).


3. Pipeline Gold (Agregação e Enriquecimento)


- Fonte de Dados: Tabela Silver.
- Ferramenta: Spark SQL para realizar agregações e cálculos.
- Processamento:
 - Gerar métricas agregadas (ex.: número de usuários ativos, média de idade).
 - Criar a camada Gold contendo dados prontos para consumo analítico em um banco de dados (por exemplo, PostgreSQL).

4. Integração com Kafka Connect e Datalake no S3

- Configurar um tópico no Apache Kafka para escutar as alterações da tabela Gold.
- Utilizar Kafka Connect para transferir os dados do tópico para um diretório no Amazon S3.

5. Orquestração no Airflow - **Opcional**

- Configurar e orquestrar os pipelines no Apache Airflow:
- 

- 
- Pipeline Bronze: Leitura e armazenamento inicial dos dados brutos.
 - Pipeline Silver: Transformações e limpeza de dados.
 - Pipeline Gold: Agregação e preparação para consumo.
 - Garantir dependências (Bronze → Silver → Gold).

Entregáveis

1. Screenshots que comprovem as tabelas carregadas no Postgres.
2. Screenshots ou logs que comprovem os Código Spark para os pipelines (incluindo consultas Spark SQL).
3. Screenshots ou logs que comprovem as Configuração do Kafka e Kafka Connect.
4. Screenshots ou logs que comprovem a execução dos pipelines.
5. Screenshots ou logs que comprovem os Dados processados no Amazon S3, organizados e particionados.

Passo a passo


1. Pré-requisitos

- Docker
- docker-compose
- Uma conta AWS free tier

2. Configurar o arquivo .env_kafka_connect

Você deve criar um arquivo .env_kafka_connect para cadastrar as chaves de sua conta aws como variáveis de ambiente que serão injetadas dentro do container do kafka connect. O arquivo deve ser conforme o modelo:

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxxx
```



AWS_SECRET_ACCESS_KEY=xx

3. Buildar a imagem do kafka-connect

Após clonar o repositório, mude para a pasta custom-kafka-connectors-image, execute o seguinte comando:

```
cd connect/custom-kafka-connectors-image
```

```
docker buildx build . -t connect-custom:1.0.0
```

```
PS D:\DockerCompose\DF EDD\custom-kafka-connector-image> docker buildx build . -t connect-custom:1.0.0
[+] Building 18.6s (3/5)                                docker:default
=> [internal] load .dockerignore                        0.0s
=> => transferring context: 2B                          0.0s
=> [internal] load build definition from Dockerfile    0.0s
=> => transferring dockerfile: 243B                     0.0s
=> [internal] load metadata for docker.io/confluentinc/cp-kafka-connect-base:7.0.0             3.9s
=> [1/2] FROM docker.io/confluentinc/cp-kafka-connect-base:7.0.0@sha256:5d4f484e219ad704a735058531e33a5c0cc6bec 14.6s
=> => resolve docker.io/confluentinc/cp-kafka-connect-base:7.0.0@sha256:5d4f484e219ad704a735058531e33a5c0cc6bec8 0.0s
=> => sha256:e1d5cfff8304763db32c90062e7452c41a4669f3b6a888ebe97419548cd28a160 26.09kB / 26.09kB 0.0s
=> => sha256:4fa716e4af8c49fe1e18eed8a4e78831a95bdf83c18b426495bb7039d31afa1e 3.05kB / 3.05kB 0.6s
=> => sha256:5d4f484e219ad704a735058531e33a5c0cc6bec89e97c7a378f3bab2f48e95bf 3.05kB / 3.05kB 0.0s
=> => sha256:28dc26abaa754ff96142aa7334babadb5f816b60cbdc0d21499c0762f32c9f0a 541.21MB / 541.21MB 11.1s
=> => extracting sha256:28dc26abaa754ff96142aa7334babadb5f816b60cbdc0d21499c0762f32c9f0a 3.3s
=> => extracting sha256:4fa716e4af8c49fe1e18eed8a4e78831a95bdf83c18b426495bb7039d31afa1e 0.0s
```

Uma nova imagem, com o nome connect-custom e tag 1.0.0, será criada. Essa é a imagem que nosso serviço connect, dentro do docker-compose.yml, irá utilizar, com os conectores que precisaremos instalados.

4. Subir o PostgreSQL

No arquivo docker-compose.yml, na pasta postgres, estamos subindo o banco de dados.

```
docker-compose up -d
```

```
PS D:\DockerCompose\DF EDD\postgres> docker-compose up -d
[+] Running 15/15
✔ postgres 14 layers [#####] 0B/0B Pulled 8.6s
✔ 69692152171a Pull complete 1.2s
✔ a31b993d5cc6 Pull complete 0.9s
✔ f65921886500 Pull complete 0.6s
✔ b9c1a94e4ca8 Pull complete 1.3s
✔ 435dd99ceb68 Pull complete 1.6s
✔ d3ee8e88c67c Pull complete 1.8s
✔ 84b08674f942 Pull complete 1.9s
✔ 7d358e850d3e Pull complete 2.2s
✔ adf2c63307b4 Pull complete 3.8s
✔ 27ff0e95dd24 Pull complete 2.7s
✔ 550e7b1ab95a Pull complete 2.9s
✔ 2287baf15bf8 Pull complete 3.4s
✔ 97d11a196325 Pull complete 3.7s
✔ 0f11fc82fe79 Pull complete 4.0s
[+] Building 0.0s (0/0)                                docker:default
time="2024-11-30T13:08:46-03:00" level=warning msg="a network with name custom_network exists but was not created for project \"postgres\". \nSet 'external: true' to use an existing network"
[+] Running 0/0
- Container postgres Creating 0.0s
Error response from daemon: Conflict. The container name "/postgres" is already in use by container "8e0c0814a431b01368048d00ba975d2d947e5549a30a7f21aa81a7ef1c8e4f8f". You have to remove (or rename) that container to be able to reuse that name.
```

5. Processar o ETL

Veja o arquivo importar.ipynb (necessário o Jupyter)

- Instalando Todas as Dependências em um Único Comando

Para instalar e verificar possíveis dependências quebradas, use:

```
pip install --upgrade numpy pandas matplotlib seaborn scikit-learn scipy  
statsmodels jupyter notebook jupyterlab tensorflow keras pytorch-lightning  
xgboost lightgbm nltk spacy gensim plotly
```

6. Subir a plataforma Confluent no docker-compose

No arquivo docker-compose.yml, estamos subindo toda a estrutura da plataforma Confluent. Para isso, vamos entrar na pasta e subir a estrutura.

connectors	30/11/2024 13:02	Pasta de arquivos	
custom-kafka-connector-image	30/11/2024 13:02	Pasta de arquivos	
MSK	30/11/2024 13:02	Pasta de arquivos	
postgres	30/11/2024 13:02	Pasta de arquivos	
.env_kafka_connect	30/11/2024 13:02	Arquivo ENV_KAFKA_C...	1 KB
<u>docker-compose.yml</u>	28/11/2024 22:23	Arquivo Fonte Yaml	5 KB
importar.ipynb	30/11/2024 13:17	JetBrains DataSpell	7 KB
README.md	28/11/2024 22:23	Arquivo Fonte Markdown	6 KB

```
docker-compose up -d
```

7. Criar dois tópicos no Kafka

- Iniciar o contêiner do Kafka Broker

Antes de criar os tópicos, você precisa acessar o contêiner onde o Kafka está rodando.

1. Certifique-se de que o Docker está ativo e os serviços do Kafka estão em execução, normalmente usando um arquivo docker-compose.yml ou diretamente com o comando docker run.

2. Para acessar o contêiner do Kafka Broker, execute o seguinte comando:

```
docker exec -it broker bash
```

Após executar esse comando, você estará no terminal do contêiner Kafka Broker.

```
D:\DockerCompose\DF EDD>docker exec -it broker bash
[appuser@broker ~]$
```

Explicação:

- `docker exec`: Executa um comando em um contêiner ativo.
- `-it`: Abre uma sessão interativa com o contêiner.
- `broker`: Nome do contêiner que roda o Kafka Broker (esse nome pode variar de acordo com a configuração do seu `docker-compose.yml`).

- Criar os tópicos no Kafka

Agora que você está dentro do contêiner do Kafka, use o comando `kafka-topics` para criar os tópicos. Cada tópico armazenará os dados movidos do PostgreSQL.

1. Comando para criar o tópico `postgres-dadostesouroipca`:

```
kafka-topics --create \
--bootstrap-server localhost:9092 \
--partitions 1 \
--replication-factor 1 \
--topic postgres-dadostesouroipca
```

```
D:\DockerCompose\DF EDD>docker exec -it broker bash
[appuser@broker ~]$ kafka-topics --create \
> --bootstrap-server localhost:9092 \
> --partitions 1 \
> --replication-factor 1 \
> --topic postgres-dadostesouroipca
Created topic postgres-dadostesouroipca.
```

- `--bootstrap-server localhost:9092`: Especifica o endereço do servidor Kafka. O `localhost:9092` é o endereço padrão usado em contêineres Kafka.
- `--partitions 1`: Define o número de partições do tópico. Para este exemplo, usamos 1 partição.

- --replication-factor 1: Define o número de réplicas para o tópico. Usamos 1, pois estamos rodando o Kafka em um único broker.
- --topic postgres-dadostesouroipca: Nome do tópico sendo criado.

2. Comando para criar o tópico postgres-dadostesouopre:

Repita o comando acima, alterando o nome do tópico:

```
kafka-topics --create \  
--bootstrap-server localhost:9092 \  
--partitions 1 \  
--replication-factor 1 \  
--topic postgres-dadostesouopre
```

```
[appuser@broker ~]$ kafka-topics --create \  
> --bootstrap-server localhost:9092 \  
> --partitions 1 \  
> --replication-factor 1 \  
> --topic postgres-dadostesouopre  
Created topic postgres-dadostesouopre.
```

- Verificar se os tópicos foram criados

Após executar os comandos acima, é importante confirmar que os tópicos foram criados com sucesso. Use o comando abaixo para listar os tópicos disponíveis no Kafka:

```
kafka-topics --bootstrap-server localhost:9092 --list
```

Isso exibirá todos os tópicos criados no Kafka. Você deve ver os nomes postgres-dadostesouroipca e postgres-dadostesouopre na lista.

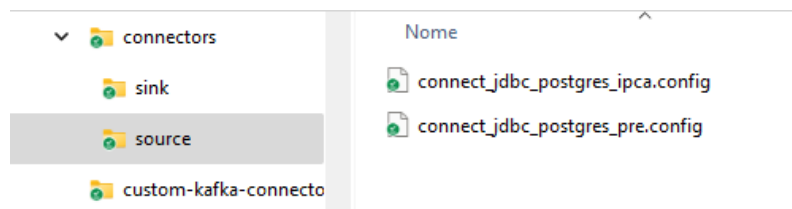
8. Registrar os conectores Kafka Source

Os conectores Kafka Source serão configurados para extrair dados do PostgreSQL e enviá-los para os tópicos no Kafka. Para isso, vamos precisar de um arquivo no

formato json contendo as configurações do conector que vamos registrar. O arquivo `connect_jdbc_postgres_ipca.config` possui a implementação do IPCA. O arquivo `connect_jdbc_postgres_pre.config` possui a implementação do PRE.

Os conectores são configurados através de arquivos JSON contendo os parâmetros necessários. Aqui está como configurar:

Crie os arquivos `connect_jdbc_postgres_ipca.config` e `connect_jdbc_postgres_pre.config` com o seguinte conteúdo. Salve cada arquivo no diretório onde você irá executar os comandos de registro.



Arquivo: `connect_jdbc_postgres_ipca.config`

```
{
  "name": "postg-connector-ipca",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": 1,
    "connection.url": "jdbc:postgresql://postgres:5432/postgres",
    "connection.user": "postgres",
    "connection.password": "Jp1987",
    "mode": "timestamp",
    "timestamp.column.name": "dt_update",
    "table.whitelist": "public.[nomedobd_ipca]",
    "topic.prefix": "postgres-",
    "validate.non.null": "false",
    "poll.interval.ms": 500
  }
}
```

Arquivo: `connect_jdbc_postgres_pre.config`

```
{
```

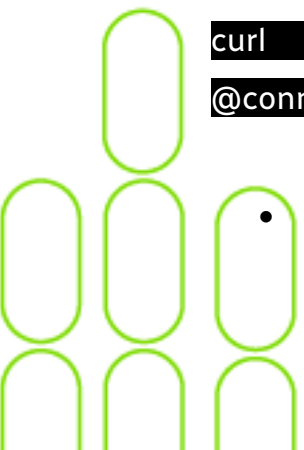




```
"name": "postg-connector-pre",
"config": {
  "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
  "tasks.max": 1,
  "connection.url": "jdbc:postgresql://postgres:5432/postgres",
  "connection.user": "postgres",
  "connection.password": "Jp1987",
  "mode": "timestamp",
  "timestamp.column.name": "dt_update",
  "table.whitelist": "public.[nomedobd_pre]",
  "topic.prefix": "postgres-",
  "validate.non.null": "false",
  "poll.interval.ms": 500
}
}
```

- Execute os comandos curl para registrar os conectores:
 - No terminal do host (não dentro do contêiner), execute os comandos para registrar os conectores. Certifique-se de estar no diretório onde os arquivos estão salvos ou forneça o caminho completo (por exemplo, /path/to/connectors/source/).

```
curl -X POST -H "Content-Type: application/json" --data
@connect_jdbc_postgres_ipca.config http://localhost:8083/connectors
```

```
curl -X POST -H "Content-Type: application/json" --data
@connect_jdbc_postgres_pre.config localhost:8083/connectors
```

- 
- Verificar os conectores e tópicos



Verifique o consumo de dados nos tópicos. O comando `kafka-console-consumer` é usado para consumir mensagens de um tópico Kafka. Vamos verificar os dados nos tópicos `postgres-dadostesouroipca` e `postgres-dadostesouropre`.

- Explicação do Comando

`kafka-console-consumer`:

- Ferramenta CLI do Kafka para consumir mensagens de um tópico.

`--bootstrap-server localhost:9092`:

- Especifica o servidor Kafka que será usado para consumir mensagens.
- No exemplo, usamos `localhost:9092`, que é a porta padrão do Kafka Broker.

`--topic <nome_do_tópico>`:

- Define o tópico Kafka de onde você quer consumir as mensagens.
- No seu caso, os tópicos são `postgres-dadostesouroipca` e `postgres-dadostesouropre`.

`--from-beginning`:

- Indica que o consumo de mensagens deve começar desde o início do tópico (todas as mensagens enviadas desde a criação do tópico).

Dentro do contêiner Docker, acesse o contêiner onde o Kafka Broker está rodando:

```
docker exec -it broker bash
```

Verifique o consumo de dados nos tópicos.

IPCA

```
kafka-console-consumer --bootstrap-server localhost:9092 \  
--topic postgres-dadostesouroipca \  
--from-beginning
```

PRE



```
kafka-console-consumer --bootstrap-server localhost:9092 \
--topic postgres-dadostesouopre \
--from-beginning
```

Verifique os detalhes dos tópicos:

IPCA

```
kafka-topics --bootstrap-server localhost:9092 \
--describe \
--topic postgres-dadostesouroipca
```

PRE

```
kafka-topics --bootstrap-server localhost:9092 \
--describe \
--topic postgres-dadostesouopre
```

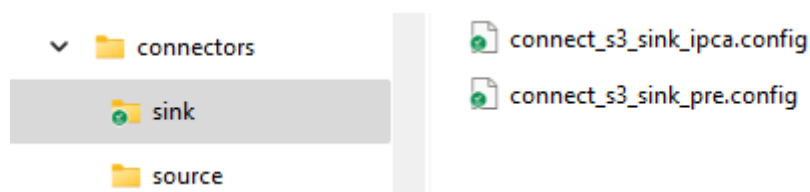
Fora do container, exiba os logs do Kafka Connect para garantir que os conectores não apresentam erros:

```
docker logs -f connect
```

9. Configurar os Sink Connectors

Os Sink Connectors enviam os dados dos tópicos para o S3.

Crie os arquivos de configuração para os Sink Connectors.



- Arquivo: connect_s3_sink_ipca.config

Este arquivo configura o conector para enviar dados do tópico postgres-dadostesouroipca para um bucket S3.

```
{
  "name": "s3-sink-ipca",
```

```

    "config": {
      "connector.class": "io.confluent.connect.s3.S3SinkConnector",
      "format.class": "io.confluent.connect.s3.format.json.JsonFormat",
      "keys.format.class": "io.confluent.connect.s3.format.json.JsonFormat",
      "schema.generator.class":
"io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator",
      "flush.size": 2,
      "schema.compatibility": "FULL",
      "s3.bucket.name": "NOME-DO-BUCKET",
      "s3.region": "us-east-1",
      "s3.object.tagging": true,
      "s3.ssea.name": "AES256",
      "topics.dir": "raw-data/kafka",
      "storage.class": "io.confluent.connect.s3.storage.S3Storage",
      "tasks.max": 1,
      "topics": "postgres-dadostesouroipca"
    }
  }
}

```

- Arquivo: connect_s3_sink_pre.config

Este arquivo configura o conector para enviar dados do tópico postgres-dadostesouropre para o mesmo bucket S3.

```

{
  "name": "s3-sink-pre",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "format.class": "io.confluent.connect.s3.format.json.JsonFormat",
    "keys.format.class": "io.confluent.connect.s3.format.json.JsonFormat",
    "schema.generator.class":
"io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator",
    "flush.size": 2,
    "schema.compatibility": "FULL",

```

```

"s3.bucket.name": "NOME-DO-BUCKET",
"s3.region": "us-east-1",
"s3.object.tagging": true,
"s3.ssea.name": "AES256",
"topics.dir": "raw-data/kafka",
"storage.class": "io.confluent.connect.s3.storage.S3Storage",
"tasks.max": 1,
"topics": "postgres-dadostesouropre"
}
}

```

- Registre os Sink Connectors no Kafka Connect:

IPCA

```

curl -X POST -H "Content-Type: application/json" --data
@connect_s3_sink_ipca.config http://localhost:8083/connectors

```

- Resultado

```

D:\DockerCompose\DF EDD\connectors\sink>curl -X POST -H "Content-type: application/json" --data @connect_s3_sink_ipca.config localhost:8083/conne
{"name":"customers-s3-sink-ipca","config":{"connector.class":"io.confluent.connect.s3.S3SinkConnector","format.class":"io.confluent.connect.s3.fo
rmat.json.JsonFormat","schema.generator.class":"io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator","flush.size":"2","schema.comp
us-east-1","s3.object.tagging":"true","s3.ssea.name":"AES256","topics.dir":"raw-data/ipca/kafka","storage.class":"io.confluent.connect.s3.storage
name":"customers-s3-sink-ipca"},"tasks":[],"type":"sink"}

```

PRE

```

curl -X POST -H "Content-Type: application/json" --data
@connect_s3_sink_pre.config http://localhost:8083/connectors

```

- Resultado

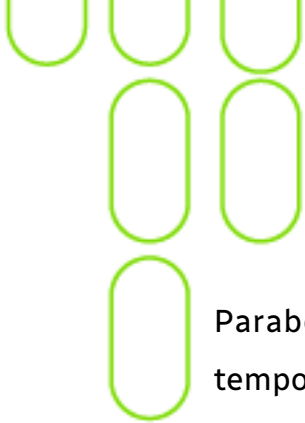
```

D:\DockerCompose\DF EDD\connectors\sink>curl -X POST -H "Content-type: application/json" --data @connect_s3_sink_pre.config http://localhost:8083/connectors
{"name":"customers-s3-sink-pre","config":{"connector.class":"io.confluent.connect.s3.S3SinkConnector","format.class":"io.confluent.connect.s3.format.json.JsonFormat",
format.json.JsonFormat","schema.generator.class":"io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator","flush.size":"2","schema.compatibility":"FULL","s3
s-east-1","s3.object.tagging":"true","s3.ssea.name":"AES256","topics.dir":"raw-data/pre/kafka","storage.class":"io.confluent.connect.s3.storage.S3Storage","tasks.max"
e":"customers-s3-sink-pre"},"tasks":[],"type":"sink"}

```

10. Verificar a entrega ao S3

Certifique-se de que os dados dos tópicos postgres-dadostesouroipca e postgres-dadostesouropre estão sendo escritos no bucket S3 configurado.



Parabéns!! Você acabou de concluir o seu pipeline de processamento de dados em tempo real usando a plataforma Confluent no docker-compose!

