

Bootcamp: Engenheiro(a) de Dados

Desafio Final

Objetivos de Ensino

O tema do desafio final é a 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.

Portanto, vamos 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.

Enunciado

Tema:

Requisitos:

1. Pipeline Bronze (Ingestão Bruta)

- **Fonte de Dados:** Vamos consumir os dados brutos de uma URL contendo um arquivo CSV com preços e taxas dos títulos públicos (Tesouro Direto), disponibilizado no portal de dados abertos do Tesouro Nacional (CKAN é o sistema de dados abertos usado).
- **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).

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 os Dados processados no Amazon S3, organizados e particionados.

PARTE 01 – CAMADAS BRONZE

Passo a passo para execução

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=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Configurar o Bucket no Amazon S3

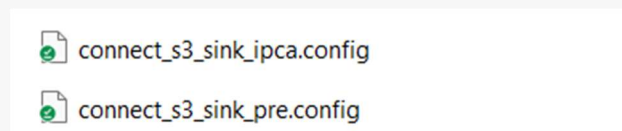
O Kafka Connect precisa das credenciais para se autenticar no Amazon S3. Essas credenciais foram ser fornecidas dentro de arquivo de configuração.

Agora vá ao console da AWS.

Crie buckets chamados my-bucket-xx-01 e my-bucket-xx-02, ou da forma que preferir.

Escolha a região compatível (ex.: us-east-1).

Você precisa acertar os arquivos de configuração que estão em ...connectors\sink com os nomes criados para os buckets 01 e 02. Procure as linhas de bucket.name nos dois arquivos abaixo.

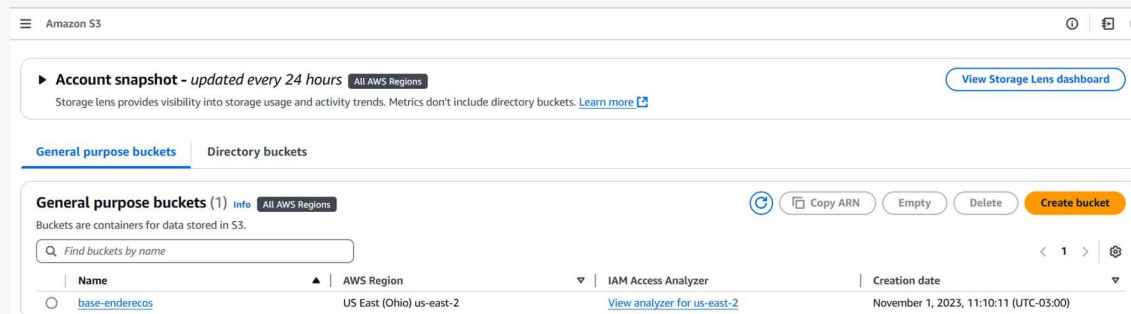


Por exemplo:

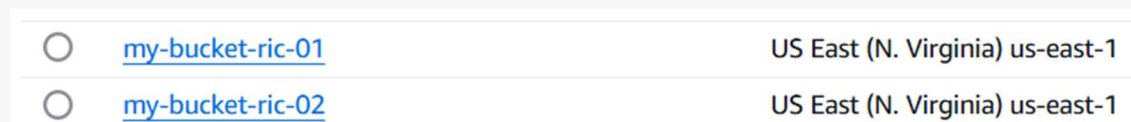
```
"s3.bucket.name": "my-bucket-jp-02",
```

```
"s3.region": "us-east-1",
```

Criando um bucket:



Por exemplo, meus buckets se chamam:



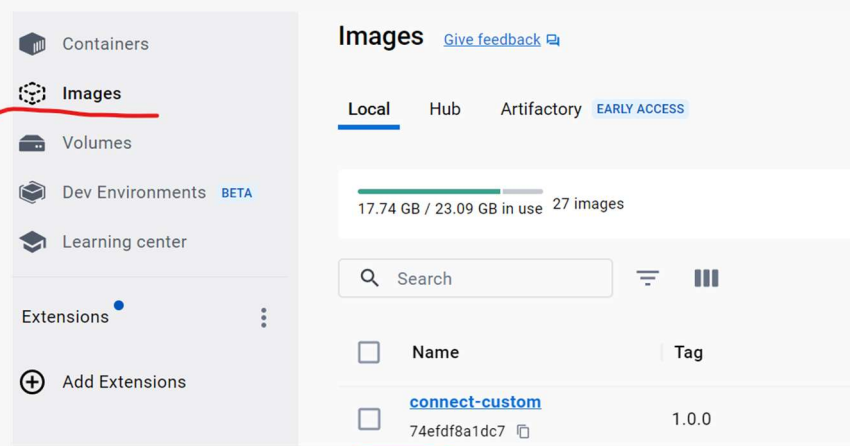
3 - Buildar a imagem do kafka-connect

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

```
../custom-kafka-connector-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:e1d5c5ff8304763db32c90062e7452c41a4669f3b6a888ebe97419548cd28a160 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.



Explicação: O comando docker buildx build cria uma nova imagem Docker a partir do conteúdo da pasta custom-kafka-connectors-image. Essa pasta contém os arquivos necessários para personalizar a imagem, como configurações específicas e conectores adicionais.

Essa imagem será usada pelo serviço Kafka Connect definido no arquivo docker-compose.yml. Os conectores personalizados incluídos na imagem são necessários para realizar a integração com as fontes de dados (PostgreSQL) e destinos (Amazon S3).

4 - Subir o PostgreSQL

Entre na pasta postgres e rode o arquivo docker-compose.yml, para subir 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 pr
ject \"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 "8e0c0814a431b013680
48d00ba975d2d947e5549a30a7f21aa81a7ef1c8e4f8f". You have to remove (or rename) that container to be able to reuse that n
ame.
```

5 - Processar o ETL

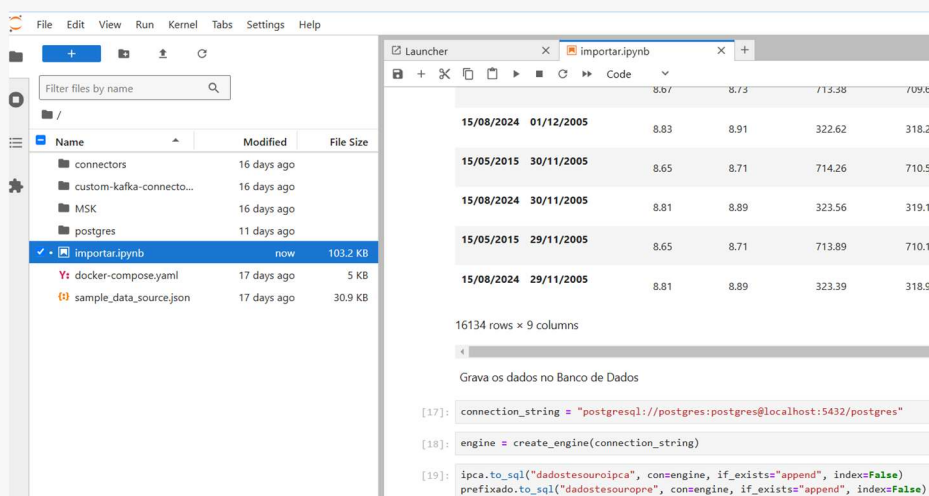
Abra e rode o arquivo importar.ipynb. Você pode rodar este notebook no Jupyter. Pelo fato do seu Postgres estar em um container local, o código vai dar erro na conexão pelo uso do Google Colab. Se você não tiver o Jupyter notebook instalado, basta executar o pip install.

D:\DockerCompose\DF EDD>pip install notebook

E para rodar, estando dentro do diretório o projeto basta executar o comando:

D:\DockerCompose\DF EDD>jupyter notebook

O Jupyter irá abrir o seu browser com a interface abaixo:

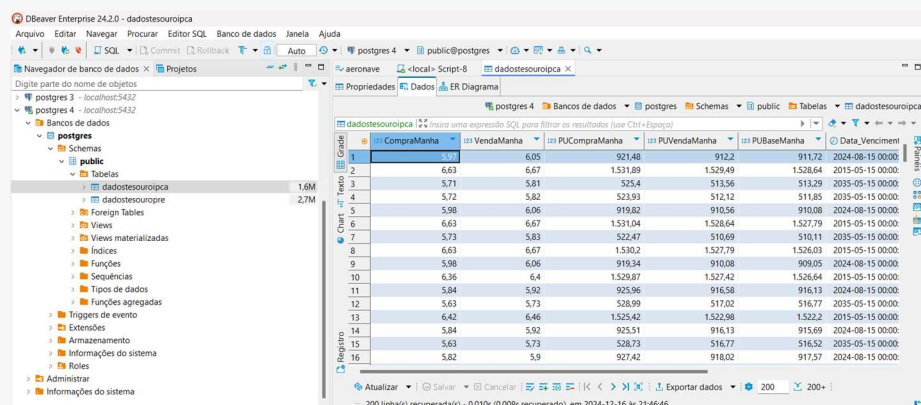


Rode o notebook **importar.ipynb**.

Depois de rodar, confira as tabelas no Postgres. Por exemplo, pode usar o DBeaver como ferramenta gerenciadora do banco de dados.

Os parâmetros de conexão estão dentro do arquivo **docker-compose.yaml** na pasta postgres.

```
postgres:
  image: postgres:13.2
  ports:
    - 5432:5432
  hostname: postgres
  container_name: postgres
  environment:
    POSTGRES_PASSWORD: postgres
```



6 - Subir a plataforma Confluent pelo docker-compose

No arquivo **docker-compose.yml** localizado na raiz do projeto 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	
postgres	05/12/2024 19:08	Pasta de arquivos	
spark	17/12/2024 11:12	Pasta de arquivos	
.env_kafka_connect	30/11/2024 13:02	Arquivo ENV_KAFKA_CO...	1 KB
aws-java-sdk-bundle-1.12.262.jar	17/12/2024 07:58	Executable Jar File	274.068 KB
docker-compose.yaml	28/11/2024 22:23	Arquivo Fonte Yaml	5 KB
etl-spark.ipynb	17/12/2024 11:32	JetBrains DataSpell	18 KB
hadoop-aws-3.3.4.jar	17/12/2024 07:56	Executable Jar File	941 KB
importar.ipynb	16/12/2024 22:33	JetBrains DataSpell	102 KB







docker-compose up -d

Este Docker Compose cria uma arquitetura de Kafka com suporte completo para:

- Streaming de Dados com Kafka Broker.
- Coordenação através do Zookeeper.
- Gerenciamento de Esquemas com Schema Registry.
- Integração com sistemas externos via Kafka Connect.
- Consultas SQL em tempo real com ksqldb-server e ksqldb-cli.

- Interface REST para Kafka via REST Proxy.

Se o container do proxy não subir na porta 8082, você pode fazer o procedimento abaixo:

<input type="checkbox"/>		schema-registry 2c113791f4 confluentinc/cp-sch Running	0.49%	8081:8081 
<input type="checkbox"/>		rest-proxy 6595f962be confluentinc/cp-kaf Created	0%	<u>8082:8082</u> 
<input type="checkbox"/>		connect 6bc6d09b9c connect-custom:1.0 Running	0.59%	8083:8083 

Identifique o processo que usa a porta 8082. No terminal, verifique qual processo está ocupando a porta:

No Windows:

```
netstat -ano | findstr :8082
```

Você verá o PID (Process ID) do processo que está usando a porta.

Para encerrar o processo, use:

Obs: para matar a tarefa, entre no terminal com acesso de administrador.

```
taskkill /PID <PID> /F
```

```
D:\DockerCompose\DF EDD>netstat -ano | findstr :8082
TCP    0.0.0.0:8082        0.0.0.0:0          LISTENING        8492
TCP    [::]:8082          [::]:0             LISTENING        8492

D:\DockerCompose\DF EDD>taskkill /PID 8492 /F
ÊXITO: o processo com PID 8492 foi finalizado.
```

Tente iniciar o container do proxy novamente.

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.
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-dadostesouropre`:

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-dadostesouoipca e postgres-dadostesouopre na lista.

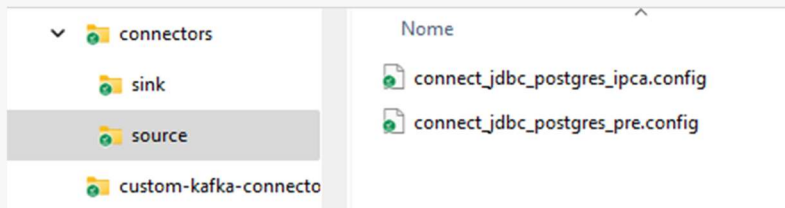
```
[appuser@broker ~]$ kafka-topics --bootstrap-server localhost:9092 --list  
_consumer_offsets  
_transaction_state  
_confluent-ksql-default__command_topic  
_schemas  
default_ksql_processing_log  
docker-connect-configs  
docker-connect-offsets  
docker-connect-status  
postgres-dadostesouoipca  
postgres-dadostesouopre
```

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": "postgres",
    "mode": "timestamp",
    "timestamp.column.name": "dt_update",
    "table.whitelist": "public.dadostesouroipca",
    "topic.prefix": "postgres-",
    "validate.non.null": "false",
    "poll.interval.ms": 500
  }
}
```

Arquivo: `connect_jdbc_postgres_pre.config`

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

Execute os comandos curl para registrar os conectores:

Esta etapa envolve o registro de conectores no Kafka Connect utilizando o comando curl no terminal. O Kafka Connect é uma ferramenta usada para integrar sistemas externos com o Apache Kafka, e, neste caso, estamos registrando conectores JDBC para conectar bancos de dados PostgreSQL ao Kafka.

O objetivo é registrar dois conectores JDBC no Kafka Connect para que ele possa ler dados de duas tabelas do PostgreSQL (provavelmente relacionadas ao IPCA e Prefixados, conforme os arquivos de configuração).

- 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, `.../connectors/source/`), antes de executar o comando.

```
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
```

Resultado dos comandos executados:

```
D:\DockerCompose\DF EDD\connectors\source>curl -X POST -H "Content-Type: application/json" --data @connect_jdbc_postgres_ipca.config http://localhost:8083/connectors
{"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":"postgres","mode":"timestamp","timestamp.column.name":"dt_update","table.whitelist":"public.dadostesouroipca","topic.prefix":"postgres-","validate.non.null":"false","poll.interval.ms":"500","name":"postg-connector-ipca"},"tasks":[],"type":"source"}

D:\DockerCompose\DF EDD\connectors\source>curl -X POST -H "Content-Type: application/json" --data @connect_jdbc_postgres_pre.config localhost:8083/connectors
{"name":"postg-connector","config":{"connector.class":"io.confluent.connect.jdbc.JdbcSourceConnector","tasks.max":"1","connection.url":"jdbc:postgresql://postgres:5432/postgres","connection.user":"postgres","connection.password":"postgres","mode":"timestamp","timestamp.column.name":"dt_update","table.whitelist":"public.dadostesouropre","topic.prefix":"postgres-","validate.non.null":"false","poll.interval.ms":"500","name":"postg-connector"},"tasks":[],"type":"source"}
```

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

Vamos entrar no Docker e acessar 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-dadostesouropre
```

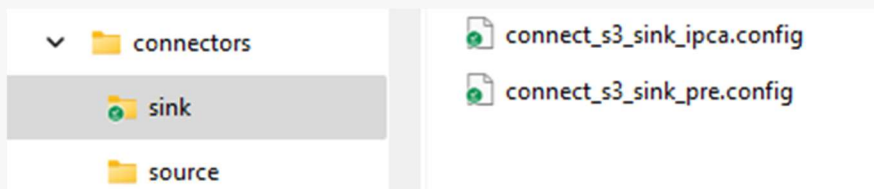
Fora do container, exiba os logs do Kafka Connect para garantir que os conectores não apresentam erros:

```
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-dadostesouopre 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-dadostesouopre"
  }
}
```

Registre os Sink Connectors no Kafka Connect.

Fora do container, vá no diretório ...connectors/sink e execute os comando abaixo.

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

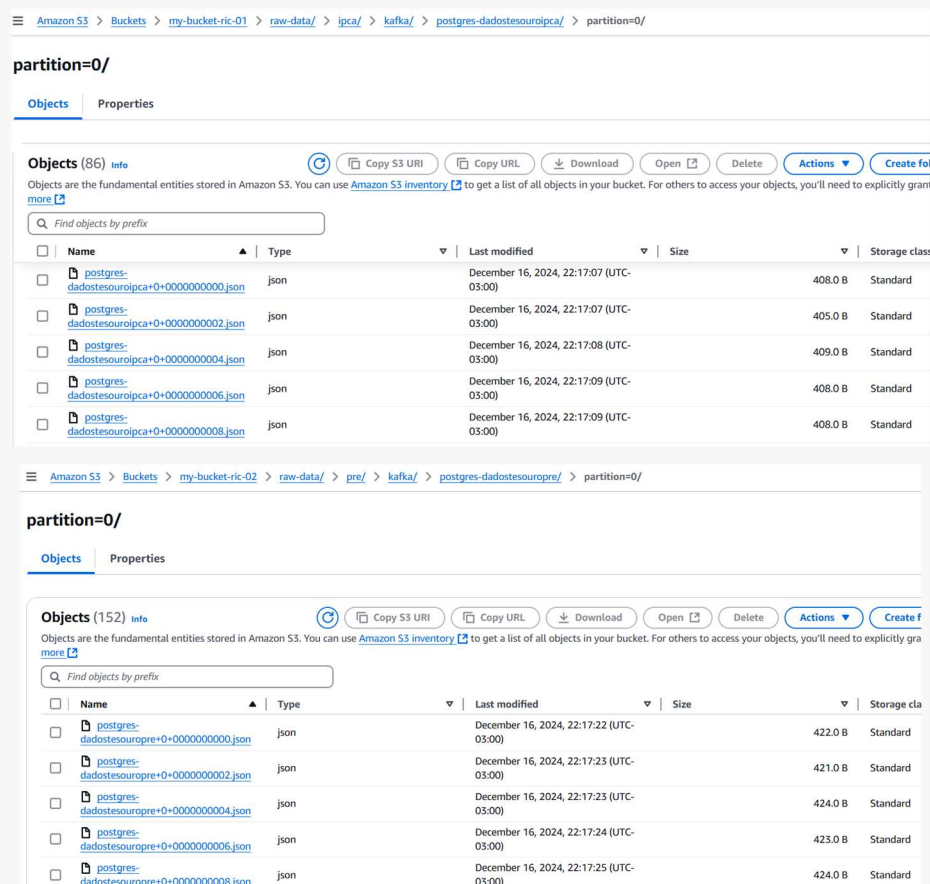
```
curl -X POST -H "Content-Type: application/json" --data @connect_s3_sink_pre.config
http://localhost:8083/connectors
```

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.fo
rmat.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-dadostesouopre estão sendo escritos no bucket S3 configurado.



The first screenshot shows the 'Objects (86)' list for the bucket 'my-bucket-ric-01' under the path 'raw-data/ > ipca/ > kafka/ > postgres-dadostesouroipca/'. The objects are JSON files with names like 'postgres-dadostesouroipca+0+0000000000.json' and sizes around 408.0 B.

The second screenshot shows the 'Objects (152)' list for the bucket 'my-bucket-ric-02' under the path 'raw-data/ > pre/ > kafka/ > postgres-dadostesouopre/'. The objects are JSON files with names like 'postgres-dadostesouopre+0+0000000000.json' and sizes around 422.0 B.

Você verá os arquivos no S3 no formato json, como por exemplo:

```
{
  "CompraManha": 12.73,
  "VendaManha": 12.79,
  "PUCompraManha": 631.4,
  "PUVendaManha": 630.11,
  "PUBaseManha": 629.81,
  "Data_Vencimento": 1420070400000,
  "Data_Base": 1298851200000,
  "Tipo": "PRE-FIXADOS",
  "dt_update": 1734381830665
}
```


PARTE 02 – CAMADAS SILVER E GOLD

Apache Spark para as camadas Silver e Gold

1. Configuração das Permissões nos Buckets

Vamos precisar dar acesso aos buckets.

Vá em cada bucket, nas permissões e crie a seguinte diretiva. Repare que você precisa buscar antes no IAM seu código e usuário.

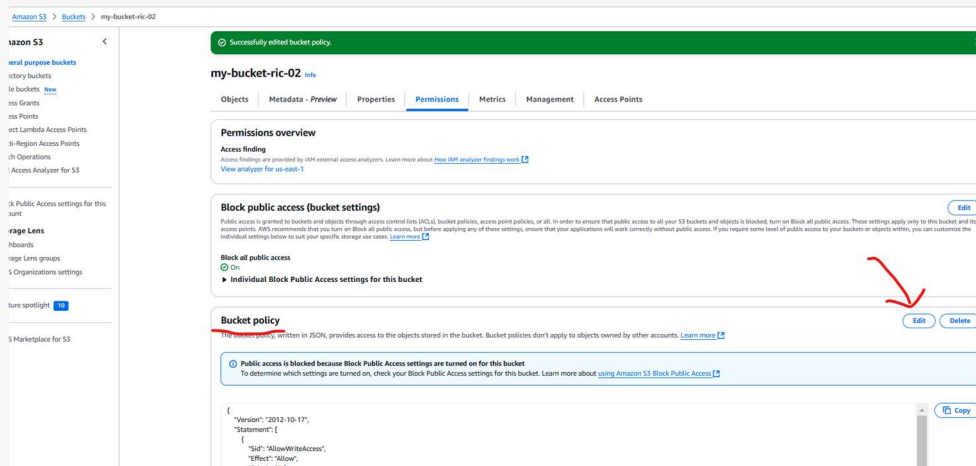
"AWS": "arn:aws:iam::123456789012:user/ricardobalves"

Permissões a serem inseridas em cada bucket. Não se esqueça de alterar seu user e o nome do bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowWriteAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/ricardobalves"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket-ric-01",
        "arn:aws:s3:::my-bucket-ric-01/*"
      ]
    }
  ]
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowWriteAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/ricardobalves"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ]
    }
  ]
}
```

```
"s3:DeleteObject",
"s3:ListBucket"
],
"Resource": [
"arn:aws:s3:::my-bucket-ric-02",
"arn:aws:s3:::my-bucket-ric-02/*"
]
}
```



2. Instalação do Apache Spark

Vamos começar usando a pasta spark, que está na estrutura de diretórios.

Você pode ter o Spark de duas formas, que vou explicar abaixo: instalação local na sua máquina ou instalação utilizando contêineres do Docker.

a) Spark + Hadoop: instalação local no Windows.

Você pode buscar as pastas spark e hadoop no drive:

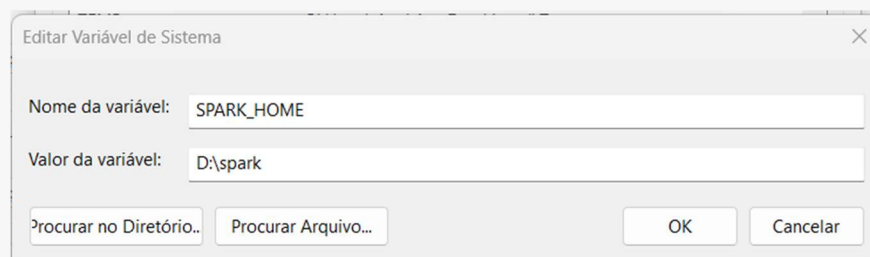
https://drive.google.com/drive/folders/1Sv_6K8QgS2N69ICafJNHIAPPBv8kX6YG?

Coloque em um lugar como por exemplo a raiz do drive d.

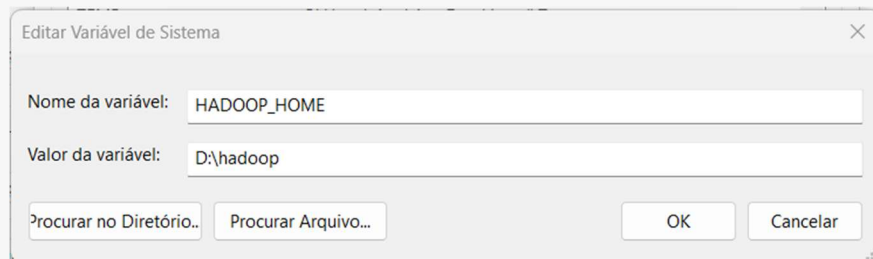
```
D:\spark
D:\hadoop
```

Edite as variáveis de ambiente e crie as seguintes variáveis.

SPARK_HOME

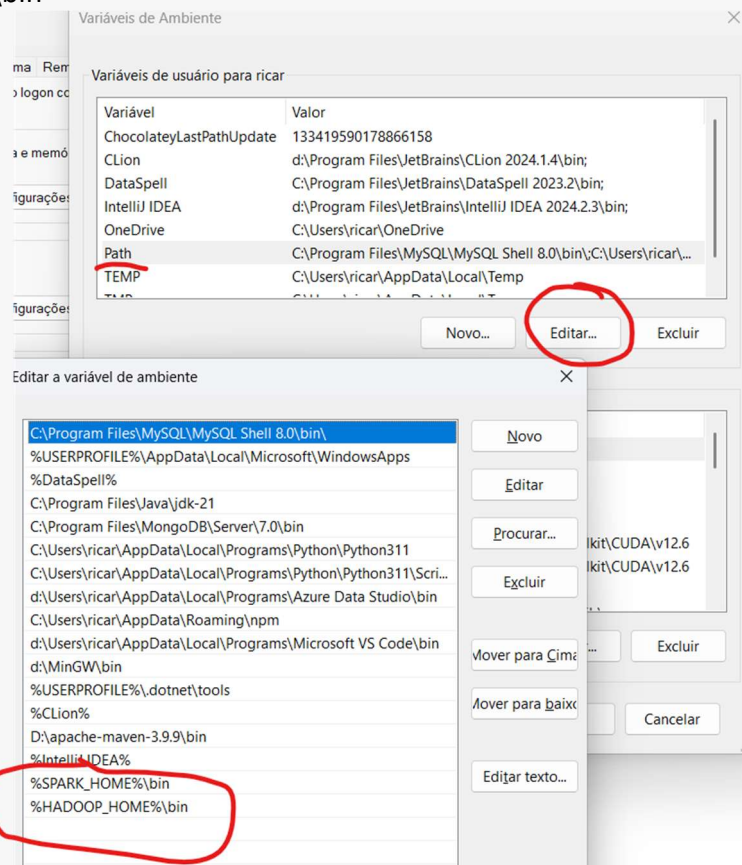


HADOOP_HOME



Adicione no path

%SPARK_HOME%\bin
%HADOOP_HOME%\bin



Você precisa ter o Jupiter notebook operando na sua máquina, para abrir o arquivo do notebook (etl-spark.ipynb).

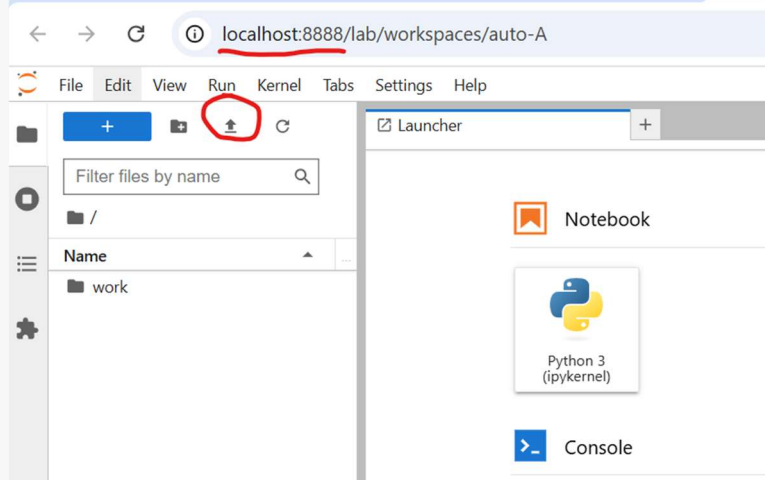
b) Uso do Spark via contêineres

Vá no diretório spark e rode o Docker compose. Ele vai subir o conjunto de contêineres necessários para rodarmos o spark, incluindo o Jupyter Notebook.

```
D:\DockerCompose\DF EDD\spark>docker-compose up -d
[+] Building 0.0s (0/0)
[+] Running 3/3
✓Container spark-master      Started
✓Container spark-worker      Started
✓Container jupyter-notebook  Started
```

Va no notebook criado para o Jupyter e clique no endereço <http://localhost:8888>. Vai abrir o Jupyter Notebook no browser, a partir do container.

Para abrir o arquivo do notebook (etl-spark.ipynb), basta fazer o upload do arquivo.



Para ambos os ambientes do Spark, precisaremos dos arquivos jar:

1. **hadoop-aws**: O conector Hadoop para AWS.
2. **aws-java-sdk-bundle**: O SDK da Amazon para AWS.

Versão Recomendada:

- hadoop-aws-3.3.4.jar
- aws-java-sdk-bundle-1.12.262.jar

 aws-java-sdk-bundle-1.12.262.jar	17/12/2024 07:58	Executable Jar File
 hadoop-aws-3.3.4.jar	17/12/2024 07:56	Executable Jar File

3. Entendendo o notebook etl-spark.ipynb para transformações

Você pode rodar o notebook, entretanto é interessante que leia as explicações abaixo antes de rodar o código.

Camada Bronze: Os dados foram ingeridos como estão, no formato bruto.

Camada Silver: Vamos converter timestamps em datas legíveis e trataremos duplicações e inconsistências.

Camada Gold: Vamos agregar dados, como médias de preços unitários (PU) e métricas por tipo de título ("PRE-FIXADOS").

Leitura dos dados brutos no S3

Carregue os dados brutos do S3 que estão em formato **JSON**:

```
from pyspark.sql import SparkSession
```

```
# Inicializar Spark Session
```

```
spark = SparkSession.builder \
```

```
.appName("ETL Pipeline - Bronze to Silver") \
```

```
.config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem") \
```

```
.getOrCreate()
```

```
# Caminho dos dados brutos no S3 (camada Bronze)
bronze_path = "s3a://my-bucket-ric-01/raw-data/ipca/kafka/"
```

```
# Ler os dados brutos do S3
df_bronze = spark.read.json(bronze_path)
```

```
# Visualizar os dados brutos
df_bronze.show()
```

Transformações para a Camada Silver (Limpeza e Transformação)

Aqui você realiza:

- Remoção de duplicações.
- Conversão de timestamps em formato de data.
- Tratamento de dados ausentes ou inválidos.
- Normalização de colunas.

Código para a Camada Silver

```
from pyspark.sql.functions import col, from_unixtime
```

```
# Remover duplicações
df_silver = df_bronze.dropDuplicates()
```

```
# Tratar timestamps e converter para formato de data legível
df_silver = df_silver.withColumn("Data_Vencimento", from_unixtime(col("Data_Vencimento") /
1000, "yyyy-MM-dd")) \
    .withColumn("Data_Base", from_unixtime(col("Data_Base") / 1000, "yyyy-MM-dd")) \
    .withColumn("dt_update", from_unixtime(col("dt_update") / 1000, "yyyy-MM-dd
HH:mm:ss"))
```

```
# Tratar valores nulos (exemplo: preencher com 0)
df_silver = df_silver.fillna({
    "PUCompraManha": 0,
    "PUVendaManha": 0,
    "PUBaseManha": 0
})
```

```
# Exibir os dados da camada Silver
df_silver.show()
```

```
# Salvar a camada Silver de volta no S3
silver_path = "s3a://my-bucket-ric-01/processed-data/ipca/silver/"
df_silver.write.mode("overwrite").parquet(silver_path)
```

Transformações para a Camada Gold (Agregação e Enriquecimento)

Aqui você agrega os dados para criar métricas analíticas, como:

- Média de PUCompraManha e PUVendaManha.
- Contagem por tipo de título.

Código para a Camada Gold

```
from pyspark.sql.functions import avg, count
```

```
# Calcular métricas agregadas
```

```
df_gold = df_silver.groupBy("Tipo").agg(  
    avg("PUCompraManha").alias("Media_PUCompraManha"),  
    avg("PUVendaManha").alias("Media_PUVendaManha"),  
    count("*").alias("Total_Registros")  
)
```

```
# Exibir os dados agregados (Gold)
```

```
df_gold.show()
```

```
# Salvar a camada Gold no S3
```

```
gold_path = "s3a://my-bucket-ric-01/analytics/ipca/gold/"  
df_gold.write.mode("overwrite").parquet(gold_path)
```

4. Validar os Resultados

- Após o processamento, verifique no bucket 01 no S3:
 - processed-data/ipca/silver/: Dados limpos e transformados.
 - analytics/ipca/gold/: Dados agregados e prontos para análise.

