Fluxo pipeline usando GCP -Dataflow & Big query.

Criador [Vinicius farinelli freire]

ps: antes de tudo para ter sucesso , crie bucket , big query e execução do dataflow todas na mesma Region no meu caso foi us-west1

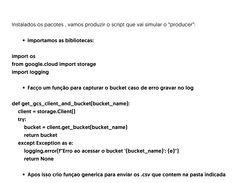
Estrategia adotada: Script "produzer" simulai evio para pasta no cloud storage , apos cloud storage recebeir o Dataflow iniciar 1 do ETL e apos sua finalização carrega no big query , e ao fim faço conexao direta com power bi que usa metodo importar e nao direct query pois os dados vem sob demanda. Ainda sobre o Dataflow podemos usar tanto o pub/sub para ficar "escutando" a pasta e acionar ou agendar um cronjob que inicia a execução do pipeline.

Funil de Processamento de Dados

Primeiro **eu** crio um projeto e, após isso, acesso o Cloud Storage, onde crio um bucket que será a "pasta" para receber os arquivos .csv enviados pelo "produzer.". Também crio subpastas chamadas **templates**, **temp** e **csv_files** para organizar melhor os arquivos CSV recebidos.
Em seguida, crio uma conta de serviço para realizar as autorizações e obter as credenciais em formato .json, necessárias para o acesso local ao GCP. Ao selecionar o papel para o projeto, escolho as opções "básico" e "proprietário". Com a conta de serviço criada, entro nela para gerar a chave de acesso em formato .json e a guardo dentro da pasta **config** do meu projeto.
Feito isso, acesso o link a seguir para ativar as APIs necessárias ao pipeline:
Ativar APIs necessárias

Localmente, na pasta do meu projeto, crio uma pasta **config** e coloco a chave baixada da conta de serviço. Em seguida, crio um ambiente virtual:

Configurando o Pipeline GCP



Comandos para criar ambiente virtual e ativar

```
venv/Scripts/activate
```

```
python -m venv venv
```

Depois vamos setar o caminho da credencial json com global no terminal e verificar se deu certo

```
set GOOGLE_APPLICATION_CREDENTIALS=[CAMINHO_DO_ARQUIVO_CREDENCIAIS]
```

```
echo $env:GOOGLE_APPLICATION_CREDENTIALS
```

Instalando as bibliotecas necessarias:

```
apache-beam==2.61.0 google-api-core==2.23.0 google-apitools==0.5.31
google-auth==2.36.0 google-cloud-bigquery==3.27.0
google-cloud-bigquery-storage==2.27.0 google-cloud-bigtable==2.27.0
google-cloud-core==2.4.1 google-cloud-datastore==2.20.1 google-cloud-pubsub==2.27.1
jsonpickle==3.4.2 jsonschema==4.23.0 numpy==2.1.3 pandas==2.2.3
```

Instalados os pacotes , vamos produzir o script que vai simular o "produzer":

- **importamos as bibliotecas:**

```
import os
from google.cloud import storage
import logging
```

- **Façco um função para capturar o bucket caso de erro gravar no log**

```
def get_gcs_client_and_bucket(bucket_name):
    client = storage.Client()
    try:
        bucket = client.get_bucket(bucket_name)
        return bucket
    except Exception as e:
        logging.error(f"Erro ao acessar o bucket '{bucket_name}': {e}")
        return None
```

- **Apos isso crio funçao generica para enviar os .csv que contem na pasta indicada**

```
def upload_file_to_gcs(local_file_path, bucket_name, destination_blob_name):
    if not os.path.exists(local_file_path):
        logging.error(f"Erro: Arquivo '{local_file_path}' não encontrado.")
        return
    # pega o bucket
    bucket = get_gcs_client_and_bucket(bucket_name)
    if bucket:
        # Cria o arquivo no gcp
        blob = bucket.blob(destination_blob_name)
        try:
            # Faz o upload do arquivo
            blob.upload_from_filename(local_file_path)
            logging.info(f"Arquivo '{local_file_path}' enviado para '{bucket_name}/{destination_blob_name}'.")
        except Exception as e:
            logging.error(f"Erro ao fazer upload do arquivo '{local_file_path}': {e}")
```

- **crio um** app.py **apenas para execução, bem simples , importando as configs e o script de upload**

```
import logging
import os
from service.upload import upload_file_to_gcs
from config.config import BUCKET_NAME, LOCAL_FILE_PATH, GCS_SUBFOLDER
# Configuração do logs
logging.basicConfig(filename='log/app.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
def upload_all_files():
    # Passa por todos os arquivos da pasta com final csv
    for filename in os.listdir(LOCAL_FILE_PATH):
        if filename.endswith('.csv'):
            local_file_path = os.path.join(LOCAL_FILE_PATH, filename)
            destination_blob_name = f"{GCS_SUBFOLDER}/{filename}"  # Define o nome no GCS
            upload_file_to_gcs(local_file_path, BUCKET_NAME, destination_blob_name)

if name == "__main__":
    upload_all_files()Vera algo assim no LOG:
```

```
2024-12-06 11:06:55,590 - INFO - Arquivo 'C:\Users\vinic\OneDrive\Área de Trabalho\nexus\arquivos\customers_file3.csv' enviado para 'nxs/csv_files/customers_file3.csv'.
2024-12-06 11:06:56,382 - INFO - Arquivo 'C:\Users\vinic\OneDrive\Área de Trabalho\nexus\arquivos\transactions_file1.csv' enviado para 'nxs/csv_files/transactions_file1.csv'.
2024-12-06 11:06:57,157 - INFO - Arquivo 'C:\Users\vinic\OneDrive\Área de Trabalho\nexus\arquivos\transactions_file2.csv' enviado para 'nxs/csv_files/transactions_file2.csv'.
```

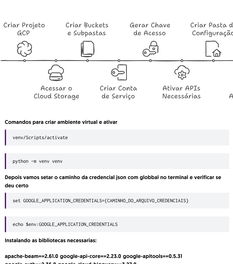
Feito isso , comecei as construções do pipeline.py com apache beam, baixado as bibliotecas ,primeiro vamos fazer um teste com o parametro do pipeline setado para **DirectRunner** que testa o fluxo **localmente**.

- **Configuração do Pipeline (PipelineOptions):** primeiro configuramos o pipeline para rodar no DirectRunner para teste local.
  São passados parâmetros como:
- **Runner**: DirectRunner, indicando que a execução será local.
- **DataflowRunner**: significa que será executado no dataflow
- **Project, Region, Temp Location e Staging Location**: indicando onde o serviço do Dataflow vai rodar, armazenar arquivos temporários etc

```
pipeline_options = PipelineOptions([
runner='DataflowRunner', # Apos sucesso local mudar paraDataflowRunner
project='project_id',
region='region',
temp_location='gs://{bucket}/temp',
staging_location='gs://{bucket}/temp',
job_name='csv-to-big-pipeline',
#template_location='gs://nxs/templates/csv-to-bq-pipeline', save_main_session=True]
```

No meu caso comentei o template_location pois quero que ele seja executado direto no dataflow.

Apos a execução, do pipeline.py **temos:**



Confirmação dbo load no bigquery:

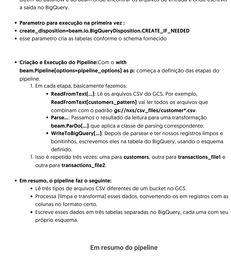
Explicando um pouco codigo:

- **Classes de Parsing (ParseCustomers, ParseTransactionsFile1, ParseTransactionsFile2)**: Essas classes são basicas e simples apenas para tratar cada csv padronizar e checar se datas sao validas , se por exemplo se os campos necessarios nao estao vazios _ que o metodo process acaba falando , no estilo parse and load.
  - Depois do sub ok, a classe "emite" (**yield**) um dicionário contendo os dados processados. Esse dicionário será o que o Beam vai passar para as próximas etapas do pipeline.
- **Definições de caminhos (Pipeline options)**:São definidas algumas variáveis com o ID do projeto, nome do dataset no BigQuery, região e bucket no GCS. Essas informações dizem ao Dataflow e ao Beam onde encontrar os arquivos de entrada e onde escrever a saída no BigQuery.
- **Parametro para execução na primeira vez :**
- **create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED**
- esse parametro cria as tabelas conforme o schema funcação

- **Criação e Execução do Pipeline:**Com o **with beam.Pipeline(options=pipeline_options) as p:** começa a definição das etapas do pipeline.
  1. Em cada etapa, basicamente fazemos:
     - **ReadFromText(...)**: Lê os arquivos CSV do GCS. Por exemplo, **ReadFromText(customers_pattern)** vai ler todos os arquivos que combinam com o padrão **gs://nxs/csv_files/customers*.csv**.
     - **Parse...**: Passamos o resultado da leitura para uma transformação **beam.ParDo(...)** que aplica a classe de parsing correspondente.
     - **WriteToBigQuery(...)**: Depois de parsear e ter nossos registros limpos e bonitinhos, escrevemos eles na tabela do BigQuery, usando o esquema definido.
  1. Isso é repetido três vezes: uma para **customers**, outra para **transactions_file1** e outra para **transactions_file2**.

- **Em resumo, o pipeline faz o seguinte:**
  - Lê três tipos de arquivos CSV diferentes de um bucket no GCS.
  - Processa (limpa e transforma) esses dados, convertendo-os em registros com as colunas no formato certo.
  - Escreve esses dados em três tabelas separadas no BigQuery, cada uma com seu próprio esquema.

Em resumo do pipeline



Apos os job ter rodado , faço as conexoes com power bi e a criação do painel , com mesmos resultados do query .

Link: Microsoft Power BI

Vinicius Farinelli Freire dia 08/12/2024