



escola  
britânica de  
artes criativas  
& tecnologia

# Profissão: Analista de dados



## 4º PROJETO: PIPELINE DE DADOS DO TELEGRAM



## GUIA DA AULA 5



# Entenda o ETL



Acompanhe aqui os temas que serão tratados na videoaula

● **Introdução**

● **AWS S3**

● **AWS Lambda**

● **AWS Event Bridge**



# 1. Introdução

A etapa de **extração, transformação e carregamento** (do inglês *extraction, transformation and load* ou **ETL**) é uma etapa abrangente responsável pela manipulação dos dados ingeridos de sistemas transacionais, ou seja, já persistidos em camadas cruas ou *raw* de sistemas analíticos. Os processos conduzidos nesta etapa variam bastante de acordo com a área da empresa, do volume/variedade/velocidade do dado consumido, etc.



Contudo, em geral, o dado cru ingerido passa por um processo recorrente de *data wrangling* onde o dado é limpo, deduplicado, etc. e persistido com técnicas de particionamento, orientação a coluna e compressão. Por fim, o dado processado está pronto para ser analisado por profissionais de dados.



No projeto, as mensagens de um único dia, persistidas na camada cru, serão compactadas em um único arquivo, orientado à coluna e comprimido, que será persistido em uma camada enriquecida. Além disso, durante este processo, o dado também passará por etapas de *data wrangling*.



Para isso, vamos utilizar uma função do `AWS Lambda` como motor de processamento e um *bucket* do `AWS S3` como camada enriquecida para a persistência do dado processado. Para garantir a recorrência, vamos configurar uma regra do `AWS Event Bridge` como gatilho diário da função.



## 2. AWS S3

Na etapa de **ETL**, o `AWS S3` tem a função de passivamente armazenar as mensagens processadas de um dia em um único arquivo no formato Parquet. Para tanto, basta a criação de um *bucket*. Como padrão, vamos adicionar o sufixo `-enriched` ao seu nome (vamos seguir esse padrão para todos os serviços desta camada).





**Nota:** um `data lake` é o nome dado a um repositório de um grande volume de dados. É organizado em zonas que armazenam replicadas dos dados em diferentes níveis de processamento. A nomenclatura das zonas varia, contudo, as mais comuns são: *raw* e *enriched* ou *bronze*, *silver* e *gold*.



### 3. AWS Lambda

Na etapa de **ETL**, o AWS Lambda tem a função de ativamente processar as mensagens captadas pelo *bot* do Telegram, persistidas na camada cru no *bucket* do AWS S3, e persisti-las na camada enriquecida, também em um *bucket* do AWS S3.



Logo, vamos criar uma função que opera da seguinte forma:

- Lista todos os arquivos JSON de uma única participação da camada crua de um *bucket* do AWS S3;
- Para cada arquivo listado:
  - Faz o *download* do arquivo e carrega o conteúdo da mensagem;
  - Executa uma função de *data wrangling*;
  - Cria uma tabela do PyArrow e a concatena com as demais
- Persiste a tabela no formato Parquet na camada enriquecida em um *bucket* do AWS S3.



**Nota:** O fato de utilizarmos duas camadas de armazenamento e processamento, permite que possamos reprocessar os dados crus de diversas maneiras, quantas vezes forem preciso.

**Nota:** Atente-se ao fato de que a função processa as mensagens do dia anterior (D-1).



## O código da função:

In [ ]:

```

import os import
json import
logging

from datetime import datetime, timedelta, timezone

import boto3
import pyarrow as pa
import pyarrow.parquet as pq

def lambda_handler(event: dict, context: dict) -> bool: '''
    Diariamente é executado para compactar as diversas mensagensm, no format JSON,
    do dia anterior, armazenadas no bucket de dados cru, em um único arquivo no
    formato PARQUET, armazenando-o no bucket de dados enriquecido '''

```



## O código da função:

In [ ]:

```

# vars de ambiente

RAW_BUCKET = os.environ['AWS_S3_BUCKET']
ENRICHED_BUCKET = os.environ['AWS_S3_ENRICHED']

# vars lógicas

tzinfo = timezone(offset=timedelta(hours=-3))
date = (datetime.now(tzinfo) - timedelta(days=1)).strftime('%Y-%m-%d')
timestamp = datetime.now(tzinfo).strftime('%Y%m%d%H%M%S%f')

# código principal

table = None
client = boto3.client('s3')

try:

    response = client.list_objects_v2 (
        Bucket=RAW_BUCKET,
        Prefix=f'telegram/context_date={ date}'
    )
  
```



## O código da função:

In [ ]:

```

for content in response['Contents']:

    key = content['Key']
    client.download_file(
        RAW_BUCKET,
        key,
        f"/tmp/{key.split('/')[ -1 ]}")

    with open(
        f"/tmp/{key.split('/')[ -1 ]}",
        mode='r',
        encoding='utf8'
    ) as fp:

        data = json.load(fp)
        data = data["message"]

    parsed_data = parse_data(data=data)
    iter_table = pa.Table.from_pydict(mapping=parsed_data)

    if table:

```



## O código da função:

In [ ]:

```

    table = pa.concat_tables([table, iter_table])

    else:

        table = iter_table
        iter_table = None

    pq.write_table(table=table, where=f'/tmp/{timestamp}.parquet')
    client.upload_file(
        f"/tmp/{timestamp}.parquet",
        ENRICHED_BUCKET,
        f"telegram/context_date={ date}/{timestamp}.parquet"
    )

    return True

except Exception as exc:
    logging.error(msg=exc)
    return
    False

```





## O código da função *data wrangling*:

In [ ]:

```
def parse_data(data: dict) -> dict:

    date = datetime.now().strftime('%Y-%m-%d')
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    parsed_data = dict()

    for key, value in data.items():

        if key == 'from':
            for k, v in data[key].items():
                if k in ['id', 'is_bot', 'first_name']:
                    parsed_data[f"{key if key == 'chat' else 'user'}_{k}"] = [
```



## O código da função *data wrangling*:

In [ ]:

```

elif key == 'chat':
    for k, v in data[key].items():
        if k in ['id', 'type']:
            parsed_data[f"{key if key == 'chat' else 'user'}_{k}"] = [

elif key in ['message_id', 'date', 'text']:
    parsed_data[key] = [value]

if not 'text' in parsed_data.keys():
    parsed_data['text'] = [None]

return parsed_data

```

Para que a função funcione corretamente, algumas configurações precisam ser realizadas.



- **Variáveis do ambiente**

Note que o código exige a configuração de duas variáveis de ambiente: `AWS_S3_BUCKET` e `AWS_S3_ENRICHED` com os nomes dos *bucket* do AWS S3 da camada cru e enriquecida, respectivamente. Para adicionar variáveis de ambiente em uma função do AWS Lambda, basta acessar configurações -> variáveis de ambiente no console da função.



- **Permissão**

Precisamos adicionar a permissão de escrita nos *buckets* do AWS S3 para a função do AWS Lambda no AWS IAM.

- **Recursos**

O *timeout* padrão de funções do AWS Lambda é de 3 segundos. Para a função, vamos aumentar o tempo para 5 minutos, principalmente para lidar com o IO (*input/output*) de arquivos do AWS S3.



- **Camadas**

Por fim, note que o código da função utiliza o pacote Python PyArrow. Contudo, o ambiente padrão do `AWS Lambda` possui poucos pacotes externos instalados, como o pacote Python boto3, logo o PyArrow não será encontrado e a execução da função falhará. Existem algumas formas de adicionar pacotes externos no ambiente de execução do `AWS Lambda`, um deles é a criação de camadas ou *layers*, onde podemos fazer o *upload* dos pacotes Python direto na plataforma ou através de um *bucket* do `AWS S3`.



Vamos então seguir com a última opção, onde teremos que:

- Criar um *bucket* no AWS S3;
- Fazer o *upload* do código do pacote Python do PyArrow (*download* no link <https://github.com/aws/aws-sdk-pandas/releases>);
- Criar *layer* e conectar na função.



## 4. AWS Event Bridge

Na etapa de **ETL**, o AWS Event Bridge tem a função de ativar diariamente a função de ETL do AWS Lambda, funcionando assim como um *scheduler*.

**Nota:** Atente-se ao fato de que a função processa as mensagens do dia anterior (D-1).

