escola britânica de artes criativas & tecnologia

Profissão: Analista de dados

escola britânica de artes criativas & tecnologia

4º PROJETO: PIPELINE DE DADOS DO TELEGRAM

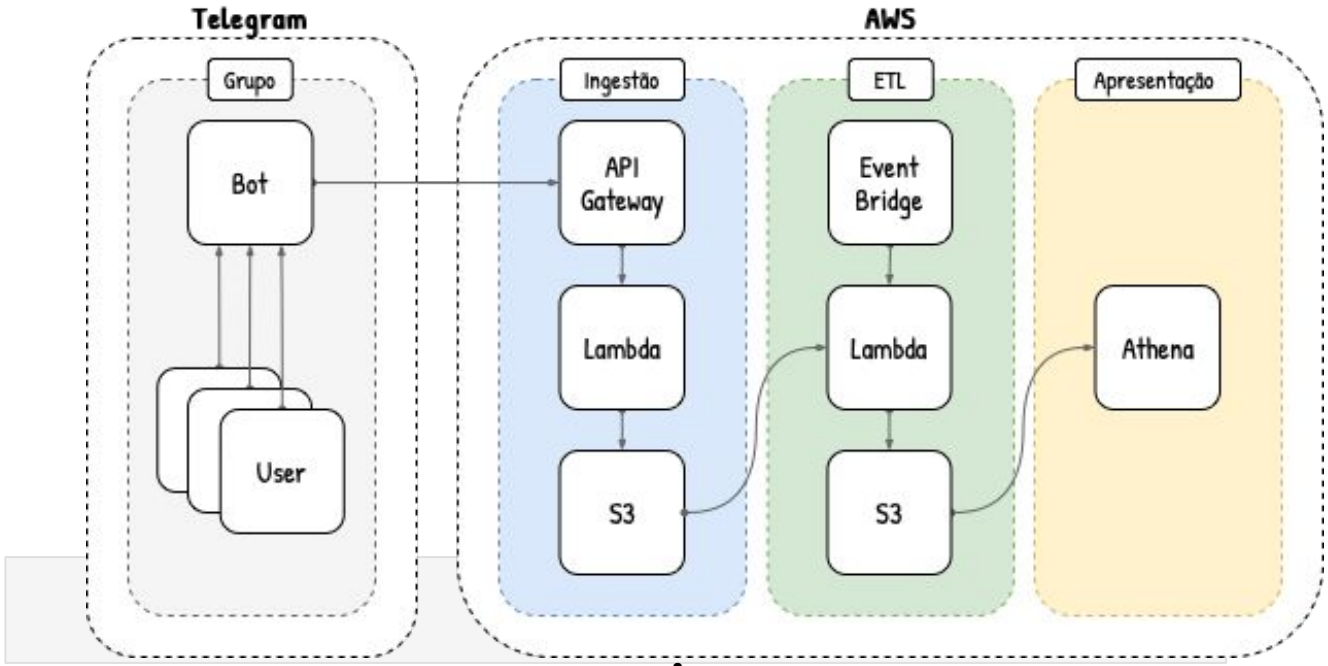
GUIA DA AULA 4

Conheça a ingestão

Acompanhe aqui os temas que serão tratados na videoaula

- **Introdução**
- **AWS S3**
- **AWS Lambda**
- **AWS API Gateway**
- **Telegram**

1. Introdução



Arquitetura

- **Telegram**

As mensagens captadas por um *bot* podem ser acessadas via API. A única informação necessária é o `token` de acesso fornecido pelo `BotFather` na criação do *bot*.

Nota: A documentação completa da API pode ser encontrada no *link* https://core.telegram.org/bots/api

In [ ]:
```
from getpass import getpass

token = getpass()
```

A `url` base é comum a todos os métodos da API.

In [ ]:
```
import json

base_url = f'https://api.telegram.org/bot{ token}'
```

- **getMe**

O método `getMe` retorna informações sobre o *bot*.

In [ ]:

```
import requests
response = requests.get(url=f'{base_url}/getMe')


print(json.dumps(json.loads(response.text), indent=2))
```

- **getUpdates**

O método `getUpdates` retorna informações sobre o *bot*.

In [ ]:
```
response = requests.get(url=f'{base_url}/getUpdates')

print(json.dumps(json.loads(response.text), indent=2))
```

A etapa de ingestão é responsável, como seu próprio nome diz, pela ingestão dos dados transacionais em ambientes analíticos. De maneira geral, o dado ingerido é persistido no formato mais próximo do original, ou seja, nenhuma transformação é realizada em seu conteúdo ou estrutura (*schema*). Como exemplo, dados de uma API *web* que segue o formato REST (*representational state transfer*) são entregues, logo, persistidos, no formato JSON.

Persistir os dados em seu formato original traz muitas vantagens, como a possibilidade de reprocessamento.

Pode ser conduzida de duas formas:

- **Batch**: blocos de dados são ingeridos em uma frequência bem definida, geralmente na escala de horas ou dias;
- **Streaming**: dados são ingeridos conforme são produzidos e disponibilizados.

No projeto, as mensagens capturadas pelo bot podem ser ingeridas através da API *web* de *bots* do **Telegram**, portanto são fornecidas no formato JSON. Como o **Telegram** retém mensagens por apenas 24h em seus servidores, a ingestão via **streaming** é a mais indicada. Para que seja possível esse tipo de **ingestão** seja possível, vamos utilizar um *webhook* (gancho *web*), ou seja, vamos redirecionar as mensagens automaticamente para outra API *web*.

Sendo assim, precisamos de um serviço da AWS que forneça um API *web* para receber os dados redirecionados, o **`AWS API Gateway`** (documentação no *link* https://docs.aws.amazon.com/pt_br/apigateway/latest/developerguide/welcome.html). Dentre suas diversas funcionalidades, o **`AWS API Gateway`** permite o redirecionamento do dado recebido para outros serviços da AWS. Logo, vamos conectá-lo ao **`AWS Lambda`**, que por sua vez, irá armazenar o dado em seu formato original (JSON) em um *bucket* do `AWS S3`.

Sistemas que reagem a eventos são conhecidos como *event-driven*.

Portanto, precisamos:

- Criar um *bucket* no `AWS S3`;
- Criar uma função no `AWS Lambda`;
- Criar uma API *web* no `AWS API Gateway`;
- Configurar o *webhook* da API de *bots* do Telegram.

2. AWS S3

Na etapa de ingestão, o `AWS S3` tem a função de passivamente armazenar as mensagens captadas pelo *bot* do Telegram no seu formato original: JSON. Para tanto, basta a criação de um *bucket*. Como padrão, vamos adicionar o sufixo `-raw` 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 réplicas 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 **ingestão**, o `AWS Lambda` tem a função de ativamente persistir nas mensagens captadas pelo *bot* do **Telegram** em um *bucket* do `AWS S3`. Para tanto vamos criar uma função que opera da seguinte forma:

- Recebe a mensagem no parâmetro `event`;
- Verifica se a mensagem tem origem no grupo do **Telegram** correto;
- Persiste a mensagem no formato JSON no *bucket* do `AWS S3`;
- Retorna uma mensagem de sucesso (código de retorno HTTP igual a 200) a API de *bots* do **Telegram**.

Nota: No **Telegram**, restringimos a opção de adicionar o *bot* a grupos, contudo, ainda é possível iniciar uma conversa em um *chat* privado.

O código da função:

In [ ]:

```
import os
import json
import
logging
from datetime import datetime, timezone

import boto3
```

O código da função:

In [ ]:

```
def lambda_handler(event: dict, context: dict) -> dict:

  '''

  Recebe uma mensagens do Telegram via AWS API Gateway, verifica no
  seu conteúdo se foi produzida em um determinado grupo e a escreve,
  em seu formato original JSON, em um bucket do AWS S3.
  '''

  # vars de ambiente

  BUCKET = os.environ['AWS_S3_BUCKET']
  TELEGRAM_CHAT_ID = int(os.environ['TELEGRAM_CHAT_ID' ])

  # vars lógicas

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

  filename = f'{timestamp}.json'
```

O código da função:

In [ ]:

```
# código principal

client = boto3.client('s3')

try:

  message = json.loads(event["body"])
  chat_id =
  message["message"]["chat"]["id"]

  if chat_id == TELEGRAM_CHAT_ID:

    with open(f"/tmp/{filename}", mode='w', encoding='utf8') as fp:
      json.dump(message, fp)

    client.upload_file(
        f'/tmp/{filename}',
        BUCKET,
        f'telegram/context_date={date}/{filename}'


        )
```

O código da função:

In [ ]:

```
            except Exception as exc:
                logging.error(msg=exc)
                return dict(statusCode="500")

            else:
                return dict(statusCode="200")
```

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

- **Variáveis de ambiente**

Note que o código exige a configuração de duas variáveis de ambiente:

`AWS_S3_BUCKET` com o nome do *bucket* do `AWS S3` e `TELEGRAM_CHAT_ID` com o id do *chat* do grupo do **Telegram**. Para adicionar variáveis de ambiente em uma função do `AWS Lambda`, basta acessar configurações -> variáveis de ambiente no console da função.

Nota: Variáveis de ambiente são excelentes formas de armazenar informações sensíveis.

- **Permissão:**

Por fim, precisamos adicionar a permissão de escrita no *bucket* do `AWS S3` para a função do `AWS Lambda` no `AWS IAM` .

4. AWS API Gateway

Na etapa de **ingestão**, o `AWS API Gateway` tem a função de receber as mensagens captadas pelo *bot* do **Telegram**, enviadas via *webhook*, e iniciar uma função do `AWS Lambda`, passando o conteúdo da mensagem no seu parâmetro *event*. Para tanto vamos criar uma API e configurá-la como gatilho da função do `AWS Lambda`:

- Acesse o serviço e selecione: *Create API -> REST API*;
- Insira um nome, como padrão, um que termine com o sufixo `-api` ;
- Selecione: *Actions -> Create Method -> POST*;
- Na tela de *setup*:
  - Selecione *Integration type* igual a *Lambda Function*;
  - Habilite o *Use Lambda Proxy integration*;
  - Busque pelo nome a função do `AWS Lambda`.

Podemos testar a integração com o `AWS Lambda` através da ferramenta de testes do serviço. Por fim, vamos fazer a implantação da API e obter o seu endereço web.

- Selecione: *Actions -> Deploy API;*
- Selecione: *New Stage* para *Deployment stage*;
- Adicione *dev* como `Stage name`.

Copie o a `url` gerada na variável `aws_api_gateway_url` .

In [ ]:
```
aws_api_gateway_url = getpass()
```

5. Telegram

Vamos configurar o `webhook` para redirecionar as mensagens para a `url` do `AWS API Gateway`.

- **setWebhook**

O método setWebhook configura o redirecionamento das mensagens captadas pelo *bot* para o endereço *web* do parametro `url`.

Nota: os métodos `getUpdates` e `setWebhook` são mutuamente exclusivos, ou seja, enquanto o *webhook* estiver ativo, o método `getUpdates` não funcionará. Para desativar o *webhook*, basta utilizar o método `deleteWebhook`.

In [ ]:

```
response = (requests
                .get(url=f'{base_url}
                /setWebhook?url={aws_api_gateway_url }')
               )

 print(json.dumps(json.loads(response.text), indent=2))
```

- **getWebhookInfo**

O método `getWebhookInfo` retorna as informações sobre o *webhook* configurado.

In [ ]:

```
response = (requests
              .get(url=f'{base_url}
              /getWebhookInfo')
             )

print(json.dumps(json.loads(response.text), indent=2))
```