PROJETO PIPELINE DE DADOS DO TELEGRAM

Tópicos

Contexto; Criação do Bot; Bot API; Dados; Ingestão; ETL; Apresentação; Conclusão.

1. Contexto:

O que é Chatbot?

Um chatbot é um programa de computador ou um tipo de inteligência artificial (IA) projetado para interagir com seres humanos por meio de conversas em linguagem natural. Ele simula uma conversa humana, permitindo que os usuários enviem perguntas, solicitem informações, realizem tarefas e recebam respostas automáticas. Os chatbots podem ser encontrados em diversos canais de comunicação, como sites, aplicativos de mensagens, redes sociais e assistentes virtuais. Eles são programados para entender a linguagem humana, interpretar as intenções do usuário e fornecer respostas relevantes ou executar ações específicas com base nessas interações. Eles são uma ferramenta eficaz para automatizar interações comuns, melhorar a eficiência e proporcionar uma experiência de usuário mais conveniente.

O que é Telegram?

O Telegram é um aplicativo de mensagens instantâneas seguro baseado em nuvem. Ele permite que os usuários enviem mensagens, áudio, vídeo e outros arquivos uns aos outros. O Telegram enfatiza a privacidade, oferecendo criptografia de ponta a ponta e recursos de bate-papo secreto. Além das conversas individuais, ele permite criar grupos de até 200.000 membros e oferece canais para compartilhar conteúdo com um público maior. O Telegram está disponível em várias plataformas e possui recursos adicionais, como a criação de bots e integração de aplicativos de terceiros.

O que é um Pipeline de Dados?

Um pipeline de dados é uma série de etapas de processamento para preparar dados corporativos para análise. As organizações têm um grande volume de dados de várias fontes, como aplicativos, dispositivos de Internet das Coisas (IoT) e outros canais digitais. No entanto, os dados brutos são inúteis; eles devem ser movidos, classificados, filtrados, reformatados e analisados para business intelligence. Um pipeline de dados inclui várias tecnologias para verificar, resumir e encontrar padrões nos dados para informar as decisões de [illegible] Pipelines de dados bem organizados oferecem suporte a vários

projetos de big data, como visualizações de dados, análises exploratórias de dados e tarefas de machine learning. (Fonte: AMAZON AWS - https://aws.amazon.com/pt/what-is/data-pipeline/)

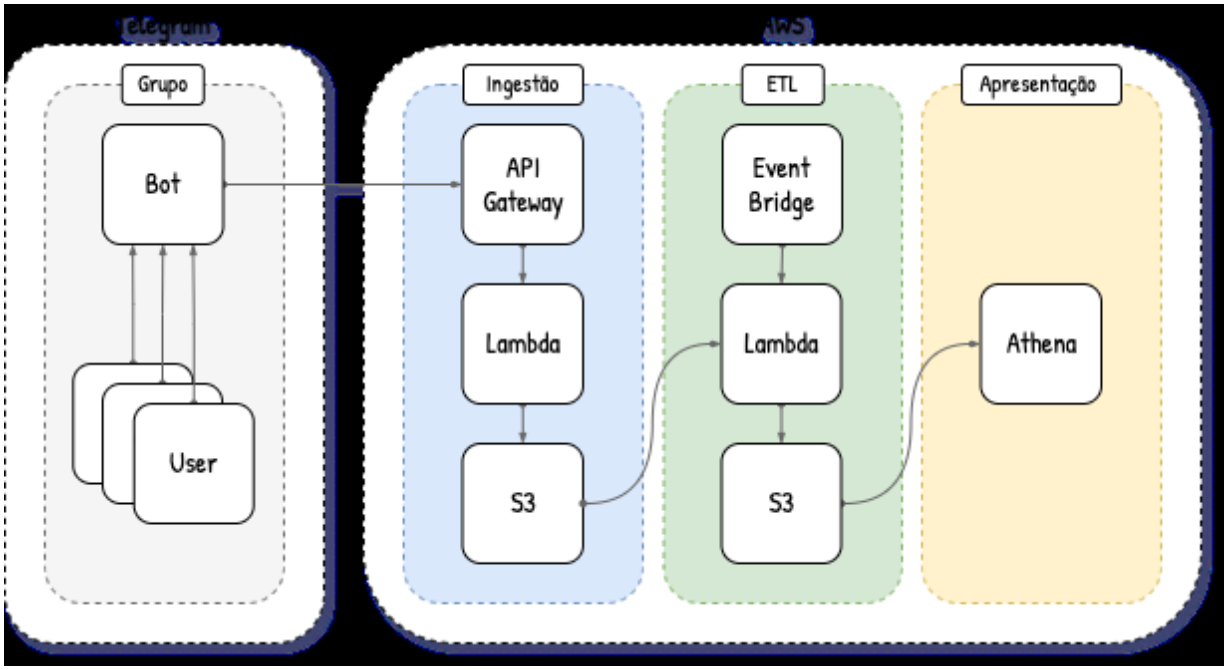
Neste projeto construirei um pipeline de dados que ingira, processe, armazene e exponha mensagens de um grupo do Telegram para que profissionais de dados possam realizar análises. A arquitetura proposta é dividida em duas: transacional, no Telegram, onde os dados são produzidos, e analítica, na Amazon Web Services (AWS), onde os dados são analisados.

In [ ]:
```
!wget -O '/content/ESTRUTURA.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
import cv2
from google.colab.patches import cv2_imshow
#A função cv2.imshow é incompatível com Jupyter
#https://github.com/jupyter/notebook/issues/3935
```

In [ ]:
```
imagem = cv2.imread('/content/ESTRUTURA.png')
cv2_imshow(imagem)
```



O Telegram representa a fonte de dados transacionais. Mensagens enviadas por usuários em um grupo são capturadas por um bot e redirecionadas via webhook do backend do aplicativo para um endpoint (endereço web que aceita requisições HTTP) exposto pelo AWS API Gateway. As mensagens trafegam no corpo ou payload da requisição.

AWS | Ingestão

Uma requisição HTTP com o conteúdo da mensagem em seu payload é recebia pelo AWS API Gateway que, por sua vez, as redireciona para o AWS Lambda, servindo assim como seu gatilho. Já o AWS Lambda recebe o payload da requisição em seu parâmetro event, salva o conteúdo em um arquivo no formato JSON (original, mesmo que o payload) e o armazena no AWS S3 particionado por dia.

AWS | ETL

Uma vez ao dia, o AWS Event Bridge aciona o AWS Lambda que processa todas as mensagens do dia anterior (atraso de um dia ou D-1), denormaliza o dado semi-estruturado típico de arquivos no formato JSON, salva o conteúdo processado em um arquivo no formato Apache Parquet e o armazena no AWS S3 particionado por dia.

AWS | Apresentação

Por fim, uma tabela do AWS Athena é apontada para o bucket do AWS S3 que armazena o dado processado: denormalizado, particionado e orientado a coluna. Profissionais de dados podem então executar consultas analíticas (agregações, ordenações, etc.) na tabela utilizando o SQL para a extração de insights.

2. Criação do Bot:

Com a conta criada no Telegram, abra o chat com o BotFather e para a criação do BOT:

Digite \start:

Digite \newbot

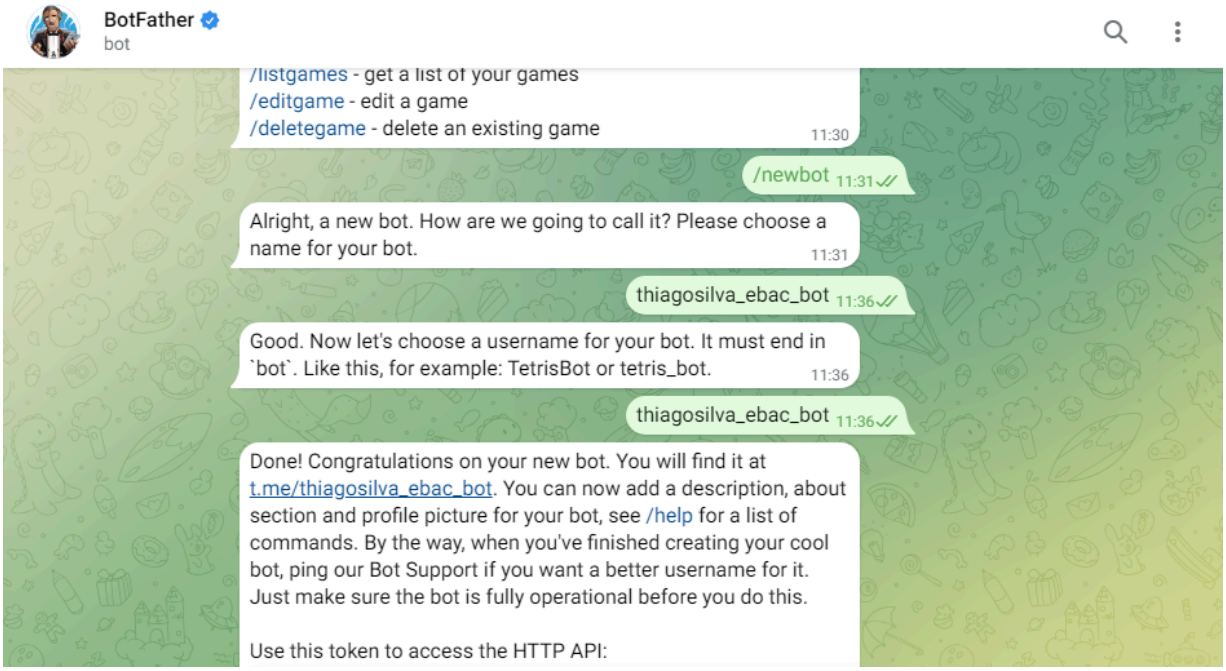
Digite o nome do bot, e logo após o nome de usuário (precisa terminar com o sufixo _bot)

OBS: Será necessário salvar o token de acesso a API HTTP em um local seguro.

In [ ]:
```
!wget -O '/content/BOT - THIAGO.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/BOT - THIAGO.png')
cv2_imshow(imagem)
```



Para ativar o BOT:

Iniciei o chat com o BOT, e digitei \start.

Criação do grupo com o Bot

Com o grupo criado, adicionei o BOT como administrador para receber todas as mensagens do grupo. Uma outra opção seria desabilitar o seu modo de privacidade.

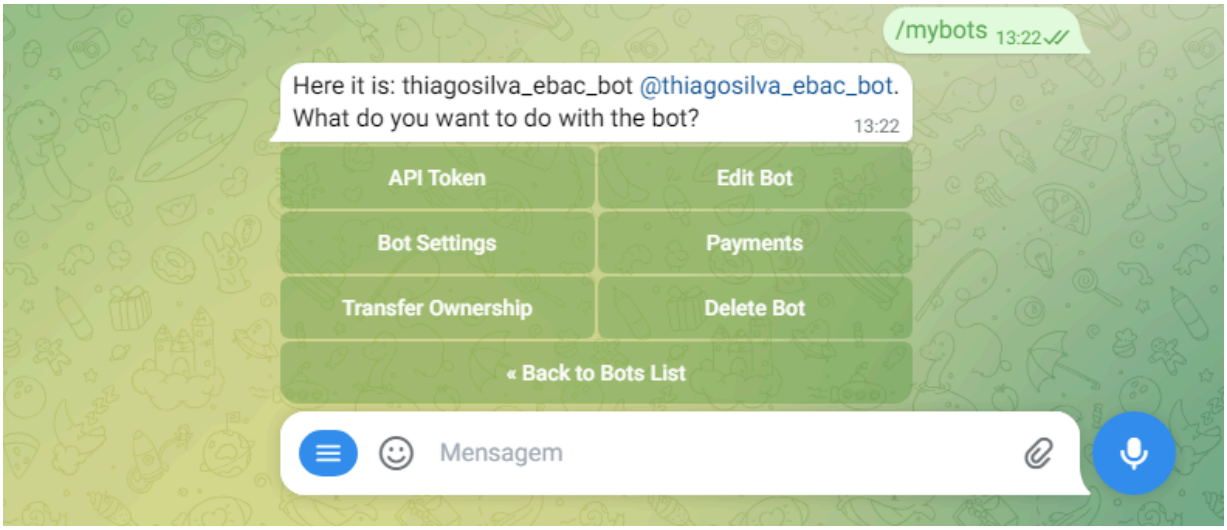
Abra o chat com o BotFather:

Digite /mybots; Selecione o bot pelo seu nome de usuário; Selecione Bot Settings; Selecione Allow Groups?; Selecione Turn groups off.

In [ ]:
```
!wget -O '/content/MYBOT.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

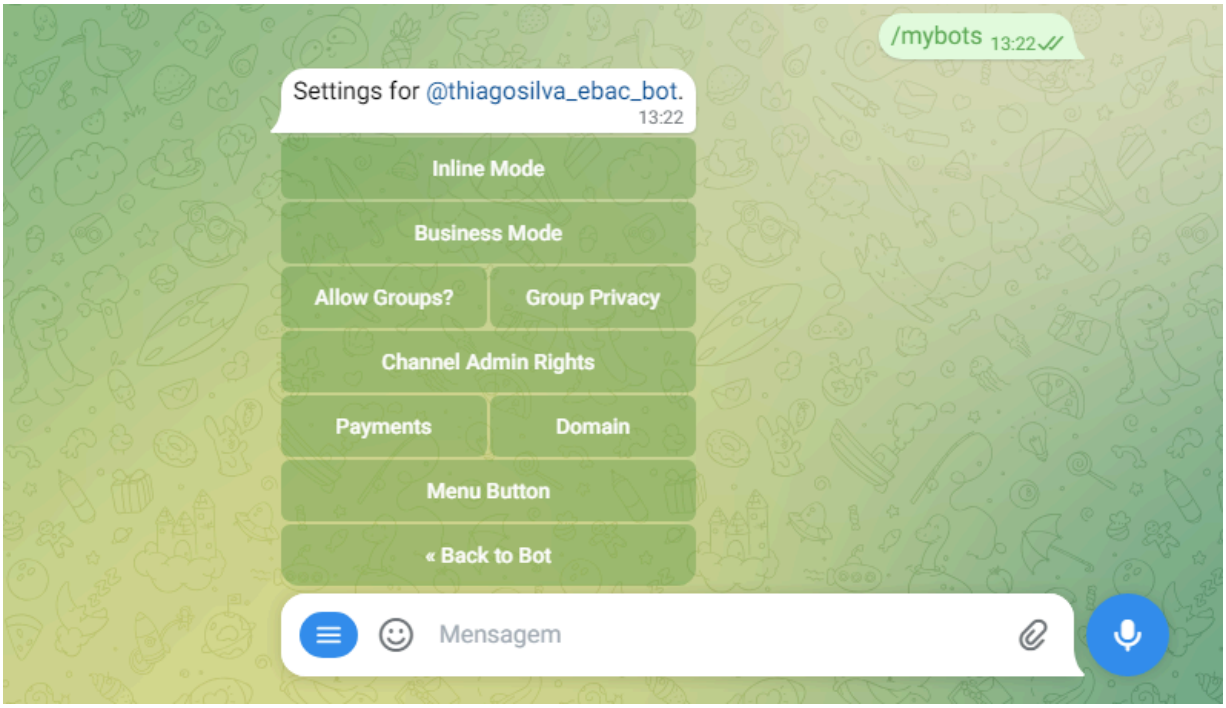
In [ ]:
```
imagem = cv2.imread('/content/MYBOT.png')
cv2_imshow(imagem)
```



In [ ]:
```
!wget -O '/content/MYBOT 01.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

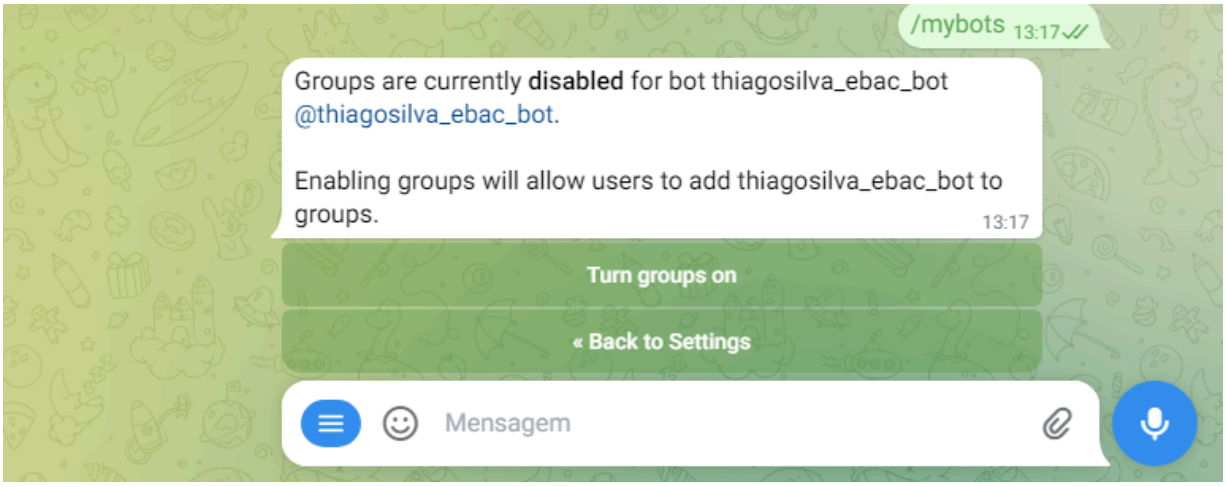
In [ ]:
```
imagem = cv2.imread('/content/MYBOT 01.png')
cv2_imshow(imagem)
```



In [ ]:
```
!wget -O '/content/CONFIGURAÇÃO - BOT.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/CONFIGURAÇÃO - BOT.png')
cv2_imshow(imagem)
```



3. Bot API: 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. Para isso, vamos utilizar o pacote nativo do Python getpass para ingerir o token, e não deixá-lo exposto no Google colab.

In [ ]:
```
from getpass import getpass

token = getpass()
```

.....

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

In [ ]:
```
import json

import requests

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

4. Dados

Antes de avançar para etapa analítica, vamos trabalhar na manipulação dos dados de mensagens do Telegram. Mensagem:

Uma mensagem recuperada via API é um dado semi-estruturado no formato JSON com algumas chaves mandatórias e diversas chaves opcionais, estas últimas presentes (ou não) dependendo do tipo da mensagem. Por exemplo, mensagens de texto apresentam a chave text enquanto mensagens de áudio apresentam a chave audio. Neste projeto vamos focar em mensagens do tipo texto, ou seja, vamos ingerir as chaves mandatórias e a chave text.

In [ ]:
```
%%writefile telegram.json
{
    "update_id": 123,
    "message": {
        "message_id": 1,
        "from": {
            "id": 321,
            "is_bot": false,
            "first_name": "thiago"
        },
        "chat": {
            "id": -789,
            "type": "group"
        },
        "date": 1640995200,
        "text": "Ola, mundo!"
    }
}
```

```
Writing telegram.json
```

In [ ]:
```
import json

with open('telegram.json', mode='r', encoding='utf8') as fp:
  data = json.load(fp)
  data = data["message"]
```

In [ ]:
```
print(json.dumps(data, indent=2))
```

```
{
  "message_id": 1,
  "from": {
    "id": 321,
    "is_bot": false,
    "first_name": "thiago"
  },
  "chat": {
    "id": -789,
    "type": "group"
  },
  "date": 1640995200,
  "text": "Ola, mundo!"
}
```

In [ ]:
```
from datetime import datetime

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}"] = [v]

    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}"] = [v]

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

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

parsed_data['context_date'] = [date]
parsed_data['context_timestamp'] = [timestamp]
```

In [ ]:
```
for k, v in parsed_data.items():
  print(f"{k}: {v}")
```

```
message_id: [1]
user_id: [321]
user_is_bot: [False]
user_first_name: ['thiago']
chat_id: [-789]
chat_type: ['group']
date: [1640995200]
text: ['Ola, mundo!']
context_date: ['2024-04-25']
context_timestamp: ['2024-04-25 14:51:51']
```

In [ ]:
```
import pyarrow as pa

table = pa.Table.from_pydict(mapping=parsed_data)
```

In [ ]:
```
table
```

Out[ ]:
```
pyarrow.Table
message_id: int64
user_id: int64
user_is_bot: bool
user_first_name: string
chat_id: int64
chat_type: string
date: int64
text: string
context_date: string
context_timestamp: string
----
message_id: [[1]]
user_id: [[321]]
user_is_bot: [[false]]
user_first_name: [["thiago"]]
chat_id: [[-789]]
chat_type: [["group"]]
date: [[1640995200]]
text: [["Ola, mundo!"]]
context_date: [["2024-04-25"]]
context_timestamp: [["2024-04-25 14:51:51"]]
```

Descrição:

In [ ]:
```
!wget -O '/content/DADOS.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/DADOS.png')
cv2_imshow(imagem)
```

chave	tipo valor	opcional	descrição
updated_id	int	não	id da mensagem enviada ao **bot**
message_id	int	não	id da mensagem enviada ao grupo
from_id	int	sim	id do usuário que enviou a mensagem
from_is_bot	bool	sim	se o usuário que enviou a mensagem é um **bot**
from_first_name	str	sim	primeiro nome do usário que enviou a mensagem
chat_id	int	não	id do *chat* em que a mensagem foi enviada
chat_type	str	não	tipo do *chat*: private, group, supergroup ou channel
date	int	não	data de envio da mensagem no formato unix
text	str	sim	texto da mensagem

5. Ingestão

A etapa de ingestão é responsável, como seu o 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.

Nota: Persistir os dados em seu formato original trás 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 fornecidos no formato JSON. Como o Telegram retem 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 neste link). Dentre suas diversas funcionalidades, o AWS API Gateway permite o redirecionamento do dado recebido para outros serviços da AWS. Logo, vamos conecta-lo ao AWS Lambda, que por sua vez, irá armazenar o dado em seu formato original (JSON) em um bucket do AWS S3.

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).

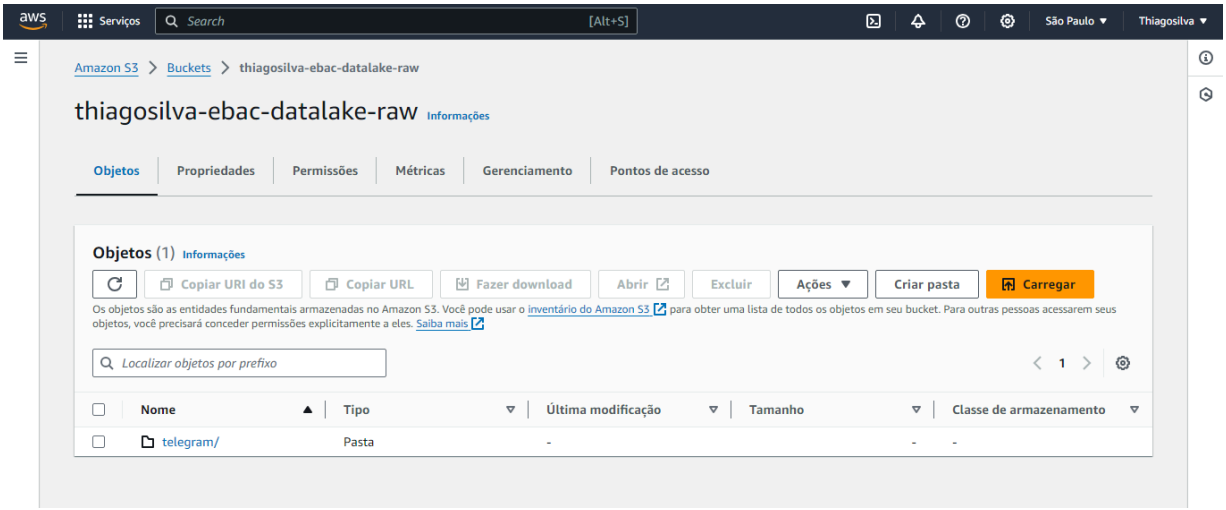
Nota: um data lake é o nome dado a um repositório de um grande volume dados. É organizado em zonas que armazenam replicadas dos dados em diferentes níveis de processamento. A nomenclatura das zonas varia, contudo, as mais comuns são: raw e enriched ou bronze, silver e gold.

Criei um bucket no AWS S3

In [ ]:
```
!wget -O '/content/thiagosilva - raw.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/thiagosilva - raw.png')
cv2_imshow(imagem)
```



AWS Lambda

Na etapa de ingestão, o AWS Lambda tem a função de ativamente persistir as mensagens captadas pelo bot do Telegram em um bucket do AWS S3. Para tanto vamos criar uma função que opera da seguinte forma:

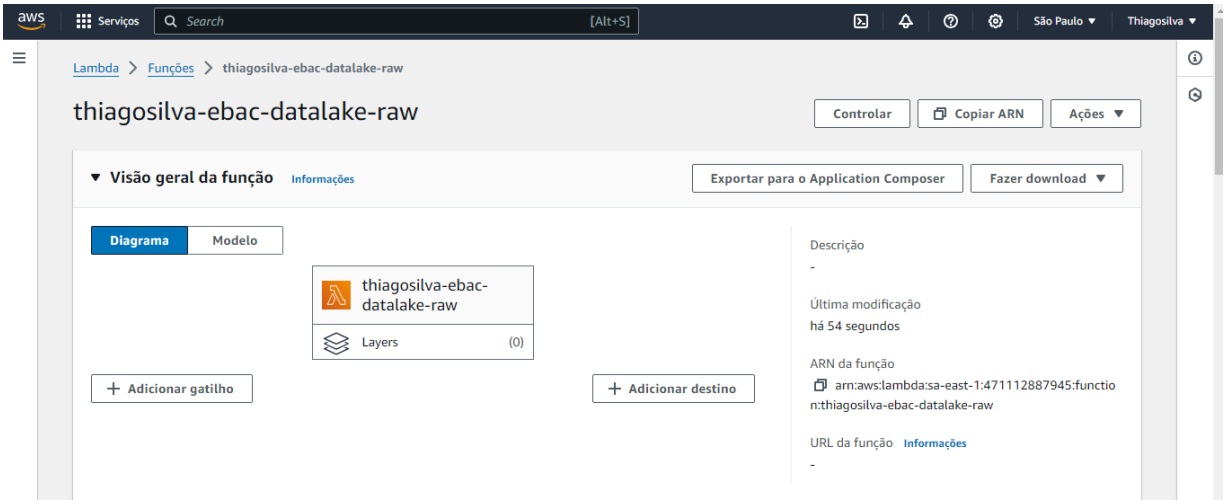
Recebe a mensagem no parâmetro event; Verifica se a mensagem tem origem no grupo do Telegram correto; Persiste a mensagem no formato JSON no bucket do AWS S3; Retorna uma mensagem de sucesso (código de retorno HTTP igual a 200) a API de bots do Telegram.

Criei a função Lambda com o mesmo nome do bucket no S3

In [ ]:
```
!wget -O '/content/lambda - raw.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/lambda - raw.png')
cv2_imshow(imagem)
```



Inseri o seguinte código na função:

In [ ]:
```
pip install boto3
```

```
Collecting boto3
  Downloading boto3-1.34.91-py3-none-any.whl (139 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.3/139.3 kB 3.1 MB/s eta 0:
00:00
Collecting botocore<1.35.0,>=1.34.91 (from boto3)
  Downloading botocore-1.34.91-py3-none-any.whl (12.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.2/12.2 MB 23.4 MB/s eta 0:0
0:00
Collecting jmespath<2.0.0,>=0.7.1 (from boto3)
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3)
  Downloading s3transfer-0.10.1-py3-none-any.whl (82 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.2/82.2 kB 7.7 MB/s eta 0:0
0:00
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/li
b/python3.10/dist-packages (from botocore<1.35.0,>=1.34.91->boto3) (2.8.2)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in /usr/local/lib/
python3.10/dist-packages (from botocore<1.35.0,>=1.34.91->boto3) (2.0.7)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pa
ckages (from python-dateutil<3.0.0,>=2.1->botocore<1.35.0,>=1.34.91->boto3)
(1.16.0)
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.34.91 botocore-1.34.91 jmespath-1.0.1 s3trans
fer-0.10.1
```

In [ ]:
```
#pacotes e bibliotecas

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


import boto3
```

```
def lambda_handler(event: dict, context: dict) -> dict:
    '''
    Recebe uma mensagem 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.
    '''

     #variáveis de ambiente

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

     #variáveis 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'

    # código principal

    client = boto3.client('s3')

    try:
        messages = json.loads(event["body"])  # A mensagem é uma lista de ob
        for message in messages:
            chat_id = message["message"]["chat"]["id"]

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

                client.upload_file(f'/tmp/{filename}', BUCKET, f'telegram/cc

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

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

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.

In [ ]:
```
!wget -O '/content/Variavel.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/Variavel.png')
cv2_imshow(imagem)
```

AWS API Gateway

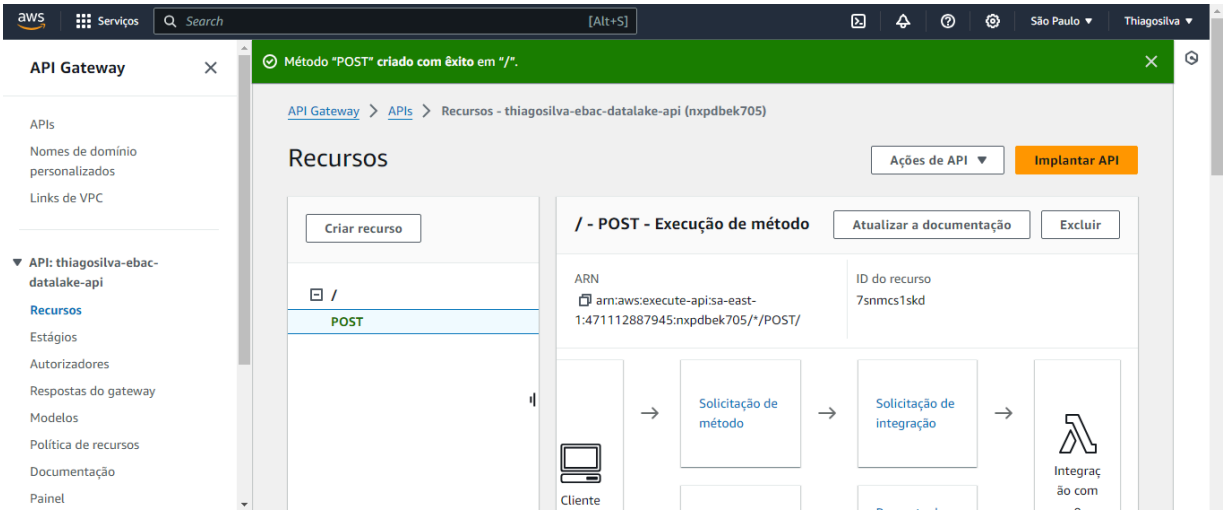
Na etapa de ingestão, o AWS API Gateway tem a função de receber as mensagens captadas pelo bot do Telegram, enviadas via webhook, e iniciar uma função do AWS Lambda, passando o conteúdo da mensagem no seu parâmetro event. Para tanto vamos criar uma API e configurá-la como gatilho da função do AWS Lambda:

Acessei o serviço e selecionei: Create API -> REST API; Inseri um nome, como padrão, um que terminou com o sufixo -api; Selecionei: Actions -> Create Method -> POST; Na tela de setup: Selecionei Integration type igual a Lambda Function; Habilitei o Use Lambda Proxy integration; Busquei pelo nome a função do AWS Lambda.

In [ ]:
```
!wget -O '/content/api.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/api.png')
cv2_imshow(imagem)
```



Inserindo url gerada na variável aws_api_gateway_url.

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

.....

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 mensagens, no formato
  JSON, do dia anterior, armazenadas no bucket de dados cru, em um único
  arquivo no formato PARQUET, armazenando-o no bucket de dados enriquecidos
  '''

  # variáveis de ambiente

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

  # variáveis 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

      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') a

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

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

        if table:

          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"tel

      return True

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

In [ ]:

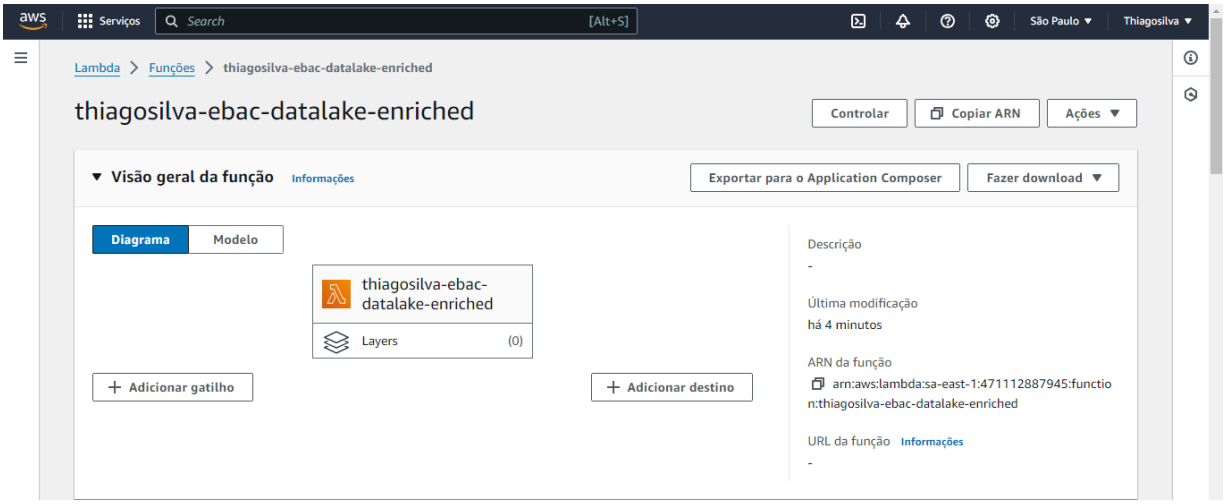
```
def parse_data(data: dict) -> dict:

  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}"] = [v]

      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}"] = [v]

      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
```

In [ ]:
```
!wget -O '/content/enriched.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/enriched.png')
cv2_imshow(imagem)
```



Apresentação

AWS Athena

Na etapa de apresentação, o AWS Athena tem função de entregar o dados através de uma interface SQL para os usuários do sistema analítico. Para criar a interface, basta criar uma tabela externa sobre o dado armazenado na camada mais refinada da arquitetura, a camada enriquecida.

In [ ]:
```
!wget -O '/content/CREATE TELEGRAM.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/CREATE TELEGRAM.png')
cv2_imshow(imagem)
```

```
CREATE EXTERNAL TABLE `telegram`(
  `message_id` bigint,
  `user_id` bigint,
  `user_is_bot` boolean,
  `user_first_name` string,
  `chat_id` bigint,
  `chat_type` string,
  `text` string,
  `date` bigint)
PARTITIONED BY (
  `context_date` date)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://<bucket-enriquecido>/'
```

Análise de Dados (SQL)

Com o dado disponível, usuário podem executar as mais variadas consultas analíticas. Seguem alguns exemplos:

In [ ]:
```
!wget -O '/content/PARTE 01.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/PARTE 01.png')
cv2_imshow(imagem)
```

- Quantidade de mensagens por dia.

```
SELECT
  context_date,
  count(1) AS "message_amount"
FROM "telegram"
GROUP BY context_date
ORDER BY context_date DESC
```

- Quantidade de mensagens por usuário por dia.

```
SELECT
  user_id,
  user_first_name,
  context_date,
  count(1) AS "message_amount"
FROM "telegram"
GROUP BY
  user_id,
  user_first_name,
  context_date
ORDER BY context_date DESC
```

In [ ]:
```
!wget -O '/content/PARTE 02.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/PARTE 02.png')
cv2_imshow(imagem)
```

- Média do tamanho das mensagens por usuário por dia.

```
SELECT
  user_id,
  user_first_name,
  context_date,
  CAST(AVG(length(text)) AS INT) AS "average_message_length"
FROM "telegram"
GROUP BY
  user_id,
  user_first_name,
  context_date
ORDER BY context_date DESC
```

In [ ]:
```
!wget -O '/content/PARTE 03.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/PARTE 03.png')
cv2_imshow(imagem)
```

- Quantidade de mensagens por hora por dia da semana por número da semana.

```
WITH
parsed_date_cte AS (
    SELECT
        *,
        CAST(date_format(from_unixtime("date"),'%Y-%m-%d %H:%i:%s') AS timestamp) AS parsed_date
    FROM "telegram"
),
hour_week_cte AS (
    SELECT
        *,
        EXTRACT(hour FROM parsed_date) AS parsed_date_hour,
        EXTRACT(dow FROM parsed_date) AS parsed_date_weekday,
        EXTRACT(week FROM parsed_date) AS parsed_date_weeknum
    FROM parsed_date_cte
)
SELECT
    parsed_date_hour,
    parsed_date_weekday,
    parsed_date_weeknum,
    count(1) AS "message_amount"
FROM hour_week_cte
GROUP BY
    parsed_date_hour,
    parsed_date_weekday,
    parsed_date_weeknum
```

In [ ]:
```
!wget -O '/content/ANALISE.png'
```

```
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

In [ ]:
```
imagem = cv2.imread('/content/ANALISE.png')
cv2_imshow(imagem)
```

parsed_date_hour	parsed_date_weekday	parsed_date_weeknum	message_amount
2	1	23	2
18	2	22	3
12	2	23	1
19	2	24	1
18	3	22	2
11	3	22	4
18	4	22	1
17	4	22	1
21	4	23	2
17	5	22	1
23	7	23	1
14	7	23	1

Conclusão

A convergência entre chatbots e análise de dados apresenta benefícios significativos para as empresas. Enquanto os chatbots simplificam a interação com os clientes, a análise dos dados resultantes proporciona insights cruciais para decisões estratégicas. Essa sinergia não apenas aprimora a experiência do cliente, mas também otimiza os processos internos e impulsiona o crescimento empresarial em diversos setores. Portanto, a integração de uma Pipeline de Dados com o Bot do Telegram emerge como uma ferramenta excepcionalmente poderosa, repleta de oportunidades para empresas que buscam se destacar em um mercado competitivo.