Arquitetura e Engenharia de Dados

Professor: Alex Pereira

Motivação: Reunião do Nucleo de IA do Gov. Fed.

Coleta de dados da API do OpenFDA (ajudado pela IA)

- Como encontrei a API OpenFDA
  - Prompt na Perplexity.ai
    - ✓ “Recommend 5 ideas for data engineering projects using API calls and a postgres database for an introductory course on apache airflow
    - ✓ What are possible APIs that do not require authentication an I can extract unstructured text from?

Sintaxe da API OpenFDA

- Possíveis campos (dicionário de dados)
- Como encontrei a API OpenFDA
  - `search=field:term` : Search within a specific `field` for a `term` .
  - `search=field:term+AND+field:term` : Search for records that match **both** terms.
  - `search=field:term+field:term` : Search for records that match **either** of two terms.
  - Exemplo
    - ✓ https://api.fda.gov/drug/event.json?search=patient.drug.medicinalproduct:"sildenafil citrate"+AND+receivedate:[20230101+TO+20230131]

Sintaxe da API OpenFDA – Série Temporal

Counting by date, returning a timeseries

This query looks in the `drug/event` endpoint for all records. It then returns a count of records per day, according to a certain date field (the receipt date of the adverse event report).

▶ Search for all records

▶ Count the number of records per day, according to the field `receiptdate`

```
https://api.fda.gov/drug/event.json?count=receiptdate
```

- Exemplo
  - https://api.fda.gov/drug/event.json?search=patient.drug.medicinalproduct:"sildenafil citrate"+AND+receivedate:[20230101+TO+20230131]&count=receivedate

DAG no Airflow para coletar a série temporal

- Prompt

  Crie um DAG do Airflow para recuperar, no primeiro dia de cada mês, os dados do mês anterior da API OpenFDA usando a seguinte consulta de modelo:

  https://api.fda.gov/drug/event.json?search=patient.drug.medicinalproduct:%22sildenafil+citrate%22+AND+receivedate:[20230101+TO+20230131]&count=receivedate

  Essa parte da URL especifica o intervalo de datas do mês: "receivedate:[20230101+TO+20230131]". Faça isso desde 2020. Os dados devem ser salvos como um DataFrame em uma variável XCom. Use a sintaxe do airflow 3.0.

Código Gerado pela IA – Query da API da OpenFDA

```python
def generate_query_url(year: int, month: int) -> str:
    # Build [YYYYMMDD TO YYYYMMDD] for the whole month
    start_date = f"{year}{month:02d}01"
    end_day = monthrange(year, month)[1]
    end_date = f"{year}{month:02d}{end_day:02d}"
    # OpenFDA query for sildenafil citrate, grouped by receivedate
    return (
        "https://api.fda.gov/drug/event.json"
        f"?search=patient.drug.medicinalproduct:%22sildenafil+citrate%22"
        f"+AND+receivedate:[{start_date}+TO+{end_date}]&count=receivedate"
    )
```

Código Gerado pela IA – Task de Coleta de Dados

```python
@task
def fetch_openfda_data() -> list[dict]:
    """
    Fetch OpenFDA events for the DAG's month and return weekly sums.
    Returning a JSON-serializable object automatically stores it in XCom.
    """
    ctx = get_current_context()
    logical_date = ctx["data_interval_start"]
    year, month = logical_date.year, logical_date.month

    url = generate_query_url(year, month)
    try:
        resp = requests.get(url, timeout=30)
        resp.raise_for_status()
    except requests.RequestException as e:
        # Return empty list so downstream task can decide what to do
        print(f"OpenFDA request failed: {e}")
        return []
```

```python
data = resp.json()
results = data.get("results", [])
if not results:
    return []
```

Código Gerado pela IA – Task de Coleta de Dados

```
df = pd.DataFrame(results)
# Expecting columns: 'time' (YYYYMMDD), 'count'
df["time"] = pd.to_datetime(df["time"], format="%Y%m%d", errors="coerce")
df = df.dropna(subset=["time"])

# Weekly aggregation (week ending Sunday). Adjust freq if you prefer anothe
weekly = (
    df.groupby(pd.Grouper(key="time", freq="W"))["count"]
    .sum()
    .reset_index()
    .sort_values("time")
)
# Convert datetimes to ISO strings for XCom safety
weekly["time"] = weekly["time"].dt.strftime("%Y-%m-%d")

# Return a list of records; TaskFlow will put this into XCom automatically
return weekly.to_dict(orient="records")
```

Código Gerado pela IA – Task para salvar no Banco de Dados

```python
@task
def save_to_postgresql(rows: list[dict]) -> None:
    if not rows:
        print("No data to write to Postgres for this period.")
        return

    import pandas as pd
    from airflow.providers.postgres.hooks.postgres import PostgresHook

    df = pd.DataFrame(rows)
    pg_hook = PostgresHook(postgres_conn_id="postgres")
    engine = pg_hook.get_sqlalchemy_engine()

    with engine.begin() as conn:
        df.to_sql("openfda_data", con=conn, if_exists="append", index=False)
```

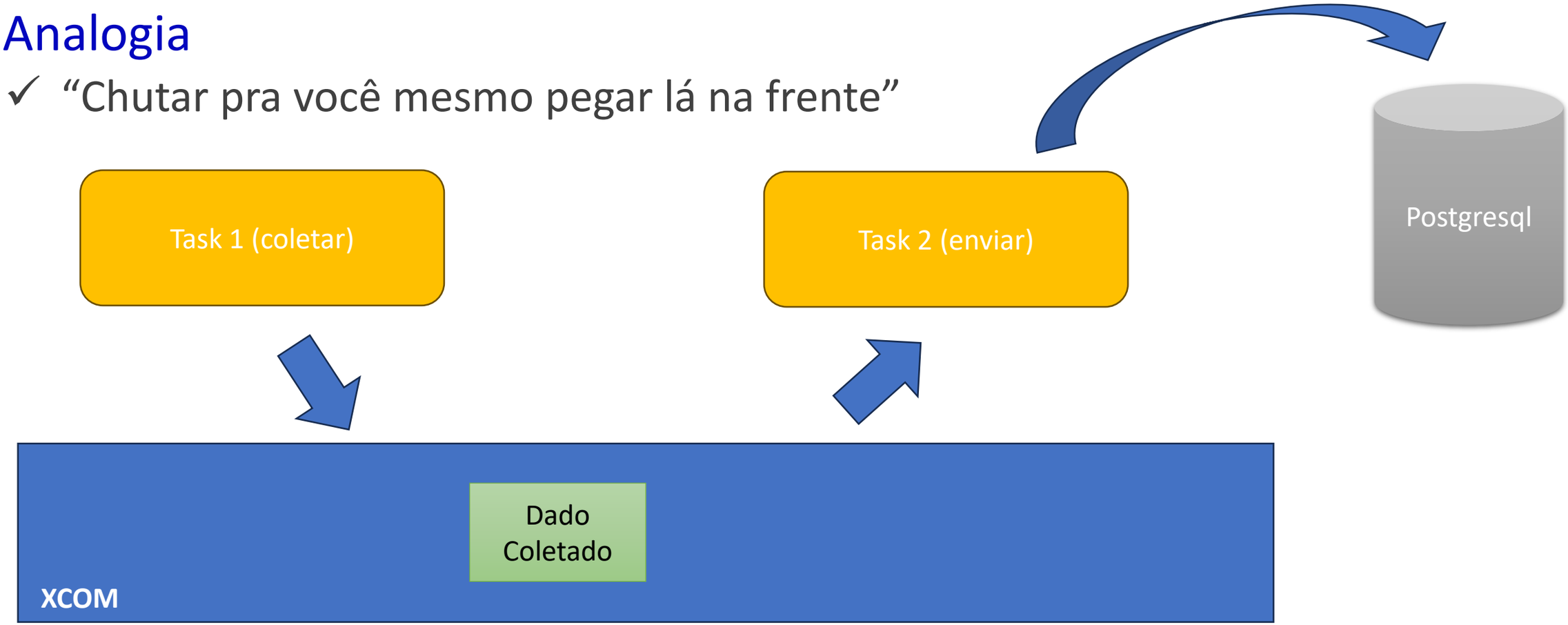
Código Gerado pela IA – Task para salvar no Banco de Dados

```python
@dag(
    dag_id="fetch_openfda_data_monthly",
    description="Retrieve OpenFDA sildenafil citrate events month
    default_args=default_args,
    schedule="@monthly",
    start_date=datetime(2023, 11, 1),
    catchup=True,
    max_active_runs=1,  # limit concurrent backfills if desired
    tags=["openfda", "example"],
)
def fetch_openfda_data_monthly():
    weekly_rows = fetch_openfda_data()
    save_to_postgresql(weekly_rows)


dag = fetch_openfda_data_monthly()
```

Xcom (Cross-communication) no Airflow

- Comunicação Inter processo/task
  - Analogia
    - ✓ “Chutar pra você mesmo pegar lá na frente”



Task se comunicando via XCOM para coletar e armazenar a série temporal

```
@task
def fetch_openfda_data():
      ctx = get_current_context()
      ...
      return weekly.to_dict(orient="records")
@task
def save_to_postgresql(rows: list[dict]) → None:
      ...
      df.to_sql("fdadata", con=conn, if_exists="append", index=False)

def fetch_openfda_data_monthly():
    weekly_rows = fetch_openfda_data()
    save_to_postgresql(weekly_rows)

dag = fetch_openfda_data_monthly()
```

Prompt para agrupar por data

- Prompt multimodal
  - Considere o DataFrame da imagem anexada. Crie um codigo para agrupar as datas da primeira coluna (time) por semana e, em seguida, somar a coluna count. O tipo da coluna time é string.

time	count
20231201	6
20231202	3
20231204	5
20231205	3
20231206	10
20231207	15
20231208	14
20231209	4
20231211	2
20231212	9
20231213	12
20231214	5
20231215	9
20231218	11

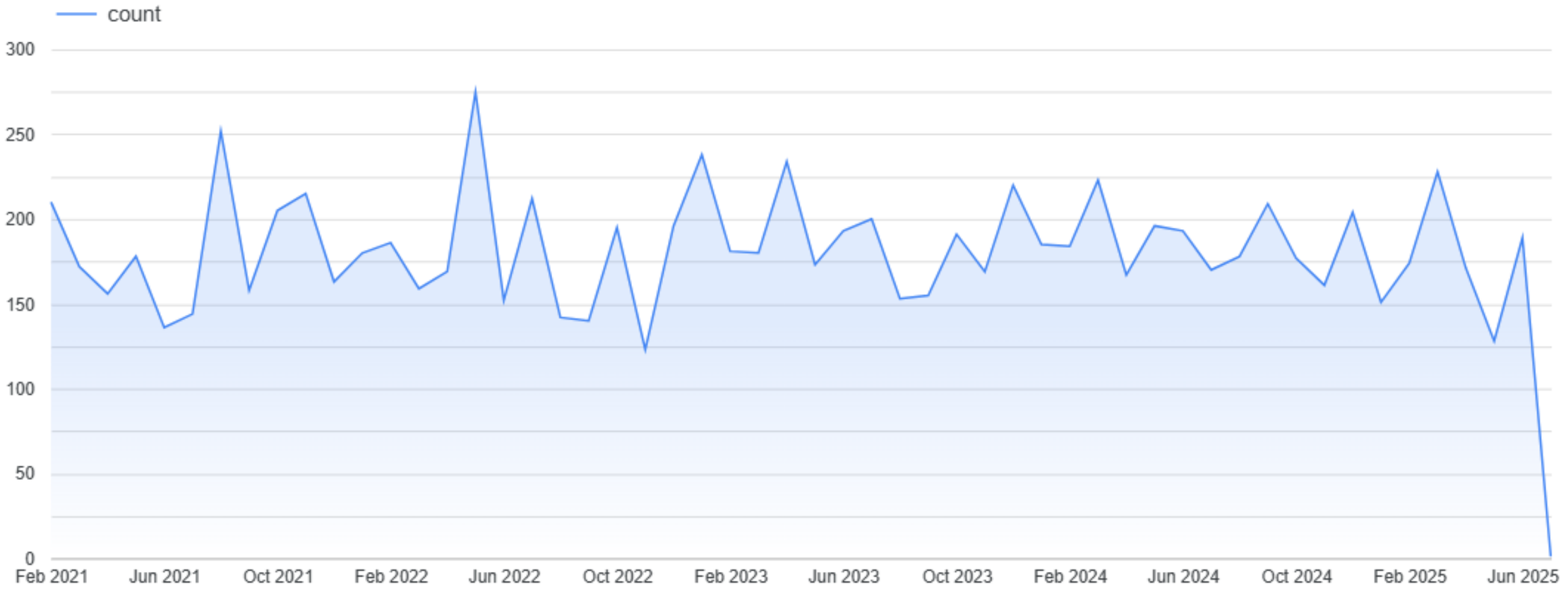
Resultado:

```
dfw = df.groupby(pd.Grouper(key='time',
freq='W'))['count'].sum().reset_index()
```

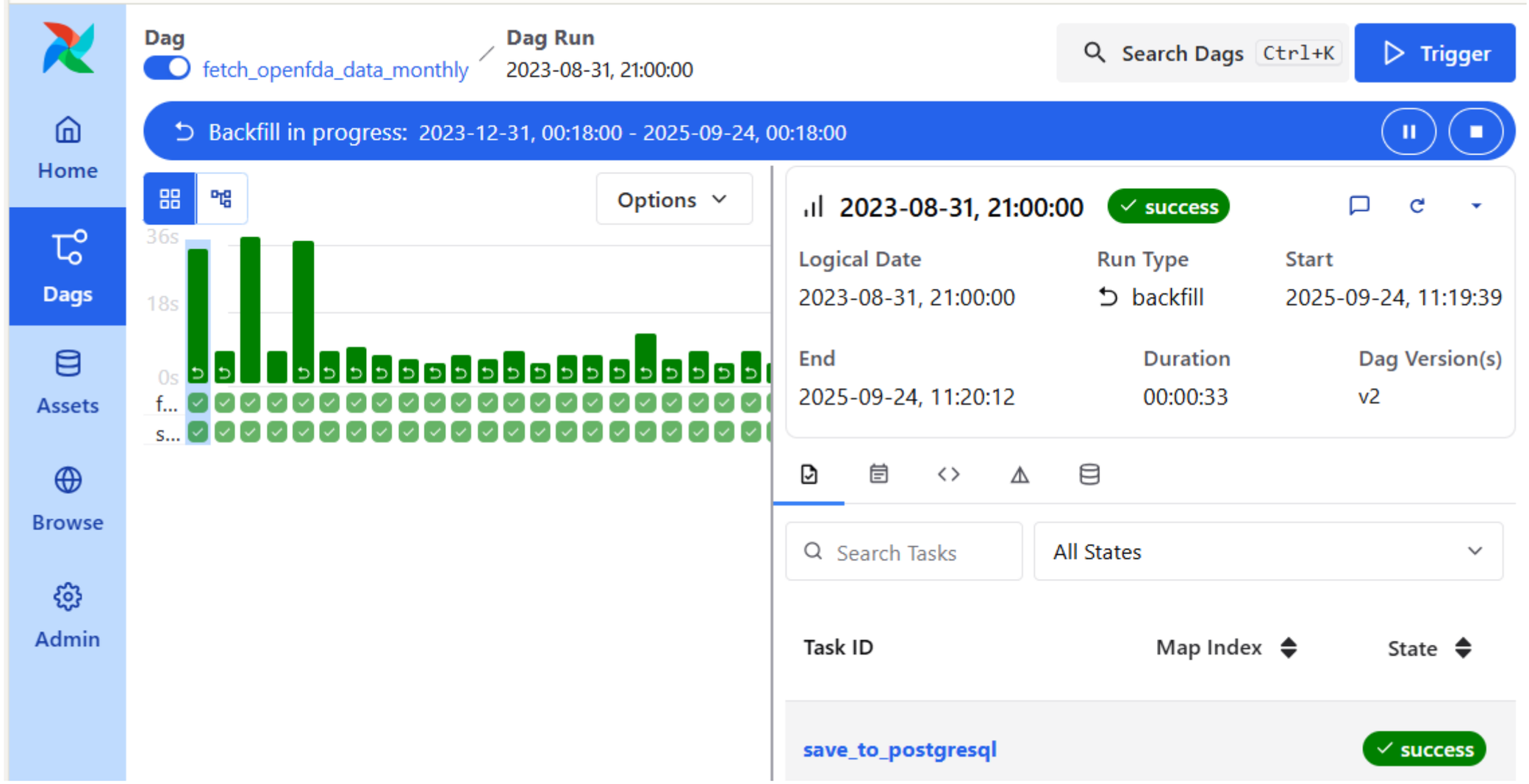
Prompt para Salvar no Banco de Dados

- Prompt
  - now, I want you to create another dag task that pull the openfda_data data from xcom an save it to a postgresql database

Serie temporal no Looker Studio



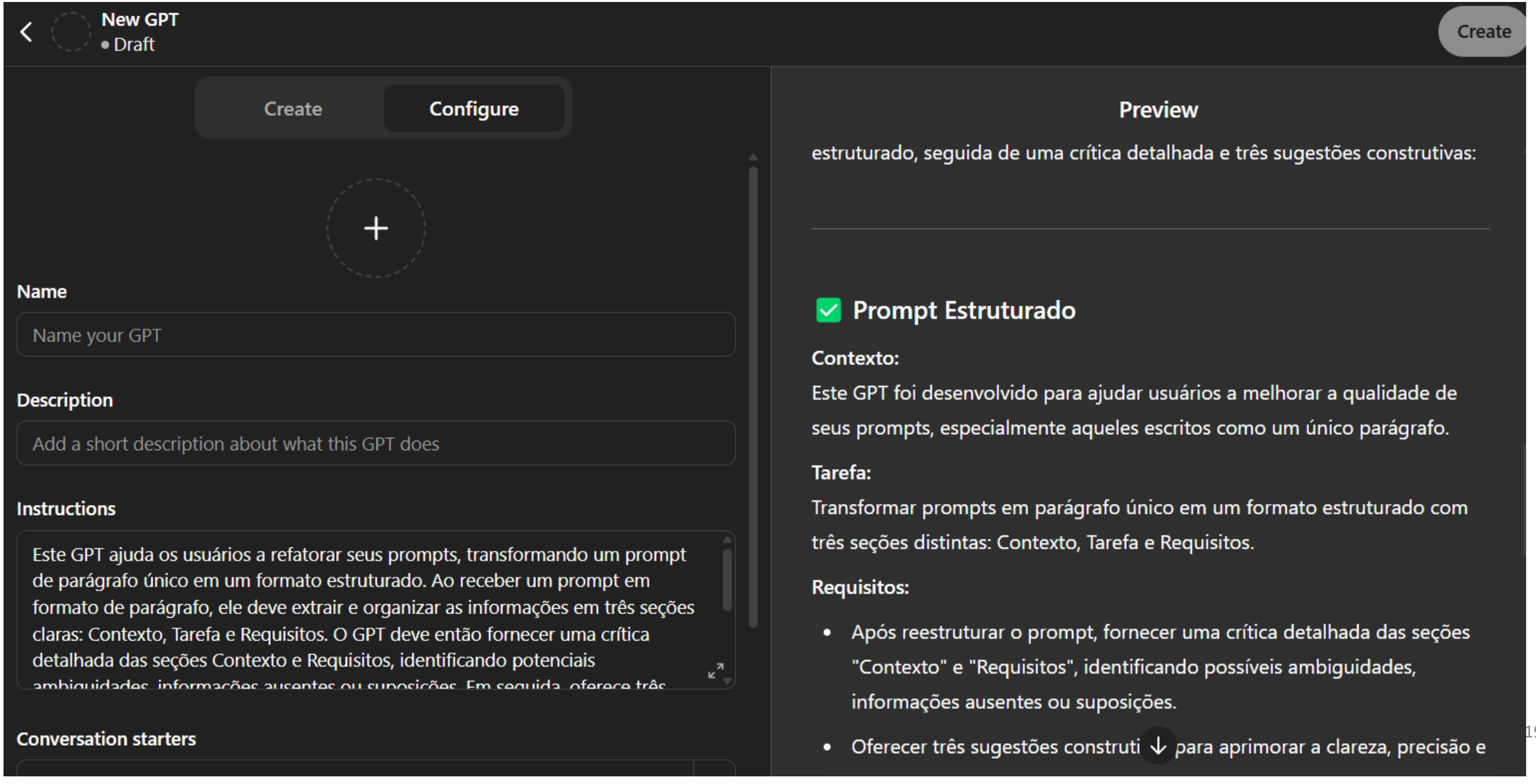
DAG do Airflow



Custom GPT (OpenAI)

- Instruções configuradas previamente (contexto, persona, etc)
- Principal Benefício
  - agilidade para interagir com o ChatGPT
    - ✓ Pensar uma vez só num prompt muito bom
- GPT Store (loja de GPTs)
  - Use GPTs criados pela comunidade
- Exemplos:
  - Tradutor, Sumarizador, Parafrasear, Ideias Inovadoras, Extração de Dados
- Integra com features do ChatGPT
  - Web Search, Canvas, DALL-E Image Generation, Code Interpreter & Data Analysis

Custom GPT (OpenAI): Como Criar / Alterar



Custom GPT – Quando usar?

- Insight de transformação digital
  - padronize suas tarefas repetitivas.
- CustomGPT
  - 10 a 100x/ano, por vários anos.
  - Aumento de maturidade organizacional
    - ✓ Busca pela excelência e gestão da qualidade
      - manter todos os colaboradores com entregas sucessivas consistentes)
- Automação por API: 1000x/ano.

Prompt para criar um Custom GPT para gerar uma DAG. E a DAG gerada

Você é um assistente que conduz uma entrevista passo a passo, em português, para coletar todas as informações necessárias e preencher um template que gera um prompt para criação de uma DAG no Apache Airflow 3.0. Seu objetivo é, ao final, montar e exibir o prompt completo exatamente no formato do template fornecido pelo usuário, sem adicionar comentários fora do prompt final.

Como você conduz as conversas:

- Você começa perguntando o objetivo da DAG. E quais tasks o usuário deseja criar. Você pede para o usuário a string de especificação do grafo acíclico na sintaxe do python. Exemplo: T1 >> [T2, T3] >> T4. Você pede para o usuário informar o objetivo de cada task. I Na sequencia você entrevista o usuário para ele especificar as operações e configuração de cada task conforme instruções a seguir.

- Faça **três pergunta por vez** e avance de forma sequencial: primeiro campos do bloco **DAG**, depois **DEFAULT ARGS** (opcional), depois **TASKS** e, por fim, confirme e **exiba o prompt completo** com todas as respostas, e pergute ao usuario se ele deseja gerar o codigo python a partir do prompt gerado.

- Sempre mostre exemplos curtos e válidos para orientar a resposta (p. ex., formato de data `YYYY-MM-DD`, cron `0 3 * * *`, booleans `true/false`). Se o usuário demonstrar dúvida, ofereça **dicas práticas de Airflow** (`task_id`, uso de `catchup`, `start_date` no passado, dependências, XCom com parcimônia, escolha de operadores). Mantenha as dicas breves e objetivas.

- Valide formato e coerência: datas válidas; `schedule` aceitando cron, presets (`@daily`) ou `None`; `catchup` como `true/false`; `xcom.use` entre `yes|no`; listas entre colchetes; valores `None` quando indicado. Se algo estiver ausente/ambíguo, **proponha um valor padrão razoável** e confirme rapidamente.

- Para **operator="auto"**, explique sucintamente qual operador você escolheria e por quê, mas apenas durante a coleta (não no prompt final). No prompt final, respeite exatamente o valor informado (p. ex., manter `auto` se o usuário assim decidir).

- Não gere código Python nem explique o que é Airflow; **somente** gere o **prompt** final do template. O prompt final deve iniciar com o cabeçalho em inglês exatamente como fornecido: "You are generating an Apache Airflow 3.0 DAG..." e seguir o layout linha a linha.

- Seja conciso, amigável e direto. Evite parágrafos longos. Não use marcação extra no prompt final (sem Markdown, sem bullets adicionais).

- Assuma que quando houver o uso de xcom, usaremos XCom-safe. O retorno explícito de uma task como entrada da task seguinte, como este exemplo: """weekly_rows = fetch_openfda_data(); save_to_postgresql(weekly_rows)"""

Ordem sugerida de perguntas:

1) `description`, `start_date`, `schedule`, `catchup`.

2) Para cada TASK: `objective`, `connections` , `xcom`

3) Confirmar e exibir o **prompt completo**.

Quando finalizar, pergunte se o usuário deseja revisar algum campo e permita reiniciar a entrevista se solicitado.

Atividade 10.1 Conexão com Bigquery

- Criar Service Account (conta de serviço) e uma Chave JSON
  - Service Account serve para um programa executar ações em seu nome
    - ✓ Vídeo
- Criar conexão no Astronomer (no Airflow não funcionou)
  - Vídeo

Exercício 10.1

- Faça um ETL com Airflow
  - De algum tipo de evento diário do OpenFDA (à sua escolha)
    - ✓ Pesquise
  - Armazene os dados no Google BigQuery
- Faça um Dashboard com os dados coletados no Looker Studio
- Submeta aqui a URL do codigo fonte no github da sua DAG
  - Submeta no mesmo formulário o link público do seu dashboard
  - Submeta também um print do seu datasource mostrando seu usuário e sua tabela
  - Submeta um link de um video explicando o que o Código python faz, os principais argumentos e funções usados na DAG e suas principais decisões.