

EXERCÍCIOS CONTAINER + AIRFLOW

- 1) Você foi contratado para preparar um ambiente de extração de dados públicos. Seu objetivo é configurar um container Docker que:

Execute um script Python que consome uma API pública (por exemplo, de clima ou criptomoedas).

Salve os dados extraídos em um arquivo .csv.

Mostre na tela os primeiros registros extraídos.

API Exemplo:

API de preços de criptomoedas (CoinGecko):

https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd

- 2) Você foi contratado para automatizar a extração de dados de criptomoedas. Seu papel é criar uma DAG no Airflow que:

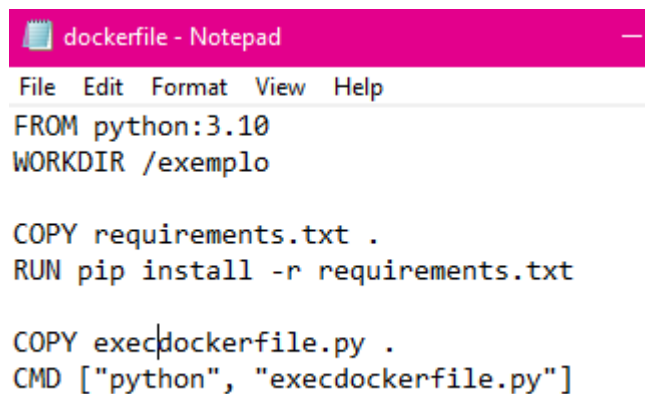
Consuma uma API pública (CoinGecko).

Salve os dados em formato .csv.

A DAG deve rodar automaticamente todos os dias às 9h da manhã.

RESOLUÇÃO EXERCÍCIO 1

1. crie uma pasta nova no ambiente documents do seu computador (a minha se chama "exemplo")
2. crie o dockerfile: abra o notepad e escreva



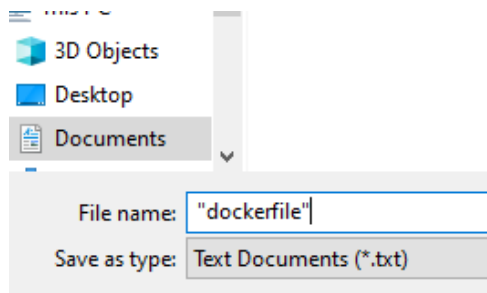
```
File Edit Format View Help
FROM python:3.10
WORKDIR /exemplo

COPY requirements.txt .
RUN pip install -r requirements.txt

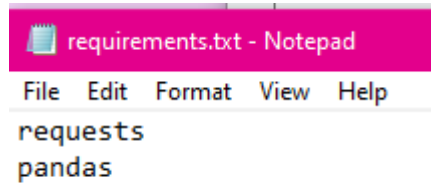
COPY execdockerfile.py .
CMD ["python", "execdockerfile.py"]
```

Neste caso, esse arquivo roda os requerimentos, e roda o python; escreva os nomes dos arquivos mesmo ainda nem tendo criado eles;

3. Salve o arquivo dockerfile SEM EXTENSÃO (use as aspas):

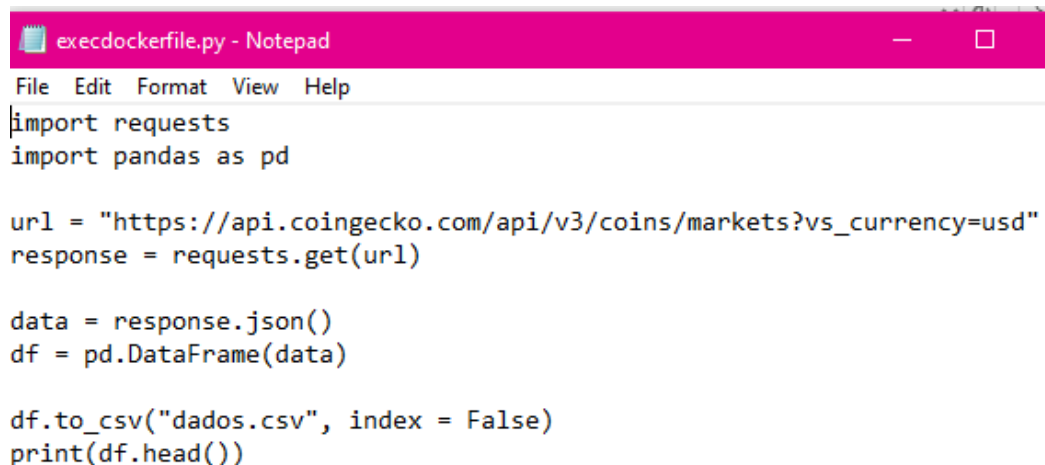


4. Crie o arquivo .txt de requerimentos no notepad:



salve com o nome IDÊNTICO ao nome escrito no dockerfile (requirements.txt)

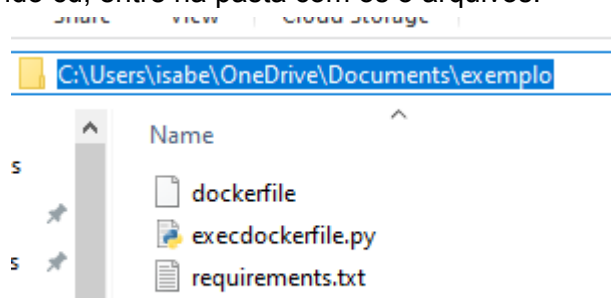
5. crie o arquivo .py no notepad:



Neste caso, ele importa as bibliotecas (já especificadas e 'automaticamente' instaladas no requerimento), depois, o exercício passado em aula pedia para lermos esse link... foi o que fizemos; neste código ele retorna um dataframe (tabela);

PS: esse arquivo deve ser salvo como nome.py (execdockerfile.py)

6. Agora, abra o docker desktop COMO ADMINISTRADOR;
7. Abra o CMD e rode o comando docker, para "ativar" ele
8. Utilizando cd, entre na pasta com os 3 arquivos:



```
C:\Users\isabe>cd OneDrive\Documents\exemplo
```

9. Depois rode o código para a criação de uma imagem com o nome que quiser (a minha: isabella)

```
C:\Users\isabe\OneDrive\Documents\exemplo>docker build -t isabella .
[+] Building 140.5s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from dockerfile                 0.0s
=> => transferring dockerfile: 202B                                0.0s
```

Images [Give feedback](#)

View and manage your local and Docker Hub images. [Learn more](#)

Local Docker Hub repositories

0 Bytes / 0 Bytes in use 1 images

Last refresh: 4 minutes ago

	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	isabella	latest	f3570f53505c	22 seconds ag	1.71 GB	▶ ⋮ 🗑

10. Quando o download terminar você roda a imagem com o seguinte código:

```
C:\Users\isabe\OneDrive\Documents\exemplo>docker run -it isabella
id symbol ... roi last_updated
0 bitcoin btc ... None 2025-04-11T18:17:47.189Z
1 ethereum eth ... {'times': 24.098853331449728, 'currency': 'btc... 2025-04-11T18:17:49.505Z
2 tether usdt ... None 2025-04-11T18:17:47.613Z
3 ripple xrp ... None 2025-04-11T18:17:43.012Z
4 binancecoin bnb ... None 2025-04-11T18:17:45.447Z
[5 rows x 26 columns]
```

note que ele retorna o head (primeiras 5 linhas) do dataframe

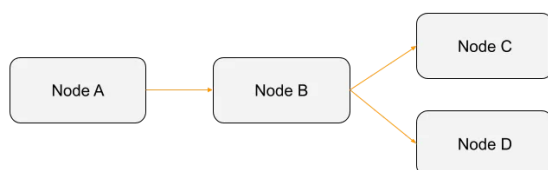
RESOLUÇÃO EXERCÍCIO 2:

sinceramente não faço a mínima ideia de como se faz isso kkkkk

INFORMAÇÕES DE PESQUISAS SOBRE O ASSUNTO:

DAG:

É um pipeline de dados definido no código Python. Cada DAG representa uma coleção de tarefas que você deseja executar sendo organizado para mostrar as relações entre as tarefas na interface gráfica do Airflow.



Um nó nada mais é que um Operator. Em outras palavras, uma Task em sua DAG é um Operator. Quando um Operator é disparado, ele se torna uma Task; cada task tem que ter seu ID

CRIAR DAG NO AIRFLOW:

Para criar uma DAG no Airflow, você sempre deve importar a classe DAG. Depois da classe DAG, vêm as importações de Operators. Basicamente, para cada Operator que deseja utilizar, deve-se fazer a importação correspondente. Finalmente, o último import geralmente é a classe de data e hora, pois você precisa especificar uma data de início para o sua DAG.

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
```

Um objeto DAG deve ter dois parâmetros: **dag_id** (identificador exclusivo da DAG em todas as DAG's) e um **start_date** (data em que sua DAG começa a ser agendada);

Além deles também tem: **schedule_interval** (intervalo de tempo em que a DAG é acionada - definido com CRON) e o **catchup** (evite a execução automática das DAGs de forma recorrente - é uma prática recomendada definir o parâmetro catchup como False)

se start_date for no passado, você vai executar vários DAGS de uma vez simultaneamente, pra isso você usa o catchup False

MEU CÓDIGO:

ps: o código NÃO foi testado porque não consegui ligar o airflow, tava dando um problema na hora de fazer pip install, e eu não soube como ligar esse código (feito no vs code) no cmd (já que tava no slide falando para importar o airflow pelo docker)...

eu mandei o copilot revisar os erros e fui alterando, já que eu não tava conseguindo rodar

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta
import requests
import csv

# peguei do slide da aula
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2025, 4, 14),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# meu código a partir de pesquisas
def link_coingecko():
    link = 'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd'
    params = { # params são os parametros que a api vai receber, já tão prontos dentro
    desse link
```

```

        'vs_currency': 'usd',
        'order': 'market_cap_desc', # ordenação por valor de mercado de uma cripto em
circulação (current_price x total_supply)
        'per_page': 100, # 100 resultados por página
        'page': 1, # vai buscar a primeira página
        'sparkline': False # sparkline é um mini gráfico que mostra a variação de preço da
cripto nos últimos 7 dias
    }

    resposta = requests.get(link, params=params) # requisição HTTP com requests (biblioteca
que faz isso)
    dados = resposta.json() # converte a resposta em JSON
    with open('cryptodata.csv', mode='w', newline='') as file: # abre o arquivo csv no modo
de escrita (w = write)
        writer = csv.writer(file)
        writer.writerow(['id', 'symbol', 'name', 'current_price', 'market_cap',
'total_volume'])
        for moeda in dados:
            writer.writerow([moeda['id'], moeda['symbol'], moeda['name'],
moeda['current_price'], moeda['market_cap'], moeda['total_volume']])

extracao_cripto = DAG(
    'extracao_cripto',
    default_args=default_args,
    descricao='uma DAG que extrai cripto do coingecko e transforma em CSV', # não é
obrigatório ter descrição
    schedule_interval='0 9 * * *', # CRON: minuto 0, hora 9, dia *, mês *, dia da semana *
(== todo dia às 9h00)
)

pega_link_coingecko = PythonOperator(
    task_id = 'link_coingecko',
    python_callable = link_coingecko, # função a ser executada
    dag=extracao_cripto, # dag pra qual essa task pertence
)

pega_link_coingecko

```