

## 1. Crear el archivo .env en la raíz del proyecto:

```
6.1 Api con Python y Flask-NEO4J.sh • .env X
C: > Users > butte > AppData > Roaming > MobaXterm > slash > RemoteFiles > 5965818_5_7 > .env
1 |Neo4j Credentials:
2 |NEO4J_URI=bolt://neo4j:7687
3 |#NEO4J_URI es la variable que define la URL de conexión a la base de datos Neo4j. *bolt:// es el protocolo que se usa
4 |#para establecer una conexión con Neo4j. bolt es un protocolo binario optimizado para la comunicación con bases de datos Neo4j. *neo4j es
5 |#el nombre del host donde se está ejecutando Neo4j. En un contexto de contenedores Docker, como en tu caso, NEO4J puede ser el NOMBRE DEL
6 |#SERVICIO de Neo4j dentro de una red de contenedores, lo que significa que Docker se encarga de resolver este nombre.
7 |NEO4J_USER=neo4j
8 |#NEO4J_USER es el nombre de usuario utilizado para autenticarte en la base de datos Neo4j. *neo4j es el nombre de usuario predeterminado
9 |#de Neo4j. Si no has cambiado la configuración predeterminada, este será el usuario para acceder a la base de datos.
10|NEO4J_PASSWORD=Dianey94*
11|
12|#Backend Configuration
13|FLASK_PORT=5010
```

## 2. Crear una carpeta con nombre backend en el proyecto, y dentro crear el archivo Dockerfile:

```
6.1 Api con Python y Flask-NEO4J.sh • Dockerfile X
C: > Users > butte > AppData > Roaming > MobaXterm > slash > RemoteFiles > 5965818_5_24 > Dockerfile > ...
1 |1. Usar una imagen base ligera de Python
2 |FROM python:3.9-alpine
3 |
4 |#2. Establecer el directorio de trabajo dentro del contenedor.Crea una carpeta de nombre "app" en la raíz del contenedor.
5 |WORKDIR /app
6 |
7 |#3. Copiar el archivo requirements.txt que contiene las dependencias de Python
8 |COPY requirements.txt .
9 |
10|#Instala las dependencias
11|RUN pip install --no-cache-dir -r requirements.txt
12|
13|#4. Copiar el resto del código fuente de la MV al CONTENEDOR, específicamente en el directorio de trabajo (en este caso "app")..
14|COPY . .
15|
16|#5. Exponer el puerto en el que correr la app
17|EXPOSE 5010
18|
19|#6. Comando para ejecutar la aplicación:
20|# *CMD: Esta es una instrucción en el Dockerfile que define el comando que debe ejecutarse cuando el contenedor se inicia.
21|# Solo se puede tener una instrucción CMD en un Dockerfile; si se define más de una, solo se ejecutará la última.
22|# *["python", "app.py"]: Este es el comando que se ejecutará cuando el contenedor se inicie. Especifica que se debe usar el
23|# ejecutable python para correr el archivo app.py en el contenedor. Aquí, se está ejecutando la aplicación Flask con python
24|# app.py.
25|CMD ["python", "app.py"]
```

## 3. Dentro de la carpeta backend crear el archivo app.py con lo básico para correr y conectar con neo4j:

```

$ 6.1 Api con Python y Flask-NEO4j.sh ● app.py 1 X appbase.py 4 ●
C: > Users > butte > AppData > Roaming > MobaXterm > slash > RemoteFiles > 5965818_5_30 > app.py > ...
1  from flask import Flask
2  import os
3  from dotenv import load_dotenv
4  from neo4j import GraphDatabase
5  import json
6
7  # Cargar las variables de entorno desde el archivo .env
8  load_dotenv()
9
10 # Creamos una instancia de la clase Flask.
11 # Esta instancia es nuestra aplicación.
12 app = Flask(__name__)
13
14 # Configuración de Neo4j usando las variables de entorno
15 NEO4J_URI = os.getenv("NEO4J_URI")
16 NEO4J_USER = os.getenv("NEO4J_USER")
17 NEO4J_PASSWORD = os.getenv("NEO4J_PASSWORD")
18
19 # Conexión con la base de datos Neo4j
20 driver = GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWORD))
21
22 # Rutas de la aplicación
23 @app.route('/')
24 def hello():
25     # Esta función retorna un mensaje simple de texto cuando se accede a la ruta "/"
26     return "¡Hola desde un contenedor Docker con Python y Flask!"
27
28 # Verificamos si el archivo está siendo ejecutado como el script principal.
29 # Si es así, iniciamos la aplicación Flask.
30 if __name__ == '__main__':
31     # Usar el puerto desde las variables de entorno o por defecto 5010
32     flask_port = os.getenv("FLASK_PORT", 5010)
33     app.run(host="0.0.0.0", port=int(flask_port))

```

#### 4. Crear el docker-compose.yml con los servicios neo4j y ETL;

```

C: > Users > butte > AppData > Roaming > MobaXterm > slash > RemoteFiles > 5965818_5_11 > docker-compose.yml
D Run All Services
1  services:
D Run Service
2    neo4j:
3      image: neo4j:latest # Usamos una imagen slim de neo4j
4      container_name: neo4j_db_docker_python10
5      environment:
6        NEO4J_AUTH: ${NEO4J_USER}/${NEO4J_PASSWORD}
7      volumes:
8        - neo4j_data_python10:/data #neo4j_data_python10: Este es el nombre de un volumen de Docker. * /data: Esta es la ruta dentro del
9          #contenedor de Neo4j donde los datos de la base de datos serán almacenados.
10     ports:
11       - "6990:7474" # Exponemos el puerto 7474 para la interfaz web de Neo4j.
12       - "4009:7687" # Exponemos el puerto 7687 para el protocolo Bolt de Neo4j. El protocolo Bolt es un protocolo de comunicación binario
13         # y eficiente utilizado por Neo4j para la interacción con su base de datos de grafos.
14     networks:
15       - app_network_python10
16
D Run Service
17  ETL:
18    image: mi-app-python10-web:v1 # Aquí se define el nombre de la imagen del servicio web.
19    build: ./backend #Se le indica a Docker que debe construir la imagen del contenedor usando un Dockerfile ubicado en el directorio ind
20    container_name: dianey-mi-app-python-docker10
21    restart: always
22    depends_on:
23      - neo4j
24    environment: #Define las variables de entorno para la aplicación
25      - FLASK_ENV=development
26    env_file:
27      - .env # Cargar todas las variables de entorno desde el archivo .env
28    networks:
29      - app_network_python10
30    ports:

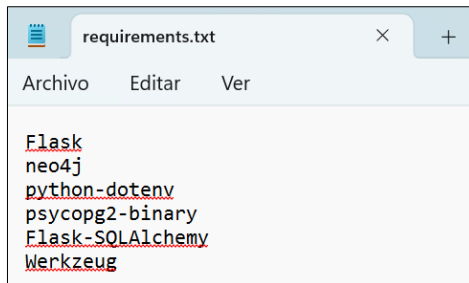
```

```

30     ports:
31         - "${FLASK_PORT}:5010"
32     volumes:
33         - ./backend/students.json:/app/students.json
34
35     volumes:
36         neo4j_data_python10:
37
38     networks:
39         app_network_python10:
40             driver: bridge
41

```

5. Creamos el archivo requirements.txt dentro de la carpeta backend:



```

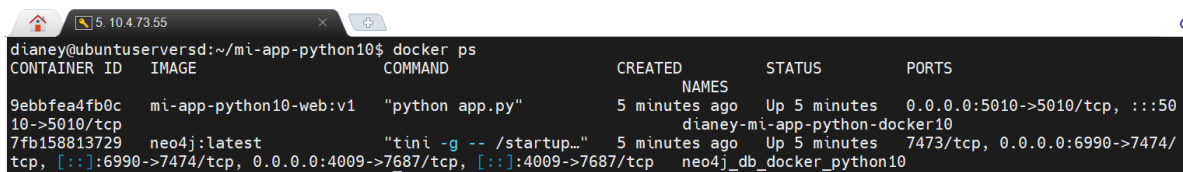
Flask
neo4j
python-dotenv
psycopg2-binary
Flask-SQLAlchemy
Werkzeug

```

6. Creamos un archivo students.json solo para que no salga errores por ahora.

7. Ejecutar el docker-compose.yml y revisar que todo vaya bien:

```
docker-compose --env-file .env up --build -d
```



```

dianey@ubuntu:~/mi-app-python10$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
9ebbfea4fb0c   mi-app-python10-web:v1              "python app.py"         5 minutes ago Up 5 minutes  0.0.0.0:5010->5010/tcp, :::5010->5010/tcp
7fb158813729   neo4j:latest                        "tini -g -- /startup..." 5 minutes ago Up 5 minutes  7473/tcp, 0.0.0.0:6990->7474/tcp, [::]:6990->7474/tcp, 0.0.0.0:4009->7687/tcp, [::]:4009->7687/tcp
neo4j_db_docker_python10

```

8. Añadir el servicio PostgreSQL, ejecutar el docker-compose.yml y revisar que todo vaya bien:

```

PostgreSQL:
  image: postgres:13-alpine # Usamos una imagen oficial de PostgreSQL
  container_name: postgres_db_docker_python10
  environment:
    POSTGRES_USER: postgres # Usuario de PostgreSQL
    POSTGRES_PASSWORD: postgres # Contraseña de PostgreSQL
    POSTGRES_DB: etl_db # Base de datos que se creará al iniciar
  volumes:
    - postgres_data:/var/lib/postgresql/data #Volumen para persistir los datos de la base de datos. *postgres_data: Es el nombre del volumen. * /var/lib/postgresql/data: Es la ubicación interna dentro del contenedor de PostgreSQL donde se almacenan los datos de la base de datos.
  ports:
    - "5432:5432" # Exponemos el puerto 5432 para conectarnos a PostgreSQL
  networks:
    - app_network_python10

volumes:
  neo4j_data_python10:
  postgres_data:

```

```
docker-compose --env-file .env up --build -d
```

```
dianey@ubuntusersvd:~/mi-app-python10$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
f8b7ecc20098   mi-app-python10-web:v1              "python app.py"         18 seconds ago Up 14 seconds 0.0.0.0:5010->5010/tcp, :::5010->5010/tcp
d1a072d45c7e   neo4j:latest                        "tini -g -- /startup..." 20 seconds ago Up 17 seconds 7473/tcp, 0.0.0.0:6990->7473/tcp, [::]:6990->7474/tcp, 0.0.0.0:4009->7687/tcp, [::]:4009->7687/tcp
ae163e0ac4a6   postgres:13-alpine                 "docker-entrypoint.s..." 20 seconds ago Up 17 seconds 0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
dianey@ubuntusersvd:~/mi-app-python10$
```

9. Colocar el archivo dataset\_b\_lenguajes.csv en la carpeta backend.

10. Crear un Bind Mount dentro de volúmenes del servicio neo4j:

- ./backend/dataset\_b\_lenguajes.csv:/import/dataset\_b\_lenguajes.csv

```
services:
  neo4j:
    image: neo4j:latest # Usamos una imagen slim de neo4j
    container_name: neo4j_db_docker_python10
    environment:
      NEO4J_AUTH: ${NEO4J_USER}/${NEO4J_PASSWORD}
    volumes:
      - neo4j_data_python10:/data #neo4j_data_python10: Este es el nombre de un volumen de Docker. * /data: Esta es la ruta dentro del contenedor de Neo4j donde los datos de la base de datos serán almacenados.
      - ./backend/dataset_b_lenguajes.csv:/import/dataset_b_lenguajes.csv #Bind Mount.* ./ indica la ruta actual en el host. La ruta /import es especial dentro del contenedor de Neo4j que está preconfigurada para cargar archivos, especialmente cuando usas el comando LOAD CSV # para cargar datos desde archivos CSV en la DB.
    ports:
      - "6990:7474" # Exponemos el puerto 7474 para la interfaz web de Neo4j.
      - "4009:7687" # Exponemos el puerto 7687 para el protocolo Bolt de Neo4j. El protocolo Bolt es un protocolo de comunicación binario y eficiente utilizado por Neo4j para la interacción con su base de datos de grafos.
    networks:
      - app_network_python10
```

11. Ejecutar el docker-compose.yml y revisar que todo vaya bien:

docker-compose --env-file .env up --build -d

```
dianey@ubuntusersvd:~/mi-app-python10$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
38a5b181de51   mi-app-python10-web:v1              "python app.py"         14 seconds ago Up 12 seconds 0.0.0.0:5010->5010/tcp, :::5010->5010/tcp
f77c433fccc1   postgres:13-alpine                 "docker-entrypoint.s..." 14 seconds ago Up 13 seconds 0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
02d0fa27335a   neo4j:latest                        "tini -g -- /startup..." 14 seconds ago Up 13 seconds 7473/tcp, 0.0.0.0:6990->7474/tcp, [::]:6990->7474/tcp, 0.0.0.0:4009->7687/tcp, [::]:4009->7687/tcp
dianey@ubuntusersvd:~/mi-app-python10$
```

12. Ejecutar: