# **Introduce Docker**

1) Create DOCKERFILE

In the first line: We load the already existing image of python with tag <3.9> from Docker Hub.

Then we run all other commands we wish.

In ENTRYPOINT we say that: When we do docker run, we want docker to do whatever is inside there, i.e., python, and

pipeline.py

And that is how we take the default image and based on that we create a new one.

2) Create a **data pipeline** pipeline.py

Write your code and do whatever you want.

3) Create Docker IMAGE and run: **docker build -t vision .**

Build the docker file to create the image: vision is the name of the docker image.

"Vision is the name, dashboards are the game!" - Efsta

4)\* Run the image: docker run -it vision *(This step is not required for the project – just for you to see what is happening if you perform this step manually)*

After vision add parameters if needed in the code.

# **Ingest Data to Postgres**

5) **Run Postgres in Docker =** Create **docker-compose.yaml** (see details below for this file)

This yaml file will set-up the connection and the container of the Postgres database.

We specify the name of the image for postgres, we created manually a folder in our local system that is for the dataset. We called it *ny\_taxi\_postgres\_data* andwe will map this to a folder in Postgres (mounting).

Later that we will run the docker image for postgres (in step 6), we will see that postgres will create and install some stuff there. Maybe it is some internal representation for the database.

6)\* Run the docker image for postgres (the name of the image is postgres:13). The “*volumes*” is a way of mapping the folder that we have in the host machine to the folder in the container. Postgres is a database and needs to keep files in a file system. However, Docker does not keep the state so we want to define this volume to remember where is the data. *(This step is not required for the project – just for you to see what is happening if you perform this step manually)*

docker run -it \  
 -e POSTGRES\_USER="root" \  
 -e POSTGRES\_PASSWORD="root" \  
 -e POSTGRES\_DB="ny\_taxi" \  
 -v $(pwd)/ny\_taxi\_postgres\_data:/var/lib/postgresql/data \  
 -p 5432:5432 \  
 postgres:13

7)\* Access this database manually from your local machine just to check the connection to the Postgres is ok. *(This step is not required for the project – just for you to see what is happening if you perform this step manually)*

Run a **cli client** for accessing the database. We use **pgcli** which is a library in python. So, in another terminal, we must pip3 install pgcli , if we don’t have pgcli already installed.

Run:

python3.10 -m pip install --upgrade pip

pip3 install pgcli

pip3 install "psycopg[binary,pool]"

**pgcli -h localhost -p 5432 -u root -d ny\_taxi**

If connection is successful: select \* from information\_schema.schemata

Run: \dt 🡪 to see the list of all the tables

8) We wget the dataset locally and we explore it a bit. To see how many samples, we have in the dataset run this: **wc -l yellow\_tripdata\_2021-01.csv**

9) Install **sqlalchemy** and connect to Postgres to **send the dataset in Postgres in batches**.

First, we need **to generate the compatible DDL schema for Postgres**, which means that we convert this dataframe to DDL: pd.io.sql.get\_schema(df, name=”yellow\_taxi\_data”)

First install:

pip install sqlalchemy

pip install psycopg2-binary

**create\_engine()** from sqlalchemy: we need to create a connection to postgres and

**pd.io.sql.get\_schema()**: we will generate the statement of the schema that is specific for Postgres.

We create the table, and we **send the data in chunks using an iterator**. Once this is done, if we go back to step 7 (that can be omitted) we can check manually what has happened in the Postgres.

Run this: SELECT count(1) FROM yellow\_taxi\_data;

See that data has been sent to Postgres.

# **Connect pgAdmin and Postgres**

10) **Install pgAdmin using Docker**: is a GUI tool to interact with the Postgres database.

So, before we used **pgcli**, but it is not convenient to use pgcli for data exploration and querying.

🡪**pgAdmin** is more convenient. Since, we have docker, we don’t need to install it, but we can just pull an image from Docker Hub (or create one by ourselves) that contains the tool.

Run pgAdmin in Docker:

docker run -it \

-e PGADMIN\_DEFAULT\_EMAIL="admin@admin.com" \

-e PGADMIN\_DEFAULT\_PASSWORD="root" \

-p 8080:80 \

dpage/pgadmin4

We map our port 8080 which is on our machine, to a port 80 on the container.

**pgAdmin** is running, is listening to requests on port 80 and we map this port to our host machine port 8080. Hence, all the requests we will send to port 8080, they will be sent to port 80 on the container.

Open localhost:8080 from your browser, and see you now have pgAdmin. Login with the credentials just above.

11) Create new Server in pgAdmin:

We need this server to communicate with Postgres. In Register – Server tab, we give the information for the Postgres database (see Jupyter notebook file: localhost, root, root). But we get a connection error because Postgres is in another container from pgAdmin.

Then, we must place them in the same network to avoid these connection errors.

So, terminate the 2 containers (the one we run the Postgres, and the one we run the pgAdmin).

12) **Docker Networks: Running Postgres and pgAdmin together in one network**

Run: docker network create pg-network

So, we have a network, and then we need to run postgres container in this network.

So, RE-run the updated **Postgres command**:

**docker run -it \**

**-e POSTGRES\_USER="root" \**

**-e POSTGRES\_PASSWORD="root" \**

**-e POSTGRES\_DB="ny\_taxi" \**

**-v $(pwd)/ny\_taxi\_postgres\_data:/var/lib/postgresql/data \**

**-p 5432:5432 \**

**--network=pg-network \**

**--name pg-database \**

**postgres:13**

*(Note: the name will be how pgAdmin will be able to discover Postgres)*

Run again (on another command line) the **pgcli -h localhost -p 5432 -u root -d ny\_taxi**

And **SELECT count(1) FROM yellow\_taxi\_data;**

Check you still have all your data there.

RE-run the updated **pgAdmin command in the same network (on another cmd line):**

**docker run -it \**

**-e PGADMIN\_DEFAULT\_EMAIL="admin@admin.com" \**

**-e PGADMIN\_DEFAULT\_PASSWORD="root" \**

**-p 8080:80 \**

**--network=pg-network \**

**--name pgadmin \**

**dpage/pgadmin4**

Reload pgAdmin page and try to create the Server now. In Register tab, we give the name that we specified in the network attribute above, i.e., pg-database, and then root, root.

Once you connect, you see the 2 databases. Open ny\_taxi, then Schemas, then Tables. Right click, view data, and see the first 100 rows. It runs this query, and you cannot edit it. But open the Query Tool from Tools and run your own queries.

Graphical user interface, text, application, email

Description automatically generated

13) Todo

14) Todo

15) Todo

16) Todo

17) Todo

18) Todo

19) Todo

20) Todo

21) Todo

22) Todo

23) Todo

24) Todo