ETL for PostgreSQL

Using Python, Docker & Docker Compose

Stamatis Sideris
DATA MANAGEMENT & ANALYTICS CONSULTANT

Contents

Introduction

This report aims to show how to create a Docker Network including a PostgreSQL DB and a python ETL script. Two ways are introduced, one using the basic Docker format and one using Docker Compose Tool.

What is Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The docker-compose CLI utility allows users to run commands on multiple containers at once, for example, building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container. The docker-compose.yaml file is used to define an application's services and includes various configuration options. For example, the build option defines configuration options such as the Dockerfile path, the command option allows one to override default Docker commands, and more.

Anaconda Installation

We install Anaconda as it includes Python 3.9 which we need to run in Terminal.

Visit the link and choose the installer that fits your OS. I choose the Linux Installer.

Download it with the wget command in your terminal.

```
stamatis@ecommerce:~$ wget https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86_64.sh
```

Install it with the bash command and choose to initialize it. Restart your terminal and Anaconda should be ready to run!

```
stamatis@ecommerce:~$ bash Anaconda3-2022.10-Linux-x86_64.sh
```

```
(base) stamatis@ecommerce:~$ conda --version
conda 22.9.0
```

Docker Compose Installation

Firstly, create a bin directory to store your downloads. Then, visit the following link and choose the version of docker-compose you prefer for downloading. We download the docker-compose-linux-x86_64 version as our subsystem works in Ubuntu.

```
stamatis@ecommerce:~$ mkdir bin
stamatis@ecommerce:~$ cd bin
stamatis@ecommerce:~/bin$ wget https://github.com/docker/compose/releases/download/v2.16.0/docker-compose-linux-x86_64 -O docker_compose
```

You will observe that the system does not recognize docker_compose as an executable and so we use the following command to do so.

```
stamatis@ecommerce:~/bin$ chmod +x docker_compose
stamatis@ecommerce:~/bin$ ls
docker_compose
```

Finally, in order to make the executable visible from all directories, we visit the bashrc file using nano and we add the bin directory to the path using the following line of code:

export PATH=”${HOME}/bin:${PATH}”

The bashrc file is the one storing the paths that are initialized when our instance starts. We write out the file and exit. Then, we use the source command to restart the file (we could also logout and login) and we are ready to use docker-compose.

```
stamatis@ecommerce:~$ nano bashrc
```

```
 GNU nano 2.9.3                                                  .bashrc

# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export PATH="${HOME}/bin:${PATH}"
```

```
stamatis@ecommerce:~$ source .bashrc
stamatis@ecommerce:~$ docker_compose version
Docker Compose version v2.16.0
```

Containerized ETL Procedures for Local Infrastructures using Python, Docker & Docker Compose

We will use Python to create a very basic ETL flow that will extract, transform and load our data from the local data folder to the PostgreSQL database. The whole procedure will run inside a Docker network which will allow us to test our procedures without the risk of affecting the underground database we have assumed the company is running.

Firstly, we proceed by creating a local folder called “local_flow” where we will store our ETL flow. Inside the directory, we create an etl directory to store our ETL procedures. Inside the etl directory, we create a etl.py python file, which we will use to perform the ETL procedures, and a constants.py python file where we will set the parameters needed for our flow.

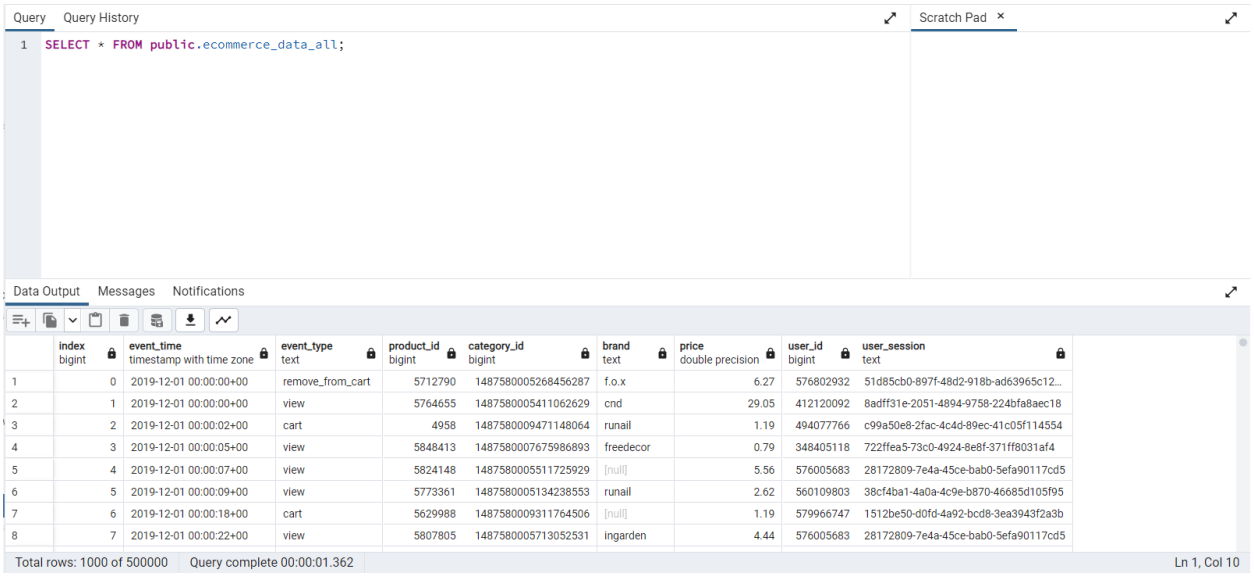
```
user = "root"
password = "root"
```

```
host = "pgdatabase"
port = "5432"
db = "ecommerce_data"
table_name = "ecommerce_data_all"
csv_name = ["2019-Dec.csv","2019-Nov.csv","2019-Oct.csv","2020-Jan.csv","2020-Feb.csv"]
data_url = "https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-cosmetics-shop/download?datasetVersionNumber=6"
data_path = "ecommerce-events-history-in-cosmetics-shop/"
```

```
# imports
import argparse
from time import time
import pandas as pd
from sqlalchemy import create_engine
from datetime import timedelta
import constants
import opendatasets as od


def extract_data(path: str) :
    # read each csv in chunks of 100000 rows
    df_iter = pd.read_csv(path, iterator=True, chunksize=100000)
    df = next(df_iter)

    return df

def transform_data(df) :
    # set the datetime format and drop a table that is full of Nulls
    df['event_time'] = pd.to_datetime(df['event_time'])
    df = df.drop('category_code', axis=1)
    return df

def load_data(table_name, df):
    # create an engine that connects to the postgres database
    engine =
create_engine(f'postgresql://{constants.user}:{constants.password}@{constants.hos
t}:{constants.port}/{constants.db}')
    # append each chunk to the table - the chunk method is needed because .to_sql
function cannot handle large volume of data
    df.to_sql(name=table_name, con=engine, if_exists='append')
    print("Finished ingesting data into the postgres database")

def log_subflow(table_name: str):
    print(f"Logging Subflow for: {table_name}")
```

```python
def main_flow():
    # download dataset from Kaggle
    od.download(constants.data_url)
    log_subflow(constants.table_name)
    for i in constants.csv_name:
        path = constants.data_path + i
        raw_data = extract_data(path)
        data = transform_data(raw_data)
        load_data(constants.table_name, data)

if __name__ == '__main__':
    main_flow()
```

As a result, our database should be loaded with our transformed data. To test it, we run a Select statement in postegres.



	index bigint	event_time timestamp with time zone	event_type text	product_id bigint	category_id bigint	brand text	price double precision	user_id bigint	user_session text
1	0	2019-12-01 00:00:00+00	remove_from_cart	5712790	1487580005268456287	f.o.x	6.27	576802932	51d85cb0-897f-48d2-918b-ad63965c12...
2	1	2019-12-01 00:00:00+00	view	5764655	1487580005411062629	cnd	29.05	412120092	8adff31e-2051-4894-9758-224bfa8aec18
3	2	2019-12-01 00:00:02+00	cart	4958	1487580009471148064	runail	1.19	494077766	c99a50e8-2fac-4c4d-89ec-41c05f114554
4	3	2019-12-01 00:00:05+00	view	5848413	1487580007675986893	freedecor	0.79	348405118	722ffea5-73c0-4924-8e8f-371ff8031af4
5	4	2019-12-01 00:00:07+00	view	5824148	1487580005511725929	[null]	5.56	576005683	28172809-7e4a-45ce-bab0-5efa90117cd5
6	5	2019-12-01 00:00:09+00	view	5773361	1487580005134238553	runail	2.62	560109803	38cf4ba1-4a0a-4c9e-b870-46685d105f95
7	6	2019-12-01 00:00:18+00	cart	5629988	1487580009311764506	[null]	1.19	579966747	1512be50-d0fd-4a92-bcd8-3ea3943f2a3b
8	7	2019-12-01 00:00:22+00	view	5807805	1487580005713052531	ingarden	4.44	576005683	28172809-7e4a-45ce-bab0-5efa90117cd5

We create a Docker image called “ecommerce_data_local_flow”, to use it as a blueprint on creating the container that will run the etl.py and constants.py .

In order to apply our flow to the containerized network, we visit our local_flow directory and create a Dockerfile. In the Dockerfile we set what we want our image to include, which is to install the libraries needed in the python scripts, the python scripts and the data to be used.

```
local_flow > Dockerfile > ...
1   FROM python:3.9.1
2
3   RUN pip install pandas sqlalchemy psycopg2
4
5   WORKDIR /app
6   COPY etl.py etl.py
7   COPY constants.py constants.py
8   COPY data.zip data.zip
9
10  ENTRYPOINT [ "python", "etl.py", "constants.py" ]
```

We also set the host variable in the constants.py file to “pg-database”, as our network is not going to run to localhost but the host we set when we created the PostgreSQL server.

```
local_flow > constants.py
1   user = "root"
2   password = "root"
3   host = "pg-database"
4   port = "5432"
5   db = "ecommerce_data"
6   table_name = "ecommerce_data_all"
7   csv_name = ["2019-Dec","2019-Nov","2019-Oct","2020-Feb","2020-Jan"]
8
```

We then proceed on creating the image.

```
(base) stamatis@ecommerce:~/local_flow$ docker build -t ecommerce_data_local_flow .
```

To test our Docker Network, we drop the “ecommerce-data” table in our postgres database and we proceed to create the container that will run our ETL procedure via Docker.

```
(base) stamatis@ecommerce:~/local_flow$ docker run -it --network=pg-network ecommerce_data_local_flow
```

Back to pgAdmin and our table must be loaded with our data!

Query | Query History | Scratch Pad

```
SELECT * FROM public.ecommerce_data_all
LIMIT 100
```

Data Output | Messages | Notifications

	index bigint	event_time timestamp with time zone	event_type text	product_id bigint	category_id bigint	brand text	price double precision	user_id bigint	user_session text
1	0	2019-12-01 00:00:00+00	remove_from_cart	5712790	1487580005268456287	f.o.x	6.27	576802932	51d85cb0-897f-48d2-918b-ad63965c12...
2	1	2019-12-01 00:00:00+00	view	5764655	1487580005411062629	cnd	29.05	412120092	8adff31e-2051-4894-9758-224bfa8aec18
3	2	2019-12-01 00:00:02+00	cart	4958	1487580009471148064	runail	1.19	494077766	c99a50e8-2fac-4c4d-89ec-41c05f114554
4	3	2019-12-01 00:00:05+00	view	5848413	1487580007675986893	freedecor	0.79	348405118	722ffea5-73c0-4924-8e8f-371ff8031af4
5	4	2019-12-01 00:00:07+00	view	5824148	1487580005511725929	[null]	5.56	576005683	28172809-7e4a-45ce-bab0-5efa90117c...
6	5	2019-12-01 00:00:09+00	view	5773361	1487580005134238553	runail	2.62	560109803	38cf4ba1-4a0a-4c9e-b870-46685d105f...
7	6	2019-12-01 00:00:18+00	cart	5629988	1487580009311764506	[null]	1.19	579966747	1512be50-d0fd-4a92-bcd8-3ea3943f2a...
8	7	2019-12-01 00:00:22+00	view	5807805	1487580005713052531	ingarden	4.44	576005683	28172809-7e4a-45ce-bab0-5efa90117c...

Total rows: 100 of 100 | Query complete 00:00:00.528 | Ln 1, Col 1

The whole procedure could become much faster and maintainable by using Docker Compose. Docker Compose will use a .yaml file, where we will configure our services, to create the network that will host our services. The file would have the following structure:

local_flow > docker-compose.yaml

```yaml
services:
  pgdatabase:
    image: postgres:13
    environment:
      - POSTGRES_USER=root
      - POSTGRES_PASSWORD=root
      - POSTGRES_DB=ecommerce_data
    volumes:
      - "./data:/var/lib/postgresql/data:rw"
    ports:
      - "5432:5432"
  pgadmin:
    image: dpage/pgadmin4
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@admin.com
      - PGADMIN_DEFAULT_PASSWORD=root
    ports:
      - "8080:80"
  local_etl_flow:
    build: .
    image: local_etl_flow
    stdin_open: true
    tty: true
    volumes:
      - .:/ecommerce_data
```

We configure three services, the pgdatabase which will use the image postgres:13 from Docker Hub to create a pgdatabase container, the pgadmin which will use the image dpage/pgadmin4 from Docker Hub to create a pgadmin container and the local_etl_flow which will search for and use our Dockerfile to create a local_etl_flow container. All the containers will run under the same network.

Before running the Docker Compose, firstly remove all the existing images from docker. We do this because images already exist from our previous manual trial without Docker Compose and they take storage space. To do so:

```
(base) stamatis@ecommerce:~/local_flow$ docker images rmi
```

Then, to run Docker Compose, use the following command:

```
(base) stamatis@ecommerce:~/local_flow$ docker_compose up -d
[+] Running 3/3
 ⠿ Container local_flow-pgdatabase-1      Started                                                    1.4s
 ⠿ Container local_flow-pgadmin-1         Running                                                    0.0s
 ⠿ Container local_flow-local_etl_flow-1  Running                                                    0.0s
```

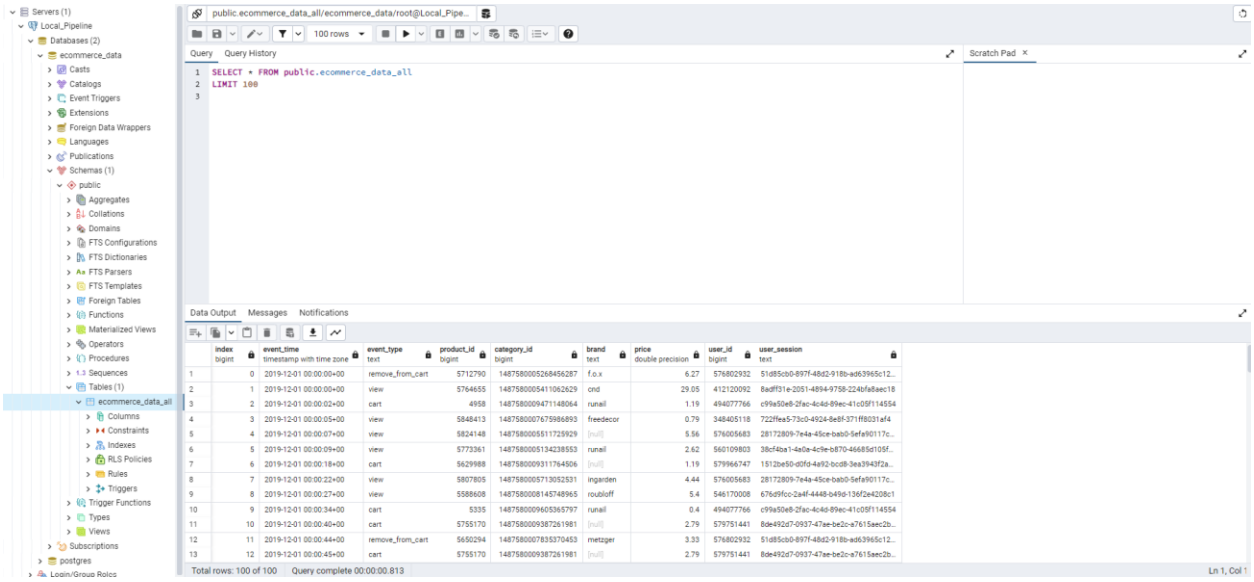
All 3 containers are running. To check them, we use the following command:

```
(base) stamatis@ecommerce:~/local_flow$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
9d5ca31e65b8   postgres:13     "docker-entrypoint.s…"   26 seconds ago   Up 23 seconds   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp              local_flow-pgdatabase-1
ed3b97e48a39   dpage/pgadmin4  "/entrypoint.sh"         26 seconds ago   Up 24 seconds   443/tcp, 0.0.0.0:8080->80/tcp, :::8080->80/tcp         local_flow-pgadmin-1
f417e1356517   local_etl_flow  "python"                 26 seconds ago   Up 24 seconds                                                          local_flow-local_etl_flow-1
```

Next step, we need to enter to our local_etl_flow container to access the directory installed in it and run our etl.py file. To do so:

```
(base) stamatis@ecommerce:~/local_flow$ docker exec -it local_flow-local_etl_flow-1 bash
root@7206beb456e6:/local_flow# ls
Dockerfile  __pycache__  docker-compose.yaml  etl
root@7206beb456e6:/local_flow# cd etl
root@7206beb456e6:/local_flow/etl# python etl.py
```

After the script is finished running, we visit again the pgadmin in localhost port 80 and register our postgres database in host pgdatabase. The database should exist and include a table “ecommerce_data_all” filled with our data.



Use the following command to close all the running containers:

```
(base) stamatis@ecommerce:~/local_flow$ docker_compose down
[+] Running 4/4
 ⠿ Container local_flow-pgadmin-1          Removed                                                                4.0s
 ⠿ Container local_flow-pgdatabase-1       Removed                                                                1.0s
 ⠿ Container local_flow-local_etl_flow-1   Removed                                                               11.6s
 ⠿ Network local_flow_default              Removed                                                                0.3s
```

References

https://docs.docker.com/get-started/08_using_compose/#:~:text=Docker%20Compose%20is%20a%20tool,or%20tear%20it%20all%20down.

https://www.anaconda.com