MLOps

1강

최민우

Database

목차

Overview

1.Docker 를 이용하여 DB server 를 생성

2.psycopg2 패키지를 이용하여 테이블 생성 및 데이터 삽입

3.Dockerfile 과 Docker Compose 파일을 생성

4.Docker 컨테이너 안에서 계속해서 데이터를 생성하는 서비스를 구축

Overview
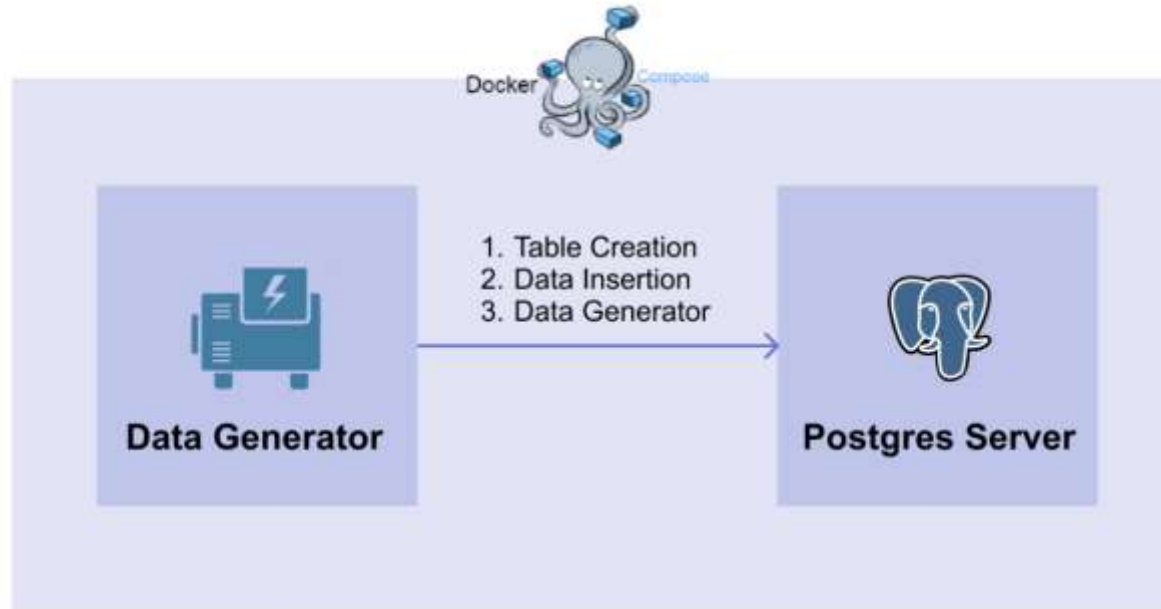
DB 서버 생성

1.docker run 명령어를 사용하면, 간단한 옵션들을 통해 DB 서버를 생성
2.docker ps 명령어를 통해, DB 서버가 잘 동작하는지 확인

```
$ docker run -d \
  --name postgres-server \
  -p 5432:5432 \
  -e POSTGRES_USER=myuser \
  -e POSTGRES_PASSWORD=mypassword \
  -e POSTGRES_DB=mydatabase \
  postgres:14.0
```
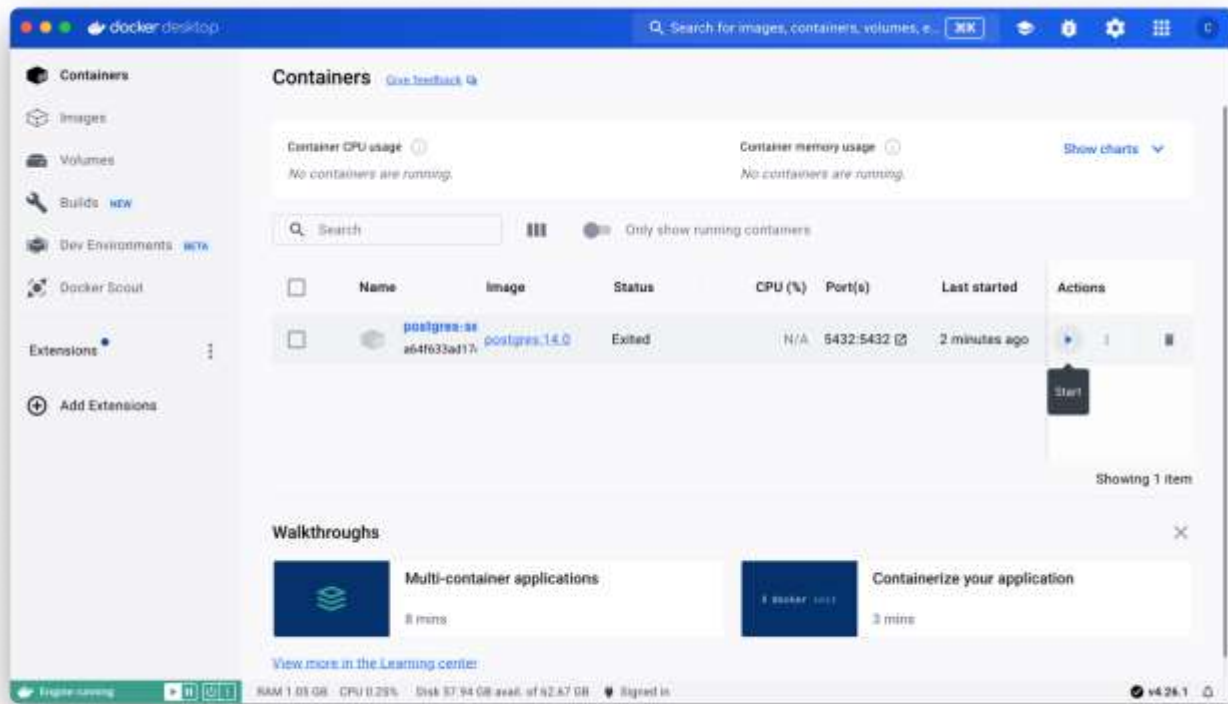
$ docker ps

DB 서버 생성

DB 서버 생성

DB 서버 생성

SQL Shell (psql)

1.psql 실행
2.psql 명령어를 통해, DB 서버가 잘 동작하는지 확인

> PGPASSWORD=mypassword psql -h localhost -p 5432 -U myuser -d mydatabase

- *PGPASSWORD=* : 접속할 유저의 비밀번호를 입력합니다.
- *-h* : 호스트를 지정합니다.
- *-p* : 포트를 지정합니다.
- *-U* : 접속할 유저의 이름을 입력합니다.
- *-d* : DB 의 이름을 입력합니다.

DB 서버 생성

1.접속 성공
2.\du 를 통해 DB 의 role name 과 attributes 을 확인

테이블 생성

1.패키지 설치: pip3 install pandas psycopg2-binary scikit-learn

2.Python을 이용하여, PostgreSQL DB 서버에 접근하는 코드를 구현(psycopg2 패키지)

3.DB Connection → Table Creation Query → Query 실행

테이블 생성
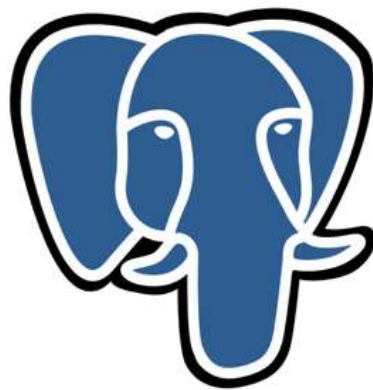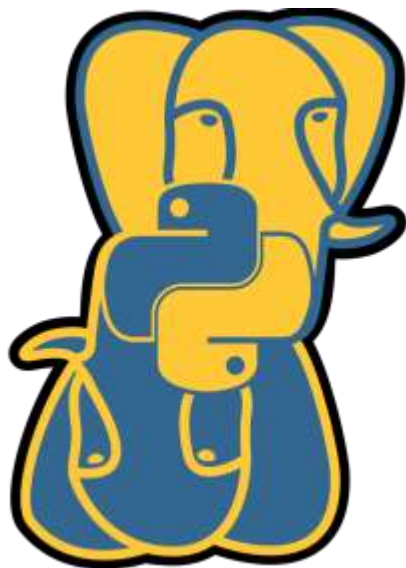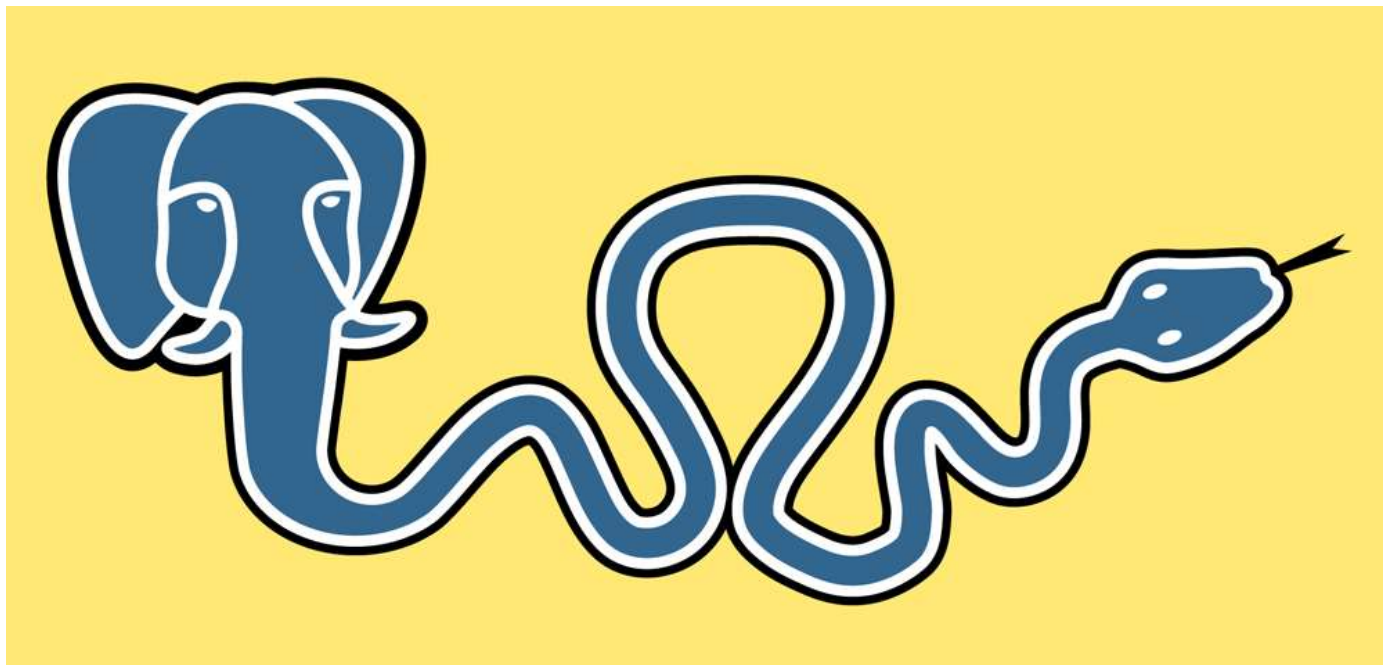
테이블 생성

테이블 생성

테이블 생성

1) DB Connection
- psycopg2 를 이용하여 DB 에 접근하기 위해서는 connect 함수를 사용
- DB 에 연결할 때 *user*, *password*, *host*, *port*, *database* 의 총 5가지 정보가 필요

```python
import psycopg2

db_connect = psycopg2.connect(
    user="myuser",
    password="mypassword",
    host="localhost",
    port=5432,
    database="mydatabase",
)
```

테이블 생성

2) Table Creation Query

- PostgreSQL에서는 float64, int 64를 지원하지 않음
- PostgreSQL에서는 float8, int를 사용

```
create_table_query = """
CREATE TABLE IF NOT EXISTS iris_data (
    id SERIAL PRIMARY KEY,
    timestamp timestamp,
    sepal_length float8,
    sepal_width float8,
    petal_length float8,
    petal_width float8,
    target int
);"""
```

테이블 생성

3) Query 실행

- Visual Studio Code 터미널에서 python table_creator.py를 입력
- 또는, 오른쪽 상단의 **실행 버튼**을 클릭

```
$ python table_creator.py

    CREATE TABLE IF NOT EXISTS iris_data (
        id SERIAL PRIMARY KEY,
        timestamp timestamp,
        sepal_length float8,
        sepal_width float8,
        petal_length float8,
        petal_width float8,
        target int
    );
```
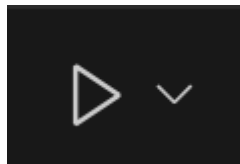
테이블 생성

1.psql 실행

2.\d를 통해 생성된 테이블들의 목록을 확인

3.iris_data 테이블의 데이터 전체를 확인: select * from iris_data;

```
mydatabase=# \d
                List of relations
[ Schema |        Name       |   Type   | Owner
--------+------------------+----------+--------
 public | iris_data        | table    | myuser
 public | iris_data_id_seq | sequence | myuser
(2 rows)
```

```
mydatabase=# select * from iris_data;
 id | timestamp | sepal_length | sepal_width | petal_length | petal_width | target
----+-----------+--------------+-------------+--------------+-------------+--------
(0 rows)
```

데이터 삽입

1.생성한 테이블에 iris 데이터 한 줄 삽입

2.iris 데이터 불러오기 → Data Insertion Query → Query 실행

데이터 삽입

1) iris 데이터 불러오기

- 생성된 테이블의 Column 이름과 일치하도록 수정

```python
import pandas as pd
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True, as_frame=True)
df = pd.concat([X, y], axis="columns")
rename_rule = {
    "sepal length (cm)": "sepal_length",
    "sepal width (cm)": "sepal_width",
    "petal length (cm)": "petal_length",
    "petal width (cm)": "petal_width",
}
df = df.rename(columns=rename_rule)
```

데이터 삽입

2) Data Insertion Query

- 삽입 순서 중요

```python
insert_row_query = f"""
INSERT INTO iris_data
    (timestamp, sepal_length, sepal_width, petal_length, petal_width, target)
    VALUES (
        NOW(),
        {data.sepal_length},
        {data.sepal_width},
        {data.petal_length},
        {data.petal_width},
        {data.target}
    );"""
```

데이터 삽입

3) Query 실행

- Visual Studio Code 터미널에서 python data_insertion.py를 입력
- 또는, 오른쪽 상단의 실행 버튼을 클릭

```
$ python data_insertion.py

    INSERT INTO iris_data
        (timestamp, sepal_length, sepal_width, petal_length, petal_width, target)
        VALUES (
            NOW(),
            6.2,
            2.9,
            4.3,
            1.3,
            1.0
        );
```

데이터 삽입

1.psql 실행

2.iris_data 테이블의 데이터 전체를 확인: select * from iris_data;

```
[mydatabase=# select * from iris_data;
 id | timestamp | sepal_length | sepal_width | petal_length | petal_width | target
----+-----------+--------------+-------------+--------------+-------------+--------
(0 rows)

[mydatabase=# select * from iris_data;
 id |          timestamp          | sepal_length | sepal_width | petal_length | petal_width | t
arget
----+-----------------------------+--------------+-------------+--------------+-------------+--
------
  1 | 2023-12-30 11:03:42.166353 |            6 |         2.7 |          5.1 |         1.6 |
    1
(1 row)
```

데이터 삽입 Loop

1.생성된 테이블 안에 데이터를 계속해서 추가

2.Loop 추가 → Query 실행

데이터 삽입 Loop

1) Loop 추가

- while True
- 너무 빠른 시간에 데이터가 추가되면, DB에 과부하가 생길 수 있음
- 데이터를 삽입 후 잠시 대기하는 시간을 추가: time.sleep(1)

```python
import time


def generate_data(db_connect, df):
    while True:
        insert_data(db_connect, df.sample(1).squeeze())
        time.sleep(1)
```

데이터 삽입 Loop

2) Query 실행

- Visual Studio Code 터미널에서 python data_insertion.py를 입력
- 또는, 오른쪽 상단의 실행 버튼을 클릭

데이터 삽입 Loop

2) Query 실행

```
$ python data_insertion_loop.py

    INSERT INTO iris_data
        (timestamp, sepal_length, sepal_width, petal_length, petal_width, target)
        VALUES (
            NOW(),
            6.5,
            2.8,
            4.6,
            1.5,
            1.0
        );


    INSERT INTO iris_data
        (timestamp, sepal_length, sepal_width, petal_length, petal_width, target)
        VALUES (
            NOW(),
            5.5,
            4.2,
            1.4,
            0.2,
            0.0
        );
```
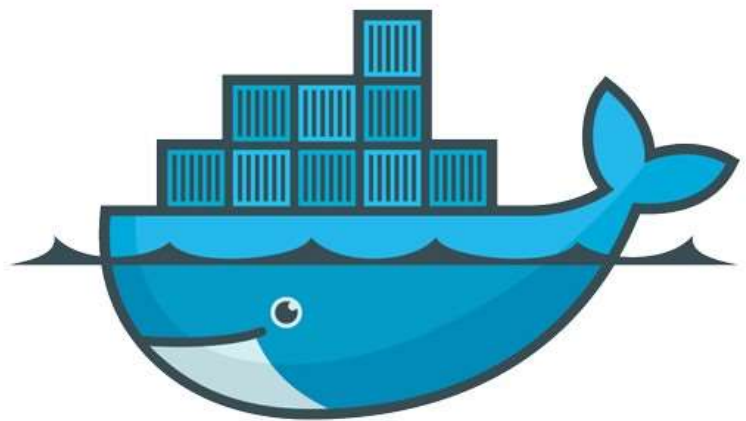
데이터 삽입 Loop

1.psql 실행

2.iris_data 테이블의 데이터 전체를 확인: select * from iris_data;

데이터 생성 on Docker

1.앞서 작성했던 코드를 Docker 컨테이너 안에서 실행: Dockerfile

2.Docker 컨테이너 간의 네트워크를 연결하여, DB에 데이터를 계속해서 삽입

3.Data Generator → Dockerfile → Docker Network

데이터 생성 on Docker

데이터 생성 on Docker

1) Data Generator

```python
parser = ArgumentParser()
parser.add_argument("--db-host", dest="db_host", type=str, default="localhost")
args = parser.parse_args()
```

데이터 생성 on Docker

2) Dockerfile

- Visual Studio Code에서 Docker 확장자를 설치

데이터 생성 on Docker

2) Dockerfile

- Dockerfile 생성

```
FROM amd64/python:3.9-slim

RUN apt-get update && apt-get install -y \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /usr/app

RUN pip install -U pip &&\
    pip install scikit-learn pandas psycopg2-binary

COPY data_generator.py data_generator.py

ENTRYPOINT ["python", "data_generator.py", "--db-host"]
# Change CMD to solve host finding error
CMD ["localhost"]
```
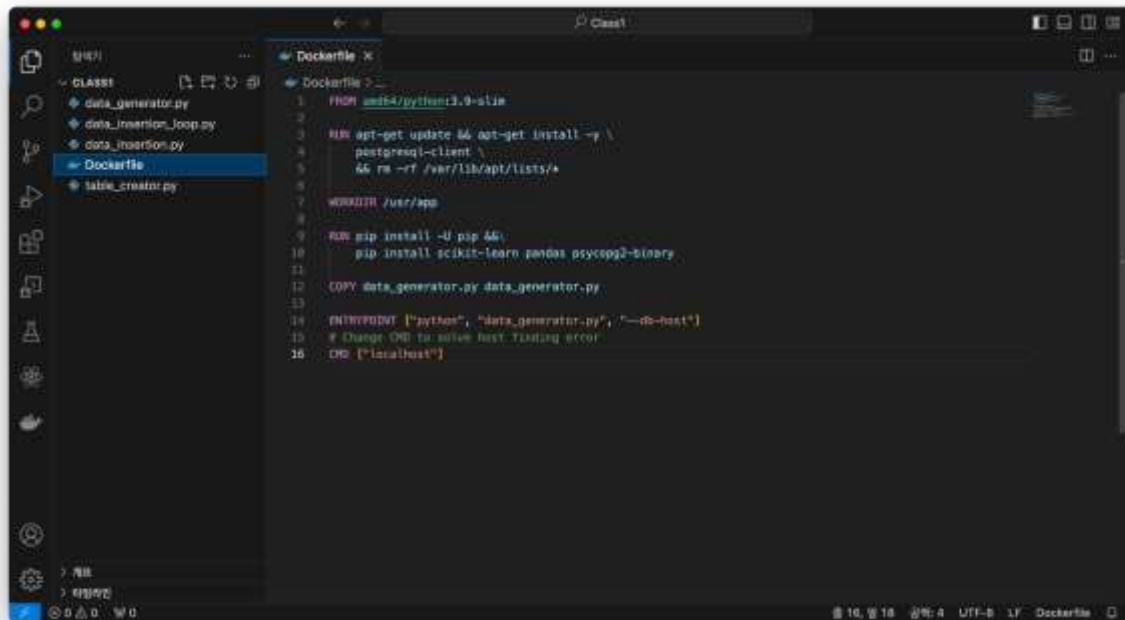
데이터 생성 on Docker

2) Dockerfile

- Dockerfile 생성

데이터 생성 on Docker

2) Dockerfile

 -   Build

```
$ docker build -t data-generator .
```

데이터 생성 on Docker

2) Dockerfile

- Build
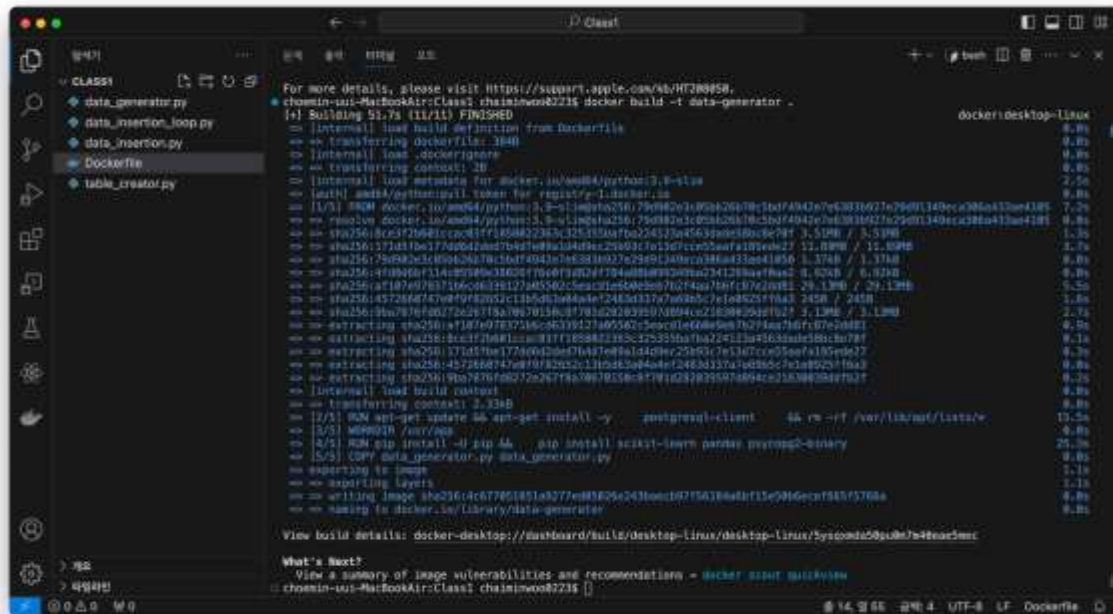
데이터 생성 on Docker

2) Dockerfile

- 이미지 생성 확인

$ docker image ls

데이터 생성 on Docker
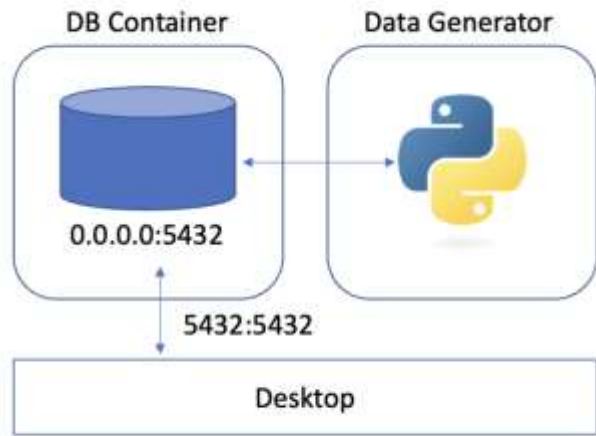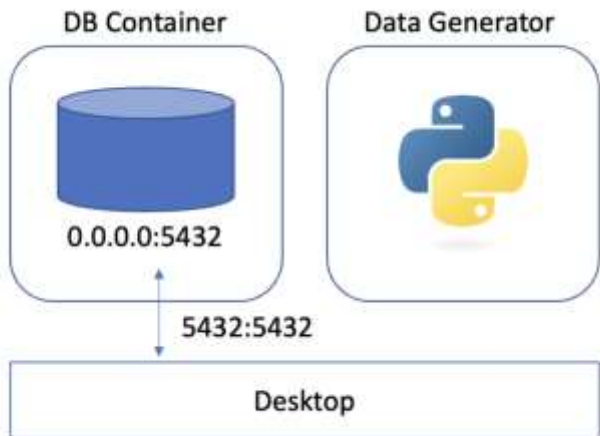
3) Docker Network

데이터 생성 on Docker

3) Docker Network

- 네트워크 생성

$ docker network create my-network

- 네트워크 연결

$ docker network connect my-network postgres-server

데이터 생성 on Docker

3) Docker Network

- data-generator 컨테이너 실행

```
$ docker run -d \
    --name data-generator \
    --network "my-network" \
    data-generator "postgres-server"
```

데이터 생성 on Docker

1.psql 실행

2.iris_data 테이블의 데이터 전체를 확인: select * from iris_data;


```
mydatabase=# select * from iris_data;
 id |        timestamp        | sepal_length | sepal_width | petal_length | petal_width | t
arget
----+-------------------------+--------------+-------------+--------------+-------------+--
=======
  1 | 2023-12-30 11:03:42.166353 |          6 |         2.7 |          5.1 |         1.6 |
    1
  2 | 2023-12-30 11:50:15.371534 |        7.2 |         3.2 |            6 |         1.8 |
    2
  3 | 2023-12-30 11:50:16.389618 |        5.5 |         3.5 |          1.3 |         0.2 |
    0
  4 | 2023-12-30 11:50:17.401693 |        5.7 |         4.4 |          1.5 |         0.4 |
    0
  5 | 2023-12-30 11:50:18.412587 |        6.9 |         3.1 |          5.1 |         2.3 |
    2
  6 | 2023-12-30 11:50:19.42266  |        5.1 |         3.8 |          1.6 |         0.2 |
    0
  7 | 2023-12-30 11:50:20.43241  |        4.3 |           3 |          1.1 |         0.1 |
    0
  8 | 2023-12-30 11:50:21.443966 |        5.7 |           3 |          4.2 |         1.2 |
    1
  9 | 2023-12-30 11:50:22.451067 |        6.2 |         2.2 |          4.5 |         1.5 |
    1
 10 | 2023-12-30 11:50:23.461263 |        5.4 |         3.9 |          1.7 |         0.4 |
    0
 11 | 2023-12-30 11:50:24.472526 |        5.2 |         4.1 |          1.5 |         0.1 |
    0
 12 | 2023-12-30 11:50:25.480016 |        7.2 |         3.2 |            6 |         1.8 |
    2
 13 | 2023-12-30 11:50:26.491447 |        6.3 |         2.7 |          4.9 |         1.8 |
    2
 14 | 2023-12-30 11:50:27.503513 |        6.4 |         3.2 |          5.3 |         2.3 |
    2
 15 | 2023-12-30 11:50:38.515577 |        6.1 |           3 |          4.9 |         1.8 |
    2
:...skipping...
```

데이터 생성 on Docker Compose

1.DB 컨테이너와 Data Generator 컨테이너를 함께 띄우기 위한 Docker Compose 파일을 작성

2.Docker Compose를 이용하기 위해, DB 서버와 Data Generator를 종료

```
$ docker rm --force postgres-server data-generator
```

3.Docker Compose → Docker Compose Healthcheck → Docker Compose Network

데이터 생성 on Docker Compose

1) Docker Compose
- docker-compose.yaml 생성
- postgres server 가 띄워진 뒤에 곧바로 Data Generator 가 띄워짐 → X

```
version: "3"

services:
  postgres-server:
    ...
  data-generator:
    ...
```

데이터 생성 on Docker Compose

2) Docker Compose Healthcheck

- postgres server 가 사용 가능한 상태가 되어있는지 체크하고,
- Data Generator 를 띄움

```
healthcheck:
  test: ["CMD", "pg_isready", "-q", "-U", "myuser", "-d", "mydatabase"]
  interval: 10s
  timeout: 5s
  retries: 5
```
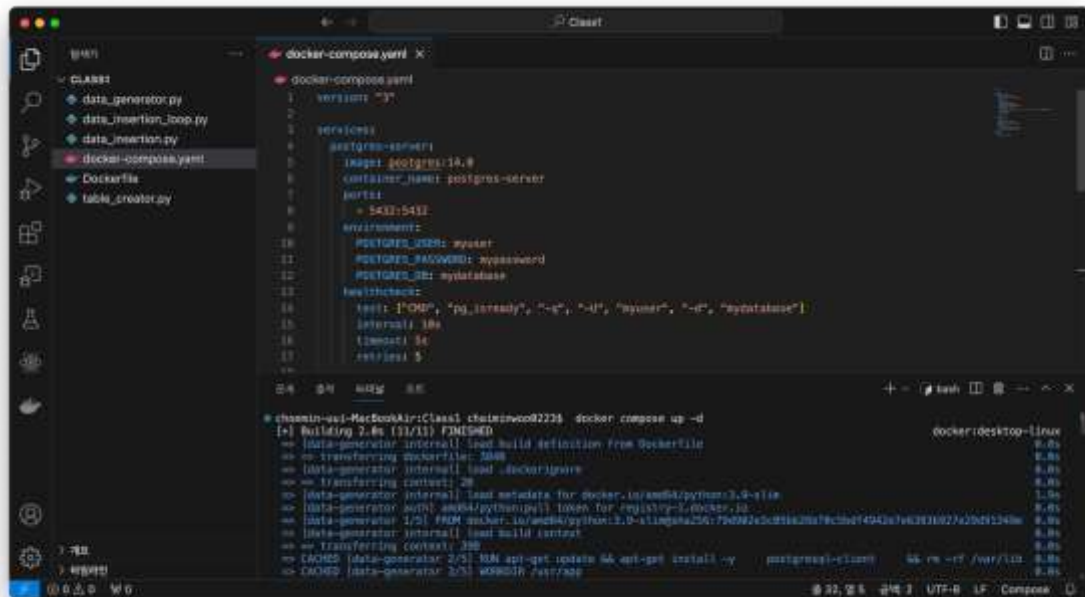
데이터 생성 on Docker Compose

2) Docker Compose Healthcheck

- Compose 파일 실행

```
$ docker compose up -d
```

- Compose 파일 종료

```
$ docker compose down -v
```

데이터 생성 on Docker Compose

2) Docker Compose Healthcheck

- Compose 파일 실행

데이터 생성 on Docker Compose

3) Docker Compose Network

- 네트워크 확인

$ docker network ls

데이터 생성 on Docker Compose

3) Docker Compose Network

- 생성된 네트워크 확인

$ docker network inspect [생성된 네트워크 이름]

```
networks:
  default:
    name: mlops-network
```

데이터 생성 on Docker Compose

3) Docker Compose Network

- 생성된 네트워크 확인

```
● choemin-uui-MacBookAir:Class1 chaiminwoo0223$ docker network inspect mlops-network
[
    {
        "Name": "mlops-network",
        "Id": "601603b2d8fc5a5bcc0b8d3309f533e527412421cf229d717f00d3f94a56b94b",
        "Created": "2024-01-09T08:39:39.936612919Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "ae83f6a9c458a921f80a45f1626471b67bc8f581e143f1ed13b60803f051a184": {
                "Name": "data-generator",
                "EndpointID": "6e2835c9679d85a23d5237a940b587709367c1b7dbe2500523bdc0c4451832ba",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
```

데이터 생성 on Docker Compose

1.psql 실행

2.iris_data 테이블의 데이터 전체를 확인: select * from iris_data;

참고자료

- https://mlops-for-mle.github.io/tutorial/
- https://www.youtube.com/playlist?list=PLuHgQVnccGMDeMJsGq2O-55Ymtx0IdKWf
- https://khj93.tistory.com/entry/Docker-Docker-%EA%B0%9C%EB%85%90
- ChatGPT 4

Thank You