Lab11. Docker Compose를 사용하는 이유

1. Docker 실행 명령어를 일일이 입력하기가 복잡해서

1)Nginx 실행하기

```
$ docker run -it nginx
<--- 실행은 되지만 port binding되지 않아 브라우저로 연결 안됨. 바로 shell로 연결됨. Ctrl + C를 누르면 바로 Container 종료됨. 외부에서 접근할 방법이 없다.
$ curl localhost:80
curl: (7) Failed to connect to localhost port 80 after 0 ms: Connection refused
```

2)Nginx Container 실행 + Host 8080 Port 연결하기

```
$ docker run -it -p 8080:80 nginx
<--- 실행도 되고 웹브라우저로 연결 가능하지만, 바로 shell로 연결됨, Ctrl + C를 누르면 바로 Container 종료됨.
$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

3)Nginx Container 실행 + Host 8080 Port 연결 + Container 종료시 자동 삭제

```
$ docker run -it –p 8080:80 --rm nginx
<---실행도 되고 웹브라우저로 연결 가능하지만, 바로 shell로 연결됨, Ctrl + C로 bash에서 나오면 바로 Container 종료됨
```

4)Nginx 컨테이너 실행 + Host 8080 Port 연결 + Container 종료시 자동 삭제 + Host의 Directory를 Container 안에서 링크하기

```
$ vi index.html
<h1>Hello, Docker Compose World!!!</h1>

$ docker run -it -p 8080:80 --rm -v $PWD:/usr/share/nginx/html nginx
<---실행은 되지만, 웹브라우저로 접속시 403 (13: Permission denied) Error 발생
<---오류를 해결하려면 nginx Container의 /etc/nginx/nginx.conf파일 첫 줄에 user root;를 넣어야 함.
<---어쨌든 복잡함.
$ curl localhost:8080
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.25.3</center>
</body>
</html>
```

2. 컨테이너끼리 연결하기 편해서 --1) 후 바로 --3)실행
1)준비 : django-sample 이미지를 빌드
$ git clone https://github.com/raccoonyy/django-sample-for-docker-compose.git django-sample
$ cd django-sample
$ docker build -t django-sample .

2)django 컨테이너 실행 + postgres 컨테이너 실행
$ docker run --rm -d --name django -p 8000:8000 django-sample

$ docker ps -a

-Web Browser를 열고 http://ip:8000
--django 잘 실행되고 있음을 확인

$ docker run --rm -d --name postgres -e POSTGRES_DB=djangosample ₩
> -e POSTGRES_USER=sampleuser ₩
> -e POSTGRES_PASSWORD=samplesecret ₩
> postgres

-Web Browser를 열고 http://ip:8000
--그냥 django만 잘 실행되고 있음.

-다음 3) 실행을 위해 모든 Docker Image와 Container 삭제하기
$ docker rm -f `docker ps -a -q`
$ docker rmi -f `docker images -q`

-django-sample image 다시 build
$ docker build -t django-sample .

3)postgres 컨테이너 실행 + django 컨테이너 실행 + 서로 연결하기
$ docker run --rm -d --name postgres -e POSTGRES_DB=djangosample ₩
> -e POSTGRES_USER=sampleuser ₩
> -e POSTGRES_PASSWORD=samplesecret ₩
> postgres

$ docker run -d --rm -p 8000:8000 -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample

3. 특정 컨테이너끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
1)postgres 컨테이너 실행 + django1 컨테이너 연결
$ docker run --rm -d --name postgres ₩
> -e POSTGRES_DB=djangosample ₩
> -e POSTGRES_USER=sampleuser ₩
> -e POSTGRES_PASSWORD=samplesecret ₩
> postgres

$ docker run -d --rm --name django1 ₩
> -p 8000:8000 ₩
> -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample

2)postgres 컨테이너는 호스트의 다른 컨테이너들이 모두 접근할 수 있음
$ docker run -d --rm --name django2 ₩
> -p 8001:8000 ₩
> -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample

3)postgres 컨테이너 + django1 컨테이너만 통신할 수 있는 가상 네트워크 만들기

-도커 네트워크 살펴보기

```
$ docker network ls
```

-도커 네트워크 생성하기

```
$ docker network create --driver bridge web-service
$ docker network ls
```

-컨테이너 실행하기

```
$ docker rm -f postgres
$ docker run --rm -d --name postgres ₩
> --network web-service ₩
> -e POSTGRES_DB=djangosample ₩
> -e POSTGRES_USER=sampleuser ₩
> -e POSTGRES_PASSWORD=samplesecret ₩
> postgres

$ docker rm -f django1
$ docker run -d --rm --name django1 ₩
> --network web-service ₩
> -p 8000:8000 ₩
> -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample

$ docker rm -f django2
$ docker run -d --rm --name django2 ₩
> -p 8001:8000 ₩
> -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample
docker: Error response from daemon: Cannot link to /postgres, as it does not belong to the default network.
```

4. 이 모든 것을 간단한 명령어로 관리하고 싶어서

-준비

```
$ docker rm -f `docker container ps -a -q`
$ dockre rmi -f `docker image ls -q`
$ docker build -t django-sample .
$ docker network rm web-service
```

1)실행 명령어와 종료 명령어

```
$ docker network create --driver bridge web-service

$ docker run --rm -d --name postgres ₩
> --network web-service ₩
> -p 5432:5432 ₩
> -e POSTGRES_DB=djangosample ₩
> -e POSTGRES_USER=sampleuser ₩
> -e POSTGRES_PASSWORD=samplesecret ₩
> postgres

$ docker run -d --rm --name django1 ₩
> --network web-service ₩
> -p 8000:8000 ₩
> -e DJANGO_DB_HOST=db ₩
> --link postgres:db ₩
> django-sample
```

```
$ docker kill django1 postgres
$ docker network rm web-service


2)docker-compose.yml 생성하기
$ cd django-sample
$ nano docker-compose.yml

version: '3'

volumes:
    postgres_data: {}

services:
    db:
        image: postgres
        volumes:
          - postgres_data:/var/lib/postgres/data    <--- ':'과 '/'는 반드시 붙인다.
        environment:
          - POSTGRES_DB=djangosample
          - POSTGRES_USER=sampleuser
          - POSTGRES_PASSWORD=samplesecret
    django:
        build:
            context: .
            dockerfile: ./compose/django/Dockerfile-dev
            volumes:
              - ./:/app/
            command: ["./manage.py", "runserver", "0:8000"]
            environment:
              - DJANGO_DB_HOST=db
            depends_on:
              - db
            restart: always
            ports:
              - 8000:8000


3)도커 컴포즈로 실행하고 종료하기
    -모든 docker process 중지
    -모든 docker images 삭제
    -docker network도 기본 3개만 남기고 모두 삭제

    $ docker-compose up -d          <---반드시 django-sample directory 안에서 실행할 것
    $ docker-compose down

    -웹 브라우저로 확인할 것
```