

05.Docker Compose를 활용해 컨테이너 관리하기

1. Docker Compose를 사용하는 이유

✓ Docker Compose란?

여러 개의 Docker 컨테이너들을 하나의 서비스로 정의하고 구성해 하나의 묶음으로 관리할 수 있게 도와주는 툴이다.

✓ Docker Compose를 사용하는 이유

1. 여러 개의 컨테이너를 관리하는 데 용이

여러 개의 컨테이너로 이루어진 복잡한 애플리케이션을 한 번에 관리할 수 있게 해준다. 여러 컨테이너를 하나의 환경에서 실행하고 관리하는 데 도움이 된다.

2. 복잡한 명령어로 실행시키던 걸 간소화 시킬 수 있음

이전에 MySQL 이미지를 컨테이너로 실행시킬 때 아래와 같은 명령어를 실행시켰다.

```
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -v /Users/jaeseong/Documents/Develop/docker-mysql/mysql_data:/var/lib/mysql -d mysql
```

너무 복잡하지 않은가? Docker Compose를 사용하면 위와 같이 컨테이너를 실행시킬 때마다 복잡한 명령어를 입력하지 않아도 된다. 단순히 **docker compose up** 명령어만 실행시키면 된다.

2. [실습] Docker Compose 전체 흐름 느껴보기 (Nginx 설치 및 실행)

✓ Docker CLI로 컨테이너를 실행시킬 때

```
$ docker run --name webserver -d -p 80:80 nginx
```

✓ Docker Compose로 컨테이너를 실행시킬 때

1. compose.yml 파일 작성하기

compose.yml

```
services:
  my-web-server:
    container_name: webserver
    image: nginx
    ports:
      - 80:80
```

- `services: my-web-server`: Docker Compose에서 하나의 컨테이너를 서비스(service)라고 부른다. 이 옵션은 서비스에 이름을 붙이는 기능이다.
- `container_name: webserver`: 컨테이너를 띄울 때 붙이는 별칭이다. CLI에서 `--name webserver` 역할과 동일하다.
- `image: nginx`: 컨테이너를 실행시킬 때 어떤 이미지를 사용할 지 정의하는 명령어이다. `$ docker run [이미지명]`와 동일한 역할이다.
- `ports`: 포트 매핑은 어떻게 할 지를 설정하는 옵션이다. CLI에서 `-p 80:80` 역할과 동일하다.

2. compose 파일 실행시키기

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

3. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

```
apple@appleui-MacBookPro compose-practice % docker compose ps
NAME                IMAGE              COMMAND              SERVICE
CREATED            STATUS            PORTS
webserver           nginx              "/docker-entrypoint..."  my-web-server
4 minutes ago      Up About a minute  0.0.0.0:80->80/tcp

apple@appleui-MacBookPro compose-practice % docker ps
CONTAINER ID   IMAGE      COMMAND              CREATED        STATUS
PORTS         NAMES
703451c58b74   nginx     "/docker-entrypoint..."  4 minutes ago  Up 2 minutes
0.0.0.0:80->80/tcp  webserver
apple@appleui-MacBookPro compose-practice %
```


4. localhost:80 들어가보기



5. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

3. 자주 사용하는 **Docker Compose CLI** 명령어

 **docker-compose**로 시작하는 명령어는 더 이상 업데이트를 지원하지 않는 Docker Compose의 v1 명령어이므로 되도록이면 사용하지 말자. v2부터는 **docker compose**로 시작하는 명령어를 사용한다.

아래 명령어들은 **compose.yml**이 존재하는 디렉토리에서 실행시켜야 한다.

compose 파일 작성

compose.yml

```
services:
  webserver:
    container_name: webserver
    image: nginx
    ports:
      - 80:80
```

compose.yml에서 정의한 컨테이너 실행

```
$ docker compose up      # 포그라운드에서 실행
$ docker compose up -d   # 백그라운드에서 실행
```

-d: 백그라운드에서 실행

✓ Docker Compose로 실행시킨 컨테이너 확인하기

```
# compose.yml에 정의된 컨테이너 중 실행 중인 컨테이너만 보여준다.  
$ docker compose ps  
  
# compose.yml에 정의된 모든 컨테이너를 보여준다.  
$ docker compose ps -a
```

✓ Docker Compose 로그 확인하기

```
# compose.yml에 정의된 모든 컨테이너의 로그를 모아서 출력한다.  
$ docker compose logs
```

```
apple@appleui-MacBookPro compose-practice % docker compose logs  
webserver | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will att  
empt to perform configuration  
webserver | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypo  
int.d/  
webserver | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-  
ipv6-by-default.sh  
webserver | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc  
/nginx/conf.d/default.conf  
webserver | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /e  
tc/nginx/conf.d/default.conf  
webserver | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resol  
vers.envsh  
webserver | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-o  
n-templates.sh  
webserver | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worke  
r-processes.sh
```

✓ 컨테이너를 실행하기 전에 이미지 재빌드하기

```
$ docker compose up --build # 포그라운드에서 실행  
$ docker compose up --build -d # 백그라운드에서 실행
```

compose.yml에서 정의한 이미지 파일에서 코드가 변경 됐을 경우, 이미지를 다시 빌드해서 컨테이너를 실행시켜야 코드 변경된 부분이 적용된다. 그러므로 이럴 때에는 `--build` 옵션을 추가해서 사용해야 한다.

참고 : `docker compose up` vs `docker compose up --build`

- `docker compose up` : 이미지가 없을 때만 빌드해서 컨테이너를 실행시킨다.
이미지가 이미 존재하는 경우 이미지를 빌드하지 않고 컨테이너를 실행시킨다.

- `docker compose up --build`: 이미지가 있건 없건 무조건 빌드를 다시해서 컨테이너를 실행시킨다.

✓ 이미지 다운받기 / 업데이트하기

```
$ docker compose pull
```

`compose.yml`에서 정의된 이미지를 다운 받거나 업데이트 한다.

로컬 환경에 이미지가 없다면 이미지를 다운 받는다.

로컬 환경에 이미 이미지가 있는데, **Dockerhub**의 이미지와 다른 이미지일 경우 이미지를 업데이트 한다.

✓ Docker Compose에서 이용한 컨테이너 종료하기

```
$ docker compose down
```

4. [실습] Docker Compose로 Redis 실행시키기

✓ Docker CLI로 컨테이너를 실행시킬 때

```
$ docker run -d -p 6379:6379 redis
```

✓ Docker Compose로 컨테이너를 실행시킬 때

1. compose.yml 파일 작성하기

compose.yml

```
services:
  my-cache-server:
    image: redis
    ports:
      - 6379:6379
```

2. compose 파일 실행시키기

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

3. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

```
apple@appleui-MacBookPro compose-practice % docker compose up -d
[+] Running 9/9
  ✓ my-cache-server 8 layers [██████████] 0B/0B Pulled 7.6s
  ✓ bc0965b23a04 Already exists 0.0s
  ✓ 9501a6ec095f Pull complete 0.9s
  ✓ 98e7597530ef Pull complete 0.9s
  ✓ 75dffa679c9b Pull complete 1.1s
  ✓ 8912a88e73c8 Pull complete 2.3s
  ✓ 141f00d6fee8 Pull complete 2.3s
  ✓ 4f4fb700ef54 Pull complete 2.4s
  ✓ 8242f9d5b464 Pull complete 2.6s
[+] Running 2/2
  ✓ Network compose-practice_default Created 0.1s
  ✓ Container compose-practice-my-cache-server-1 Started 0.8s
apple@appleui-MacBookPro compose-practice % docker compose ps
NAME                                IMAGE                                COMMAND
SERVICE                            CREATED                            STATUS                            PORTS
compose-practice-my-cache-server-1  redis                              "docker-entrypoint.s..."
my-cache-server                    15 seconds ago                    Up 13 seconds                    0.0.0.0:6379->6379
/tcp
apple@appleui-MacBookPro compose-practice % docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED                            STATUS
PORTS
NAMES
33a102871cf8   redis                              "docker-entrypoint.s..."  18 seconds ago                    Up 17 seconds
0.0.0.0:6379->6379/tcp   compose-practice-my-cache-server-1
```

4. 컨테이너 실행시킬 때 에러 없이 잘 실행됐는 지 로그 체크

```
$ docker logs [컨테이너 ID 또는 컨테이너명]
```

5. Redis 컨테이너에 접속

```
$ docker exec -it [컨테이너 ID 또는 컨테이너명] bash
```

6. 컨테이너에서 redis 사용해보기

```
$ redis-cli

127.0.0.1:6379> set 1 jscode

127.0.0.1:6379> get 1
```

```
apple@appleui-MacBookPro compose-practice % docker exec -it 33a bash
root@33a102871cf8:/data# redis-cli
127.0.0.1:6379> set 1 jscore
OK
127.0.0.1:6379> get 1
"jscore"
127.0.0.1:6379> █
```

7. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

5. [실습] Docker Compose로 MySQL 실행시키기

✓ Docker CLI로 컨테이너를 실행시킬 때

```
$ docker run -e MYSQL_ROOT_PASSWORD=pwd1234 -p 3306:3306 -v
/Users/jaeseong/Documents/Develop/docker-mysql/mysql_data:/var/lib
/mysql -d mysql
```

✓ Docker Compose로 MySQL 실행시키기

1. compose 파일 작성하기

compose.yml

```
services:
  my-db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: pwd1234
    volumes:
      - ./mysql_data:/var/lib/mysql
    ports:
      - 3306:3306
```

- **environment: ...** : CLI에서 **-e MYSQL_ROOT_PASSWORD=password** 역할과 동일하다.
- **volumes: ...** : CLI에서 **-v {호스트 경로}:/var/lib/mysql** 역할과 동일하다.

2. compose 파일 실행시키기

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

3. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

```
apple@appleui-MacBookPro compose-practice % docker exec -it fcf bash
bash-5.1# mysql -u root -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.1.0 MySQL Community Server - GPL

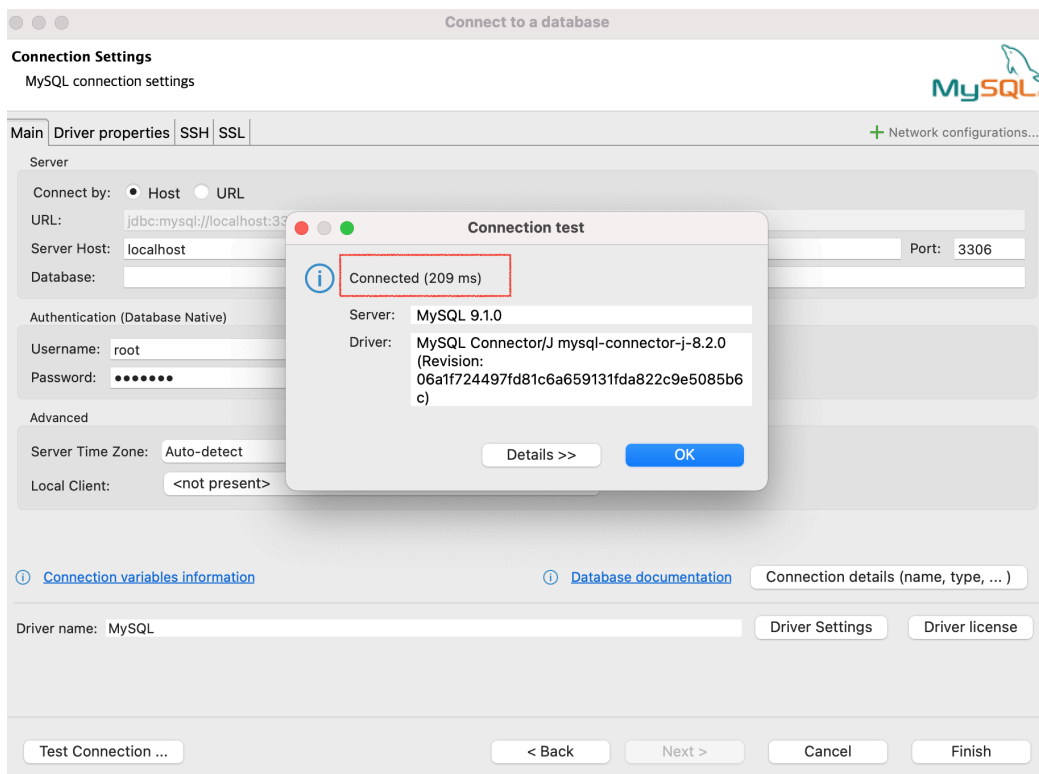
Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

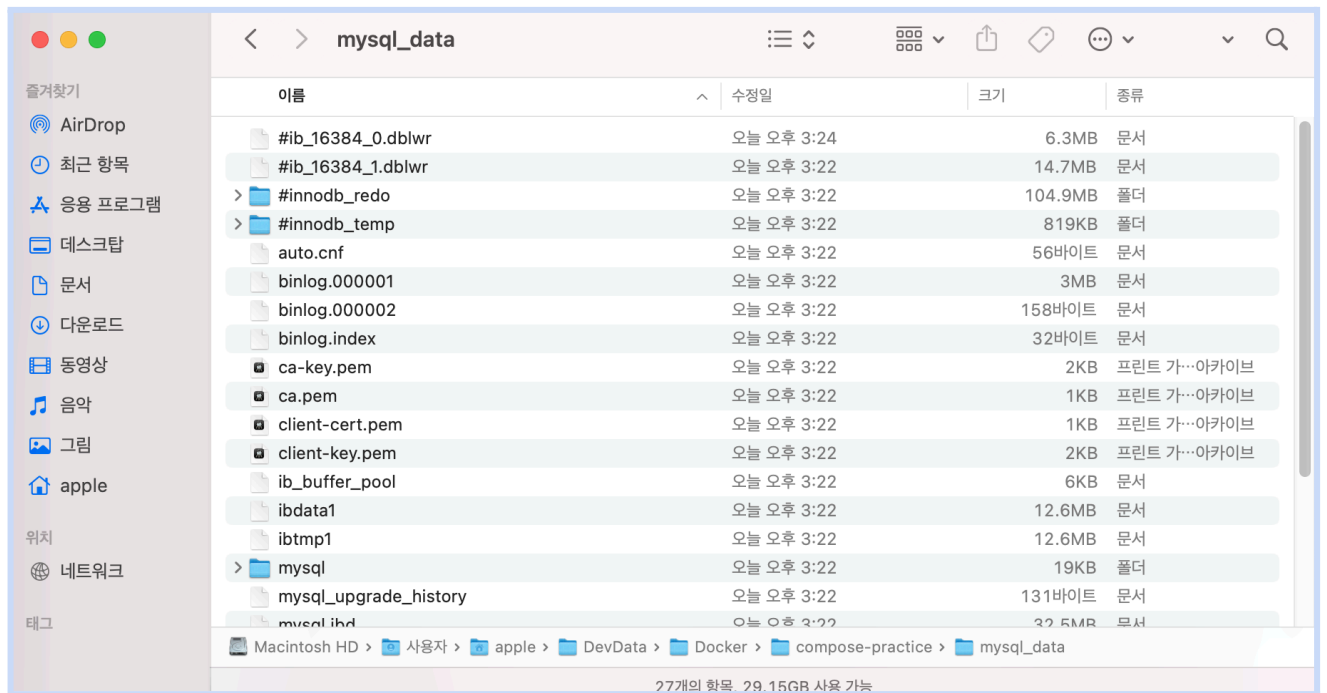
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

4. 잘 작동하는 지 DBeaver에 연결시켜보기



5. volume의 경로에 데이터가 저장되고 있는 지 확인하기



6. compose로 실행된 컨테이너 삭제

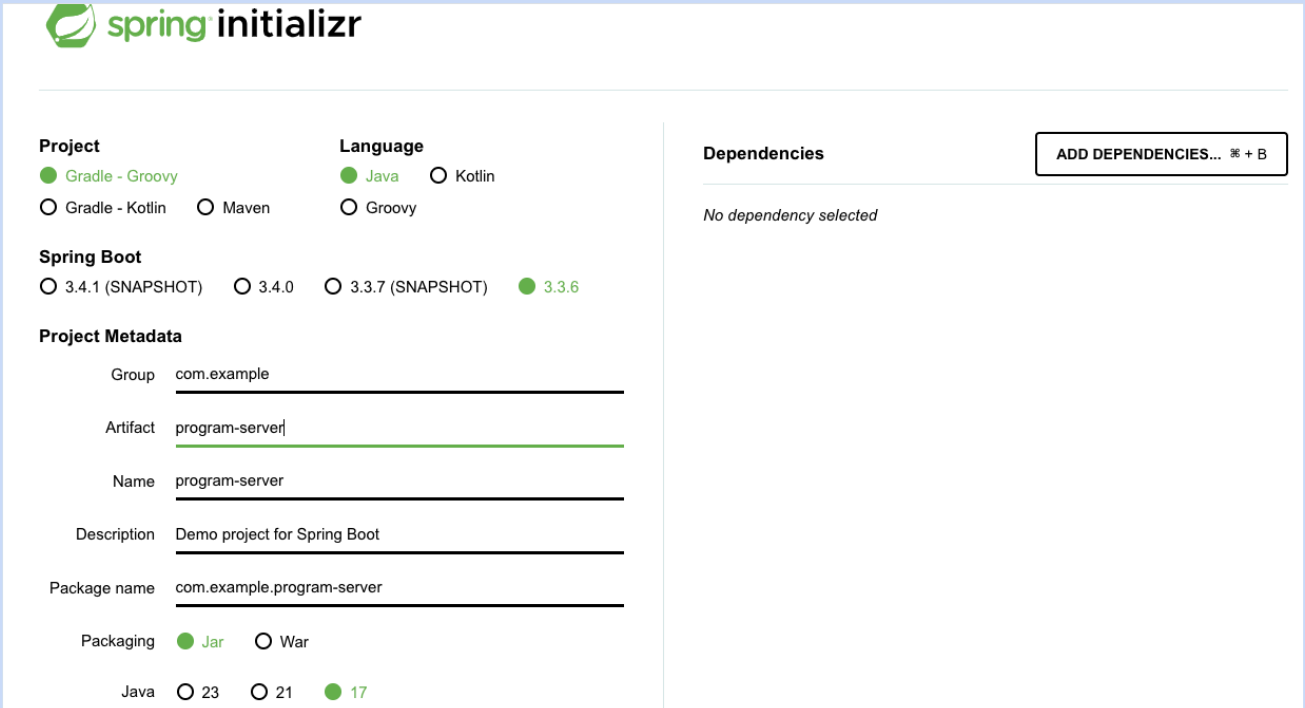
```
$ docker compose down
```

6. [실습] Docker Compose로 백엔드(Spring Boot) 실행시키기

✓ Docker Compose로 백엔드(Spring Boot) 실행시키기

1. 프로젝트 셋팅

start.spring.io



The image shows the Spring Initializr web form. It is divided into three main sections: Project, Language, and Dependencies. The Project section has radio buttons for Gradle - Groovy (selected), Gradle - Kotlin, and Maven. The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for 3.4.1 (SNAPSHOT), 3.4.0, 3.3.7 (SNAPSHOT), and 3.3.6 (selected). The Project Metadata section has input fields for Group (com.example), Artifact (program-server), Name (program-server), Description (Demo project for Spring Boot), and Package name (com.example.program-server). The Packaging section has radio buttons for Jar (selected) and War. The Java section has radio buttons for 23, 21, and 17 (selected). The Dependencies section has a button labeled 'ADD DEPENDENCIES... ⌘ + B' and the text 'No dependency selected'.

- Java 17 버전을 선택하자. 아래 과정을 Java 17 버전을 기준으로 진행할 예정이다.

2. 간단한 코드 작성

AppController

```
package com.example.program_server;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AppController {
    @GetMapping("/")
    public String home() {
        return "Hello, World!";
    }
}
```

3. Dockerfile 작성하기

Dockerfile

```
FROM openjdk:17-jdk

COPY build/libs/*SNAPSHOT.jar /app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

4. Spring Boot 프로젝트 빌드하기

```
$ ./gradlew clean build
```

```
apple@appleui-MacBookPro docker % cd instagram-server
apple@appleui-MacBookPro instagram-server % ./gradlew clean build
Starting a Gradle Daemon (subsequent builds will be faster)
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended

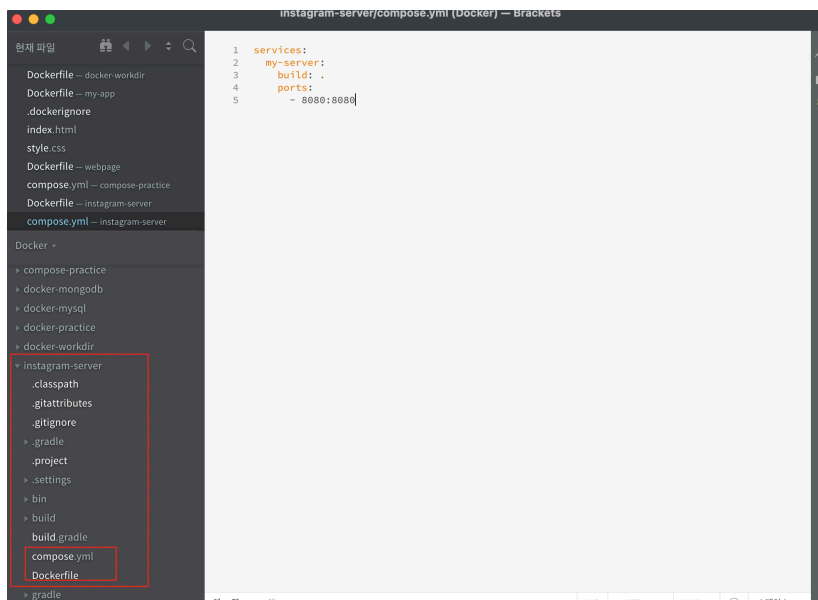
BUILD SUCCESSFUL in 17s
8 actionable tasks: 8 executed
apple@appleui-MacBookPro instagram-server %
```

5. compose 파일 작성하기

- 참고) compose를 작성하지 않고 Docker CLI로 실행시킬 때

compose.yml

```
services:
  my-server:
    build: .
    ports:
      - 8080:8080
```



- `build: .: compose.yml`이 존재하는 디렉토리(.)에 있는 `Dockerfile`로 이미지를 생성해 컨테이너를 띄우겠다는 의미이다.

6. compose 파일 실행시키기

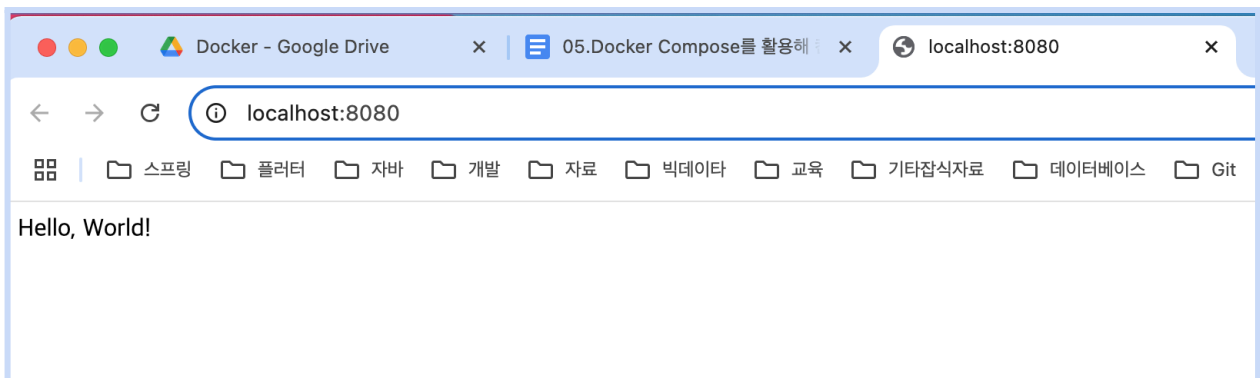
```
$ docker compose up -d --build
```

```
apple@appleui-MacBookPro instagram-server % docker compose up -d --build
[+] Building 5.5s (7/7) FINISHED
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                  0.0s
=> [internal] load build definition from Dockerfile            0.0s
=> => transferring dockerfile: 136B                             0.0s
=> [internal] load metadata for docker.io/library/openjdk:17-jdk 4.4s
=> CACHED [1/2] FROM docker.io/library/openjdk:17-jdk@sha256:528707081fd 0.0s
=> [internal] load build context                                0.7s
=> => transferring context: 20.66MB                             0.6s
=> [2/2] COPY build/libs/*SNAPSHOT.jar app.jar                0.2s
=> exporting to image                                           0.2s
=> => exporting layers                                          0.2s
=> => writing image sha256:ff8e3610f5c5e79577879de823538287aa96a1e7d5188 0.0s
=> => naming to docker.io/library/instagram-server-my-server 0.0s
[+] Running 2/2
✔ Network instagram-server_default                            Created      0.2s
✔ Container instagram-server-my-server-1                     Started       1.0s
```

7. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

8. localhost:8080으로 들어가보기



9. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

7. [실습] Docker Compose로 백엔드(Nest.js) 실행시키기

✓ Docker Compose로 백엔드(Nest.js) 실행시키기

1. Nest.js 프로젝트 만들기

```
# Nest CLI 설치
$ npm i -g @nestjs/cli # 안되면 sudo 앞에 넣을 것.

# nest new {프로젝트명}
$ nest new my-server
```

```
apple@appleui-MacBookPro docker % sudo nest new my-server
⚡ We will scaffold your app in a few seconds..
```

```
? Which package manager would you ❤️ to use? npm
Nothing to be done.
```

```
✓ Installation in progress... ☕
```

```
🚀 Successfully created project my-server
👉 Get started with the following commands:
```

```
$ cd my-server
$ npm run start
```

Thanks for installing Nest 🙏
Please consider donating to our open collective
to help us maintain this package.

🍷 Donate: <https://opencollective.com/nest>

2. Dockerfile 작성하기

Dockerfile

```
FROM node

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build
```

```
EXPOSE 3000
```

```
ENTRYPOINT [ "node", "dist/main.js" ]
```

3. .dockerignore 작성하기

.dockerignore

```
node_modules
```

- 이미지를 생성할 때 `npm install`을 통해 처음부터 깔끔하게 필요한 의존성만 설치한다. 따라서 호스트 컴퓨터에 있는 `node_modules`는 컨테이너로 복사해갈 필요가 없다.

4. compose 파일 작성하기

- 참고) `compose`를 작성하지 않고 `Docker CLI`로 실행시킬 때

compose.yml

```
services:
  my-server:
    build: .
    ports:
      - 3000:3000
```

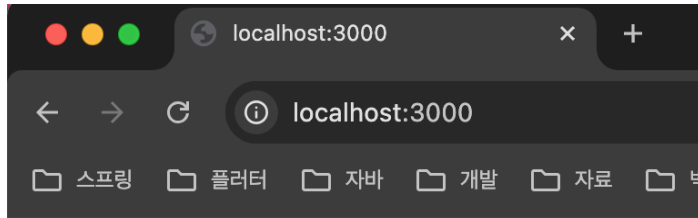
5. compose 파일 실행시키기

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

6. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

7. localhost:3000으로 들어가보기



Hello World!

8. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

8. [실습] Docker Compose로 프론트엔드(Next.js) 실행시키기

✓ Docker Compose로 프론트엔드(Next.js) 실행시키기

1. Next.js 프로젝트 만들기(앞의 예제 사용해도 됨)

```
$ npx create-next-app@latest
```

2. Dockerfile 작성하기

Dockerfile

```
FROM node:20-alpine

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build

EXPOSE 3000

ENTRYPOINT [ "npm", "run", "start" ]
```

3. .dockerignore 작성하기

.dockerignore

```
node_modules
```

- 이미지를 생성할 때 `npm install`을 통해 처음부터 깔끔하게 필요한 의존성만 설치한다. 따라서 호스트 컴퓨터에 있는 `node_modules`는 컨테이너로 복사해갈 필요가 없다.

4. compose 파일 작성하기

- 참고) `compose`를 작성하지 않고 Docker CLI로 실행시킬 때

compose.yml

```
services:
  my-web-server:
```



```
build: .
ports:
  - 80:3000
```

5. compose 파일 실행시키기

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

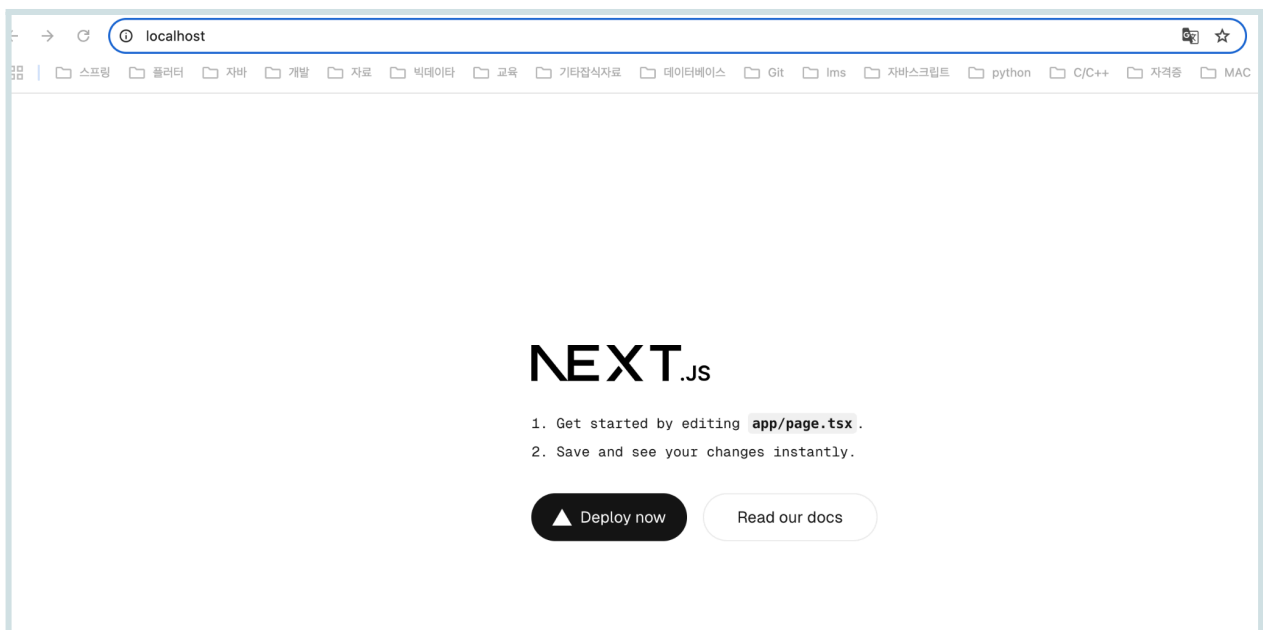
6. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

```
[apple@appleui-MacBookPro my-app % docker compose ps
NAME                                IMAGE                                COMMAND                                SERVICE
  CREATED                          STATUS                              PORTS                                my-web-serve
my-app-my-web-server-1             my-app-my-web-server               "npm run start"                      0.0.0.0:80->3000/tcp
r 12 seconds ago                   Up 11 seconds

[apple@appleui-MacBookPro my-app % docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED          STATUS
PORTS          NAMES
35db82cb2a40   my-app-my-web-server               "npm run start"                      16 seconds ago   Up 14 s
econds        0.0.0.0:80->3000/tcp               my-app-my-web-server-1
```

7. localhost:80으로 들어가보기



8. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

9. [실습] Docker Compose로 프론트엔드(HTML, CSS, Nginx) 실행시키기

✓ Docker Compose로 프론트엔드(HTML, CSS, Nginx) 실행시키기

1. HTML, CSS 파일 만들기

index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>My Web Page</h1>
</body>
</html>
```

- 주의) Nginx의 기본 설정에 의하면 메인 페이지(첫 페이지)의 파일명을 `index.html`이라고 지어야 한다.

style.css

```
* {
  color: blue;
}
```

2. Dockerfile 작성하기

Dockerfile

```
FROM nginx
COPY ./ /usr/share/nginx/html
```

3. compose 파일 작성하기

- 참고) compose를 작성하지 않고 Docker CLI로 실행시킬 때

compose.yml

```
services:
  my-web-server:
```

```
build: .
ports:
  - 80:80
```

4. compose 파일 실행시키기

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

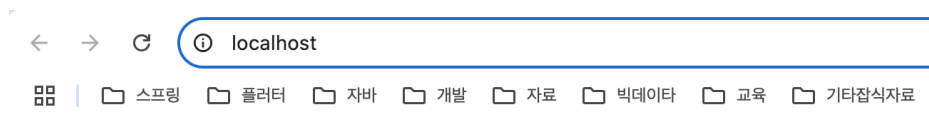
```
apple@appleui-MacBookPro docker % cd webpage
apple@appleui-MacBookPro webpage % docker compose up -d --build
[+] Building 0.1s (7/7) FINISHED
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 80B 0.0s
=> [internal] load metadata for docker.io/library/nginx:latest 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 443B 0.0s
=> CACHED [1/2] FROM docker.io/library/nginx 0.0s
=> [2/2] COPY ./ /usr/share/nginx/html 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:e01da6ddb5ca490be27d4a36023c153d9945b967cc439 0.0s
=> => naming to docker.io/library/webpage-my-web-server 0.0s
[+] Running 2/2
✔ Network webpage_default Created 0.1s
✔ Container webpage-my-web-server-1 Started 0.8s
apple@appleui-MacBookPro webpage %
```

5. compose 실행 현황 보기

```
$ docker compose ps
$ docker ps
```

```
apple@appleui-MacBookPro webpage % docker compose ps
NAME                                IMAGE                                COMMAND                                SERVI
CE                                CREATED                                STATUS                                PORTS
webpage-my-web-server-1            webpage-my-web-server              "/docker-entrypoint...."          my-we
b-server                          49 seconds ago                    Up 48 seconds                        0.0.0.0:80->80/tcp
apple@appleui-MacBookPro webpage % docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED
STATUS        PORTS                                NAMES
b31678700c42  webpage-my-web-server              "/docker-entrypoint...."          52 seconds ago
Up 50 seconds  0.0.0.0:80->80/tcp                  webpage-my-web-server-1
```

6. localhost:80으로 들어가보기



My Web Page

7. compose로 실행된 컨테이너 삭제

```
$ docker compose down
```

10. Docker CLI ↔ Docker Compose 쉽게 작성하기

지금까지의 예제를 보면 **Docker CLI**로 작성할 수 있는 명령어는 전부 **compose.yml** 파일로 옮길 수 있다. 반대로 **compose.yml**에 작성한 모든 값은 **Docker CLI**로 나타낼 수 있다. 이를 편하게 변환해주는 사이트가 존재한다.

✓ **Docker CLI → compose.yml**로 변환

[Composerize](https://www.decomposerize.com/)

✓ **compose.yml → Docker CLI**로 변환

<https://www.decomposerize.com/>