

Docker A to Z

목차

1. Docker란?
2. Docker CLI
3. Docker 볼륨
4. Dockerfile
5. Docker Compose

1. Docker란?

Docker란?

컨테이너를 사용하여 각각의 프로그램을 분리된 환경에서 실행 및 관리할 수 있는 툴

1. Docker란?

Docker를 왜 사용할까?

Docker를 쓰는 이유에는 여러 가지 장점이 있지만 가장 핵심적인 장점은 **이식성**입니다.

이식성: 특정 프로그램을 다른 곳으로 쉽게 옮겨서 설치 및 실행할 수 있는 특성

1. Docker란?

Docker를 왜 사용할까?

<예시>

A의 컴퓨터에 MySQL을 깔고 정상 작동하는 것을 확인했습니다. 그런데 B의 컴퓨터에 MySQL을 깔았더니 에러가 발생했습니다.
왜 이런 상황이 발생할까요?

B의 컴퓨터에 에러가 발생하는 이유는 다양합니다.

A와 다른 버전을 설치했거나, 운영체제가 다르거나, B의 컴퓨터에 깔려있는 다른 프로그램과 충돌이 일어났거나와 같은 다양한 이유로 프로그램이 정상적으로 설치되지 않을 수 있습니다.

에러를 해결하기 위해서는 복잡한 과정을 거쳐야 합니다.

이러한 상황을 해결해주는 도구가 **Docker** 입니다.

Docker를 사용하면 명령어 한 줄로 어떤 컴퓨터에든 MySQL을 에러 없이 설치하고 실행할 수 있습니다.

1. Docker란?

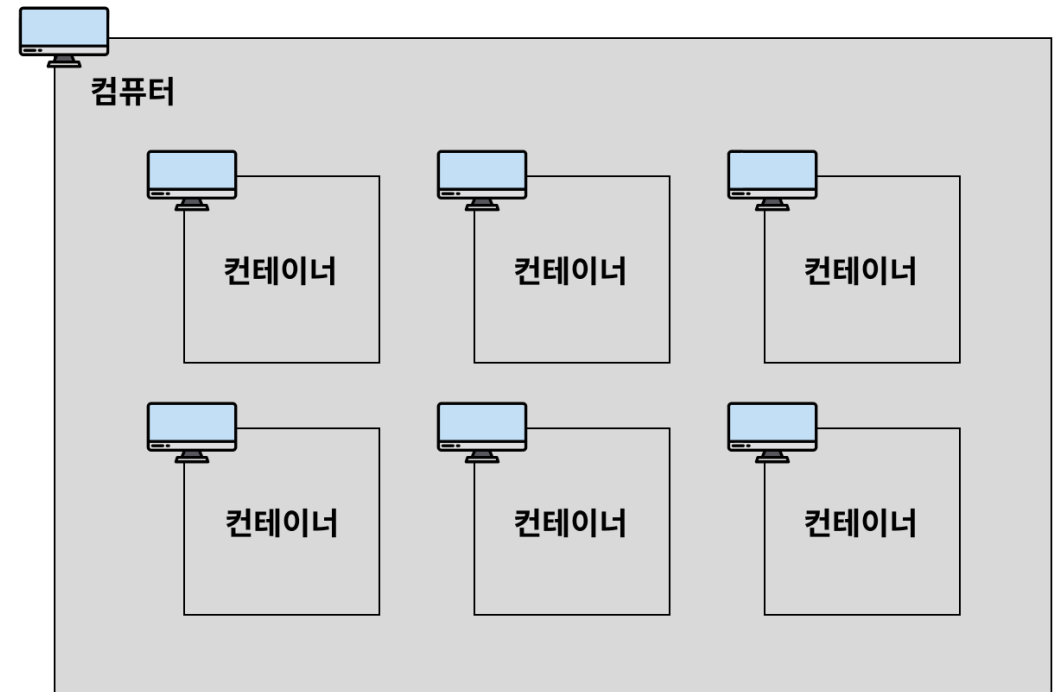
Docker의 기본 개념

컨테이너(Container): 오른쪽 그림에서 나와있는 분리된 환경

이미지(Image): 컨테이너를 정의하는 읽기 전용 템플릿


이미지는 프로그램을 실행하는 데 필요한 **설치 과정, 설정, 버전 정보 등 프로그램을 실행하는 데 필요한 모든 것을 포함하고** 있습니다.

Docker를 통해 하나의 컴퓨터 **환경에서 독립적인 컴퓨터 환경**을 구성해서 각 환경에 프로그램을 별도로 설치할 수 있습니다.



2. Docker CLI

이미지

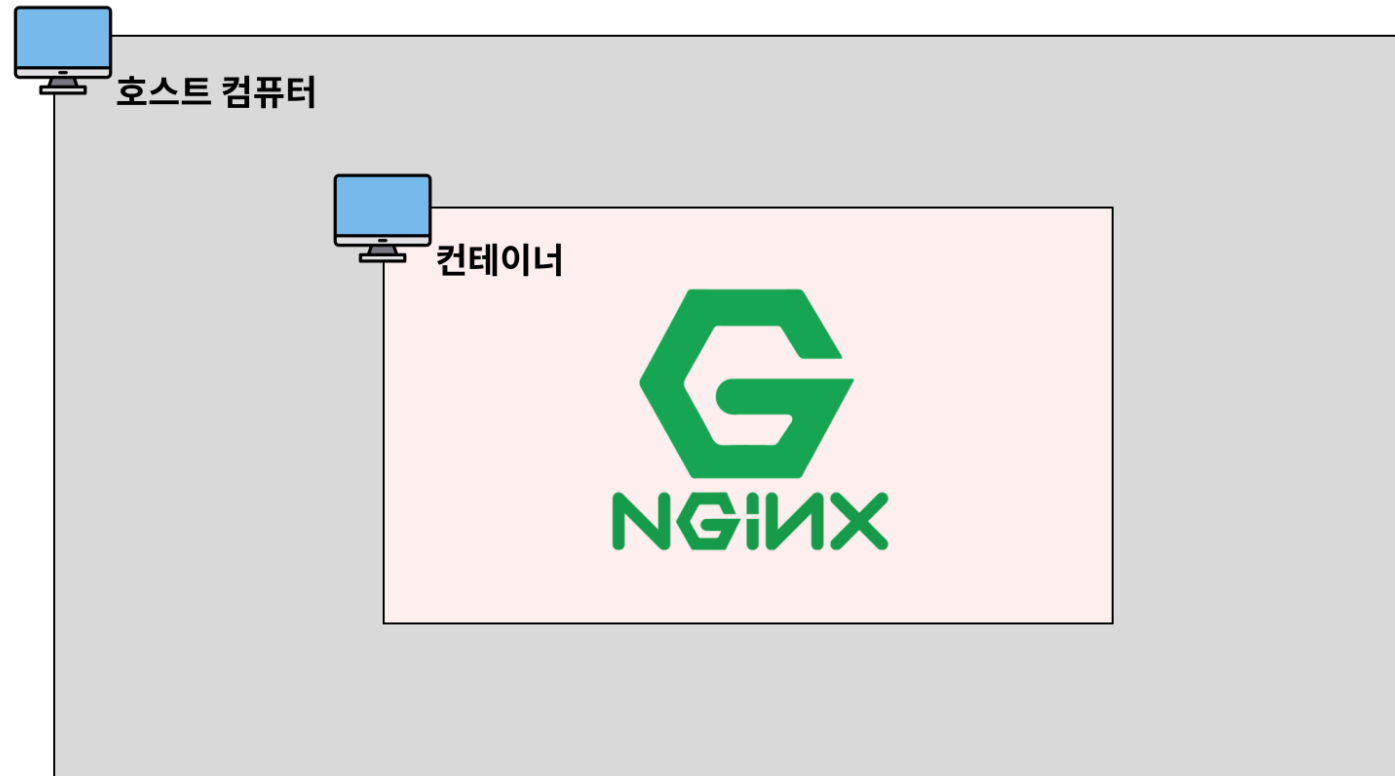


```
# 1. 이미지 다운
# docker pull 이미지명
$ docker pull nginx # docker pull nginx:latest와 동일하게 작동
# 2. 이미지 조회
$ docker image ls

# 3. 이미지 제거
# 특정 이미지 삭제
$ docker image rm [이미지 ID 또는 이미지명]
# 중지된 컨테이너에서 사용하고 있는 이미지 강제 삭제하기
$ docker image rm -f [이미지 ID 또는 이미지명]
```

2. Docker CLI

컨테이너

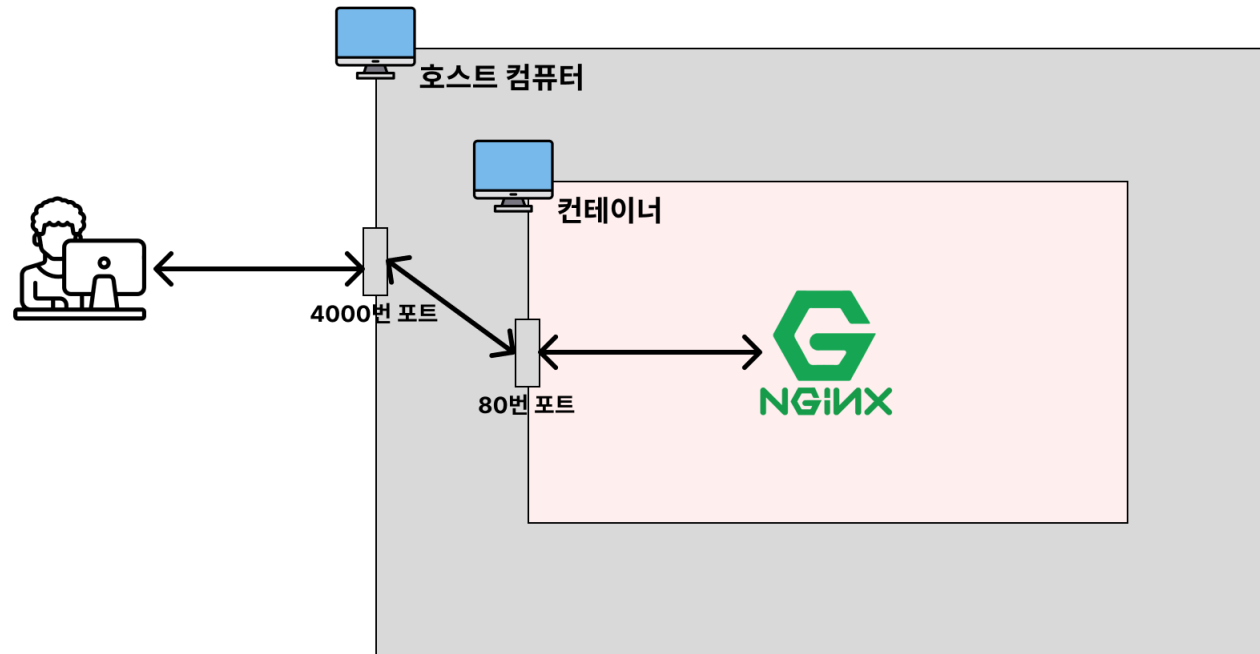


2. Docker CLI

컨테이너



```
# docker run -d -p [호스트 포트]:[컨테이너 포트] 이미지명[:태그명]  
$ docker run -d -p 4000:80 nginx
```



2. Docker CLI

컨테이너

1. 컨테이너 조회

\$ docker ps # 실행 중인 컨테이너들만 조회

\$ docker ps -a # 모든 컨테이너 조회(작동 중인 컨테이너 + 작동을 멈춘 컨테이너)

2. 컨테이너 중지

\$ docker stop 컨테이너명[또는 컨테이너 ID]

\$ docker kill 컨테이너명[또는 컨테이너 ID]

3. 컨테이너 삭제

\$ docker rm 컨테이너명[또는 컨테이너 ID] # 중지되어 있는 특정 컨테이너 삭제

\$ docker rm -f 컨테이너명[또는 컨테이너 ID] # 실행되고 있는 특정 컨테이너 삭제

2. Docker CLI

컨테이너



```
# 4. 컨테이너 로그 조회
```

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

```
$ docker run -d nginx
```

```
$ docker logs [nginx가 실행되고 있는 컨테이너 ID]
```

```
# 5. 실행 중인 컨테이너 내부에 접속하기
```

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

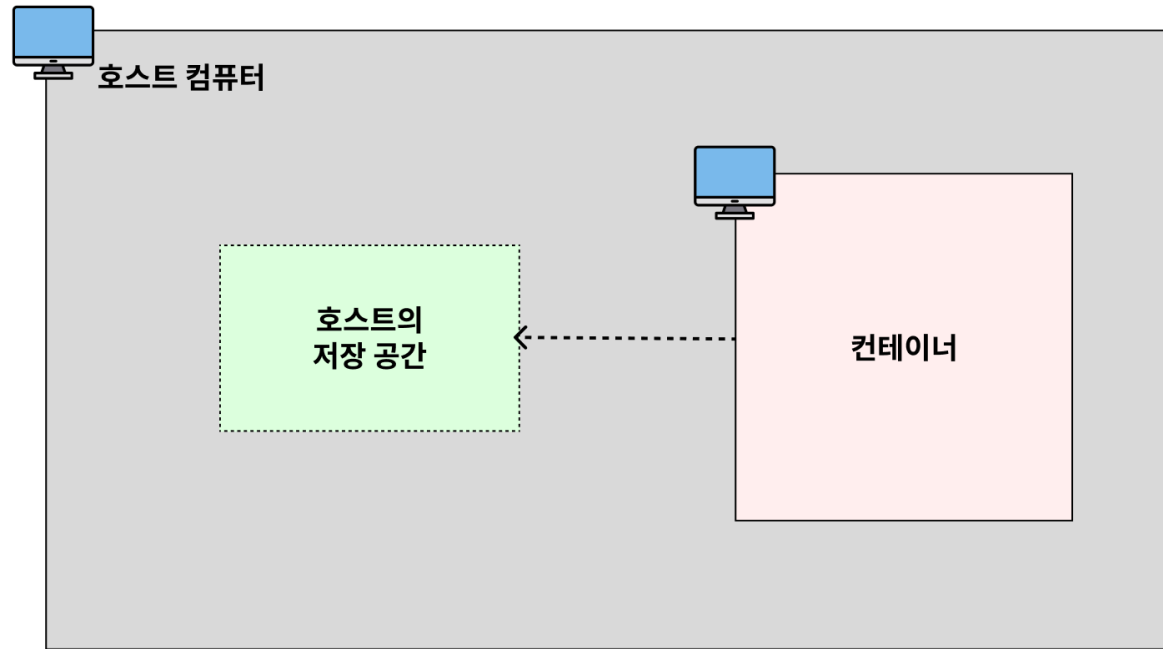
```
$ docker run -d nginx
```

```
$ docker exec -it [Nginx가 실행되고 있는 컨테이너 ID] bash
```

3. Docker 볼륨

도커의 볼륨(Volume)이란 **도커 컨테이너에서 데이터를 영속적으로 저장하기 위한 방법**입니다.

볼륨(Volume)은 컨테이너 자체의 저장 공간을 사용하지 않고, 호스트 자체의 저장 공간을 공유해서 사용하는 형태입니다.



3. Docker 볼륨

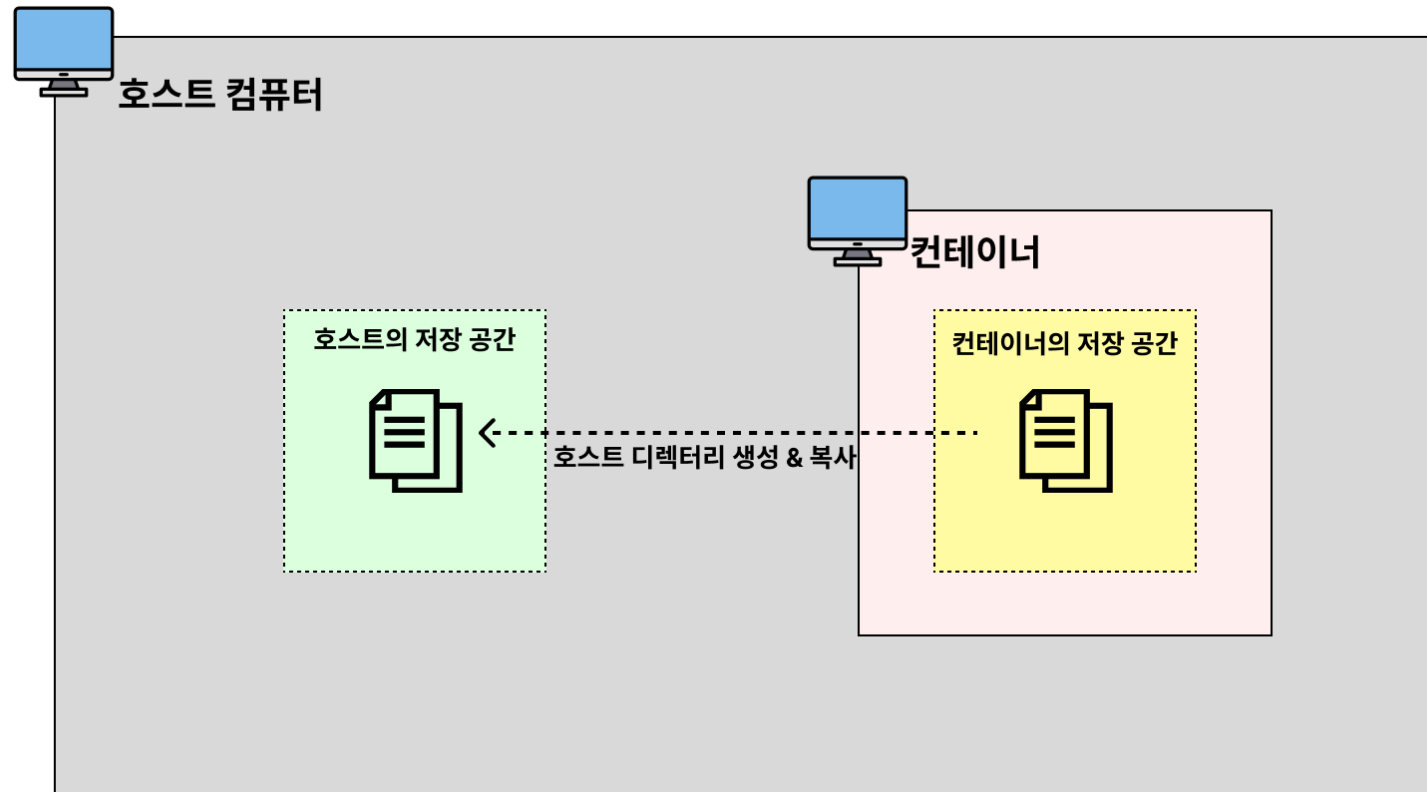


```
$ cd /Users/yereumi/Documents/Develop
$ mkdir docker-mysql # MySQL 데이터를 저장하고 싶은 폴더 만들기

# docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306
-v {호스트의 절대경로}/mysql_data:/var/lib/mysql -d mysql
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306
-v /Users/yereumi/Documents/Develop/docker-
mysql/mysql_data:/var/lib/mysql -d mysql
```

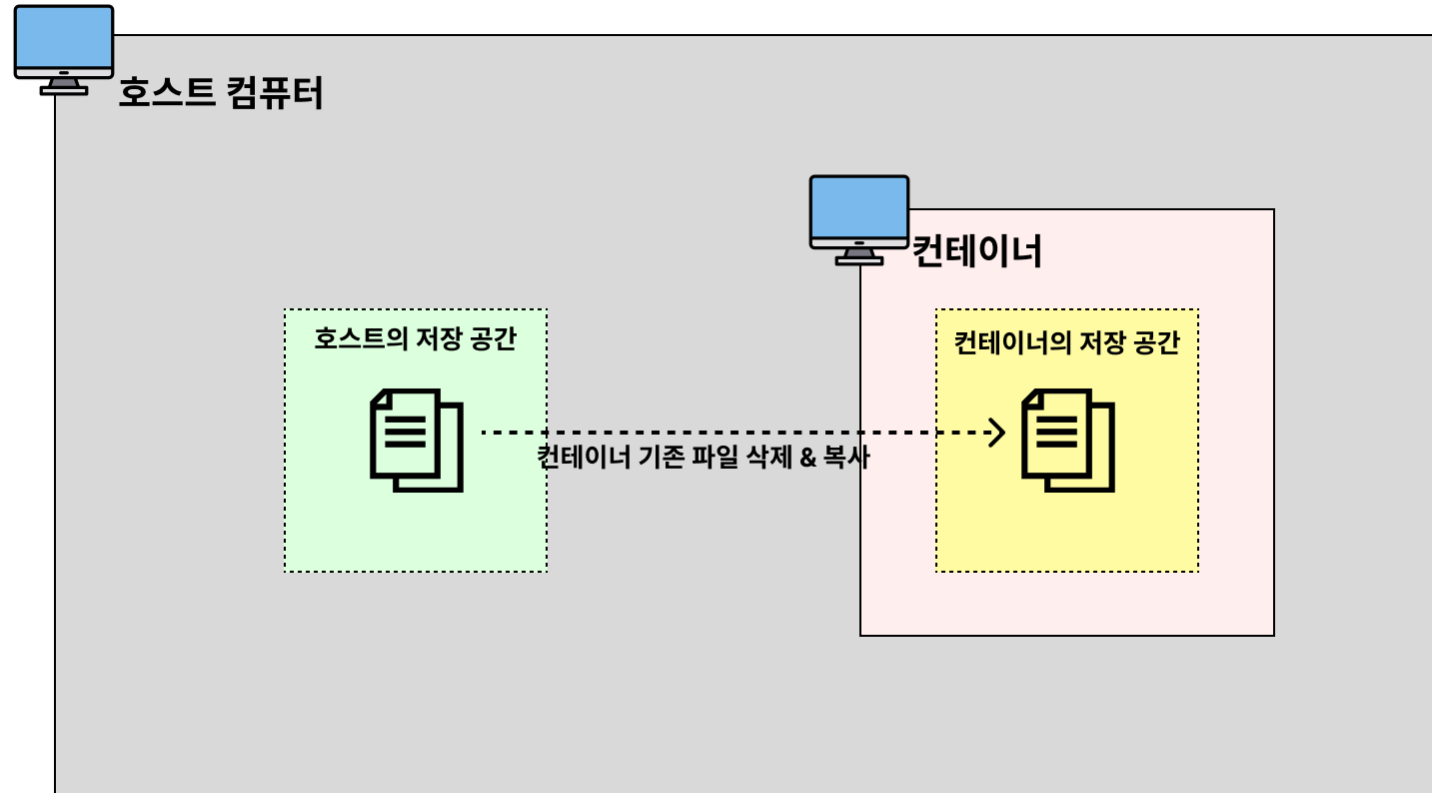
3. Docker 볼륨

호스트의 디렉토리 절대 경로에 디렉토리가 존재하지 않을 경우, 호스트의 디렉터리 절대 경로에 디렉터리를 새로 만들고 컨테이너의 디렉터리에 있는 파일들을 호스트의 디렉터리로 복사해옴



3. Docker 볼륨

호스트의 디렉토리 절대 경로에 디렉토리가 이미 존재할 경우, 호스트의 디렉터리가 컨테이너의 디렉터리를 덮어씌움



4. Dockerfile

Dockerfile이란 Docker 이미지를 만들게 해주는 파일입니다.



FROM - 베이스 이미지를 생성하는 역할

FROM [이미지명]

FROM [이미지명]:[태그명]

COPY - 호스트 컴퓨터에 있는 파일을 복사해서 컨테이너로 전달

COPY [호스트 컴퓨터에 있는 복사할 파일의 경로] [컨테이너에서 파일이 위치할 경로]

ENTRYPOINT - 컨테이너가 생성되고 최초로 실행할 때 수행되는 명령어

ENTRYPOINT [명령문...]

4. Dockerfile

스프링 부트 예시



```
FROM openjdk:17-jdk
```

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

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

4. Dockerfile

스프링 부트 예시



```
$ ./gradlew clean build  
  
$ docker build -t hello-server .  
  
$ docker image ls
```

```
yereumi docker-practice % docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-server	latest	b4aaa2885b75	6 seconds ago	563MB
docker-practice-my-server	latest	d9de8039abd3	4 hours ago	563MB
<none>	<none>	d8e449cbad5a	4 hours ago	563MB
<none>	<none>	d46112fb965b	4 hours ago	563MB
<none>	<none>	4ac86ab5d5c6	4 hours ago	563MB
<none>	<none>	60ab58adf72e	4 hours ago	554MB
prom/prometheus	latest	bddbc4cc906	6 weeks ago	289MB
grafana/grafana	latest	58ab523165b1	7 weeks ago	539MB
mysql	latest	2700d94f59a8	2 months ago	814MB

4. Dockerfile

스프링 부트 예시

```
yereumi docker-practice % docker run -d -p 8080:8080 hello-server  
d6d6f9de068d4e5b8fc99f05076ba515cb4a6267f1516d71055836289831e410
```

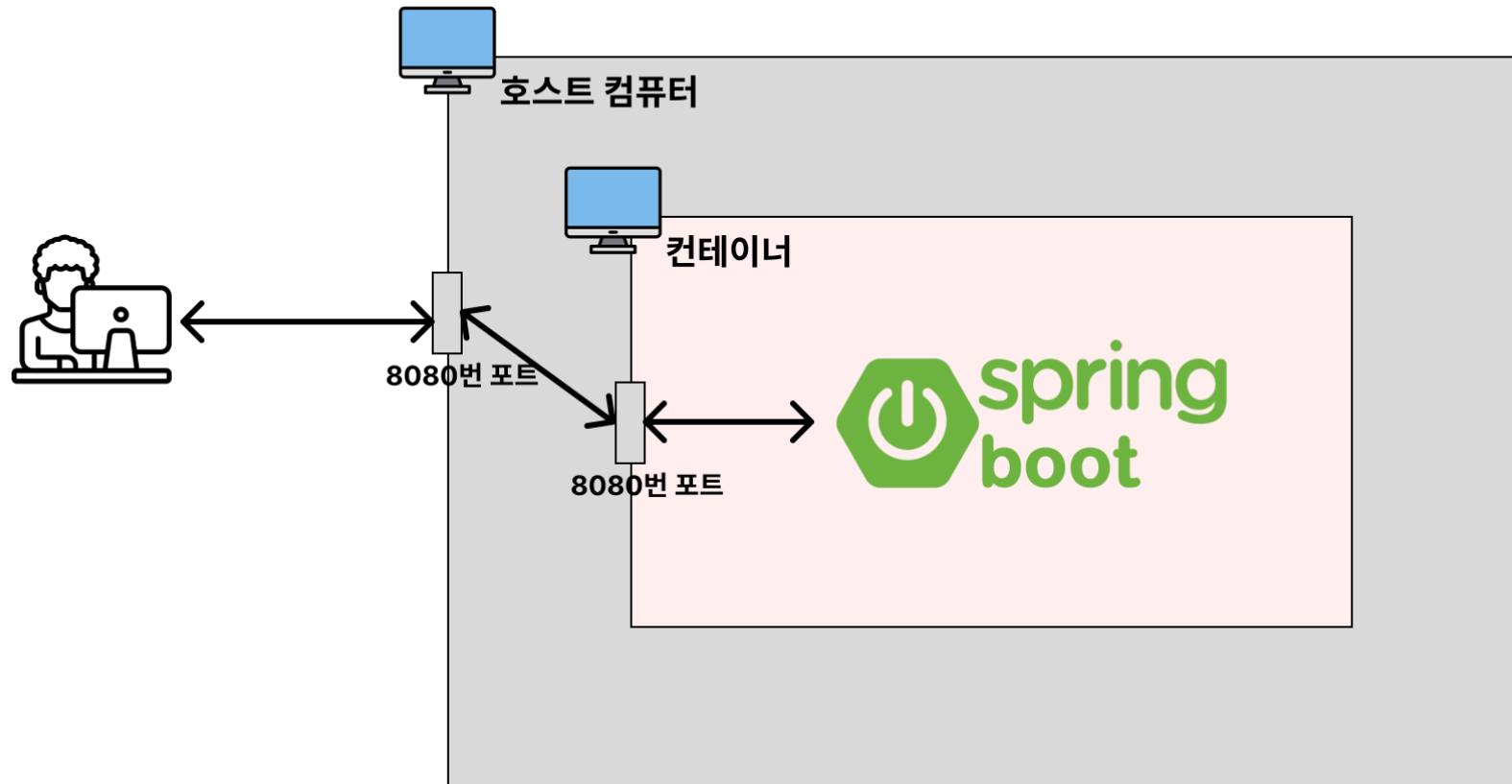


← → ↻ ⓘ localhost:8080

Hello, World!

4. Dockerfile

스프링 부트 예시



5. Docker Compose

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

Docker Compose를 사용하는 이유

1. 여러 개의 컨테이너를 관리하는 데 용이

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

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
```

Docker Compose를 사용하면 위와 같이 컨테이너를 실행시킬 때마다 복잡한 명령어를 입력하지 않고, 단순히 **docker compose up** 명령어만 실행시키면 됩니다.

5. Docker Compose



```
# compose.yaml에서 정의한 컨테이너 실행
$ docker compose up      # 포그라운드에서 실행
$ docker compose up -d   # 백그라운드에서 실행

# Docker Compose로 실행시킨 컨테이너 확인하기
# compose.yaml에 정의된 컨테이너 중 실행 중인 컨테이너만 보여준다.
$ docker compose ps


# compose.yaml에 정의된 모든 컨테이너를 보여준다.
$ docker compose ps -a

# 컨테이너를 실행하기 전에 이미지 재빌드하기
$ docker compose up --build # 포그라운드에서 실행
$ docker compose up --build -d # 백그라운드에서 실행

# Docker Compose에서 이용한 컨테이너 종료하기
$ docker compose down
```

5. Docker Compose

AppController.java



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

5. Docker Compose

application.yml



```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: root
    password: pwd1234
```


5. Docker Compose

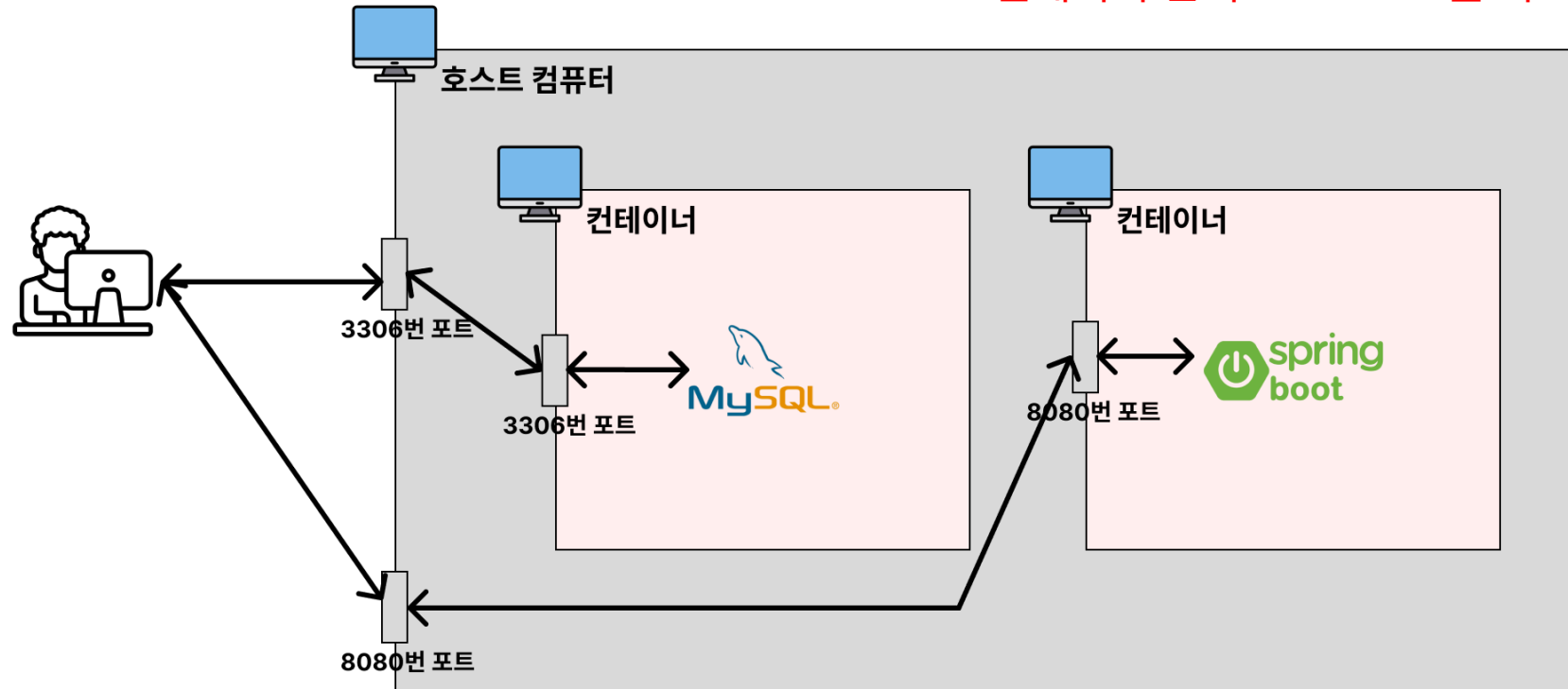
compose.yml

```
services:
  my-server:
    build: .
    ports:
      - 8080:8080
    # my-db의 컨테이너가 생성되고 healthy 하다고 판단 될 때, 해당 컨테이너를 생성한다.
    depends_on:
      my-db:
        condition: service_healthy
      my-cache-server:
        condition: service_healthy

  my-db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: pwd1234
      MYSQL_DATABASE: mydb # MySQL 최초 실행 시 mydb라는 데이터베이스를 생성해준다.
    volumes:
      - ./mysql_data:/var/lib/mysql
    ports:
      - 3306:3306
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping" ] # MySQL이 healthy 한 지 판단할 수 있는 명령어
      interval: 5s # 5초 간격으로 체크
      retries: 10 # 10번까지 재시도
```

5. Docker Compose

localhost:3306은 spring-boot
컨테이너 안의 3306 포트를 가르킴!



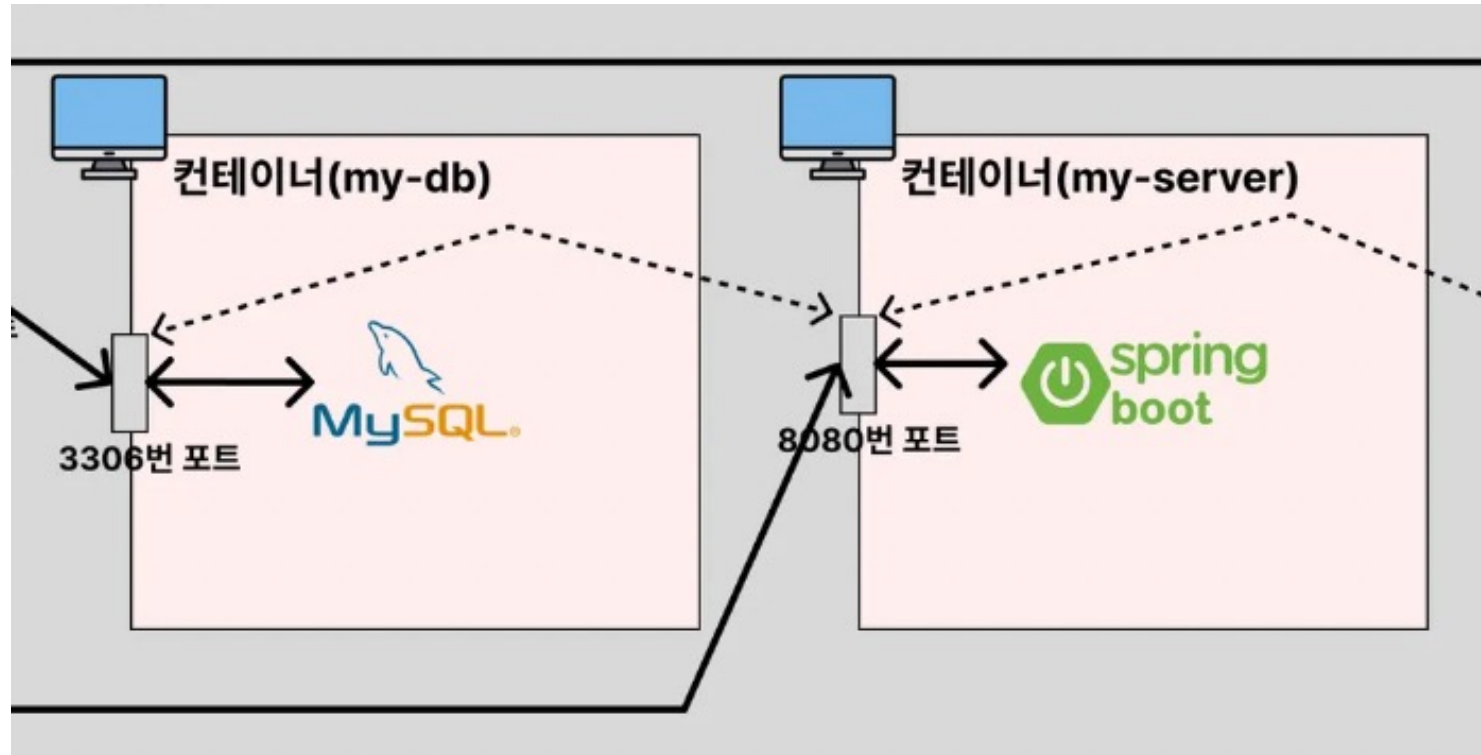
5. Docker Compose

application.yml



```
spring:
  datasource:
    url: jdbc:mysql://my-db:3306/mydb
    username: root
    password: pwd1234
```

5. Docker Compose



5. Docker Compose



```
$ ./gradlew clean build # 스프링 부트 프로젝트 빌드  
$ docker compose up --build -d # 도커 컴포즈 파일 빌드
```



localhost:8080



iCloud



Google



NAVER

Hello, World!

더 자세한 내용은 블로그에 업로드 예정...