

부제: 도커 뽀개기

BACKEND CHALLENGE

Docker : 나만의 도커 이미지 만들기 부터, 클라우드 배포까지!

2023.12.06 (수)

TIL & Blog

제 1 강

[1] <https://blog.naver.com/matrixpower1004/223284236714>

[2] <https://blog.naver.com/matrixpower1004/223284355732>

[3] <https://velog.io/@jeay123/원티드프리온보딩-백엔드챌린지-Week-1-1>

[4] <https://velog.io/@prettylee620/원티드-백엔드-챌린지-1주-1편-docker란-무엇인가-왜-쓰는가#-그렇다면-dockerfile과-docker-image란-뭘까>

[5] <https://velog.io/@enjoy89/Docker-도커-컨테이너-기술-제대로-이해하기>

나만의 도커 이미지 콘테스트



- 개발 / 일상생활에 필요했던 나만의 도커 이미지 파일을 정의하고 올려봐요
- 기한
 - 12/06 (수) 15:00 까지 제출
 - 12/06 (수) 수업시간에 시연
- 선정
 - 시연자 중 가장 많은 표를 받은 두분을 선정
 - 상품
 - 스타벅스 기프트 쿠폰



찾아보면 좋은 키워드



- 키워드 정리하기:
 - 도커 파일, 도커 이미지 레이어 구조
(공식 문서: <https://docs.docker.com/build/guide/layers/>)
- 기한
 - 12/06 (수) 15:00 까지 제출 (디스코드에 올려주시면 됩니다)
 - 12/06 (수) 수업시간에 간단히 설명
- 상품
 - 스타벅스 커피 쿠폰



커리큘럼

- [제 1 강] 컨테이너 기술에 대해서 알아보고, Docker의 기본 개념과 사용법에 대해 알아보자!
- [제 2 강] 로컬 환경에서 도커를 활용해보자!
- [제 3 강] 여러 개의 도커 컨테이너를 제어해보자!
- [제 4 강] 도커를 활용하는 클라우드 서비스에 대해 알아보자!

제1강 복습

- 가상화 기술
- 컨테이너 기술
- 도커 아키텍처
- 도커 CLI
- 도커 컨테이너의 라이프 사이클

Docker 란 무엇일까?

- 컨테이너 기반 가상화 도구
 - 리눅스 컨테이너 기술인 LXC(Linux Containers) 기반
- 애플리케이션을 컨테이너라는 단위로 격리하여 실행하고 배포하는 기술

가상화 (Virtualization) 기술이란 무엇일까?

하나의 물리적인 컴퓨터 자원(CPU, 메모리, 저장장치 등)을

가상적으로 분할하여 여러 개의 가상 컴퓨터 환경을 만들어 내는 기술

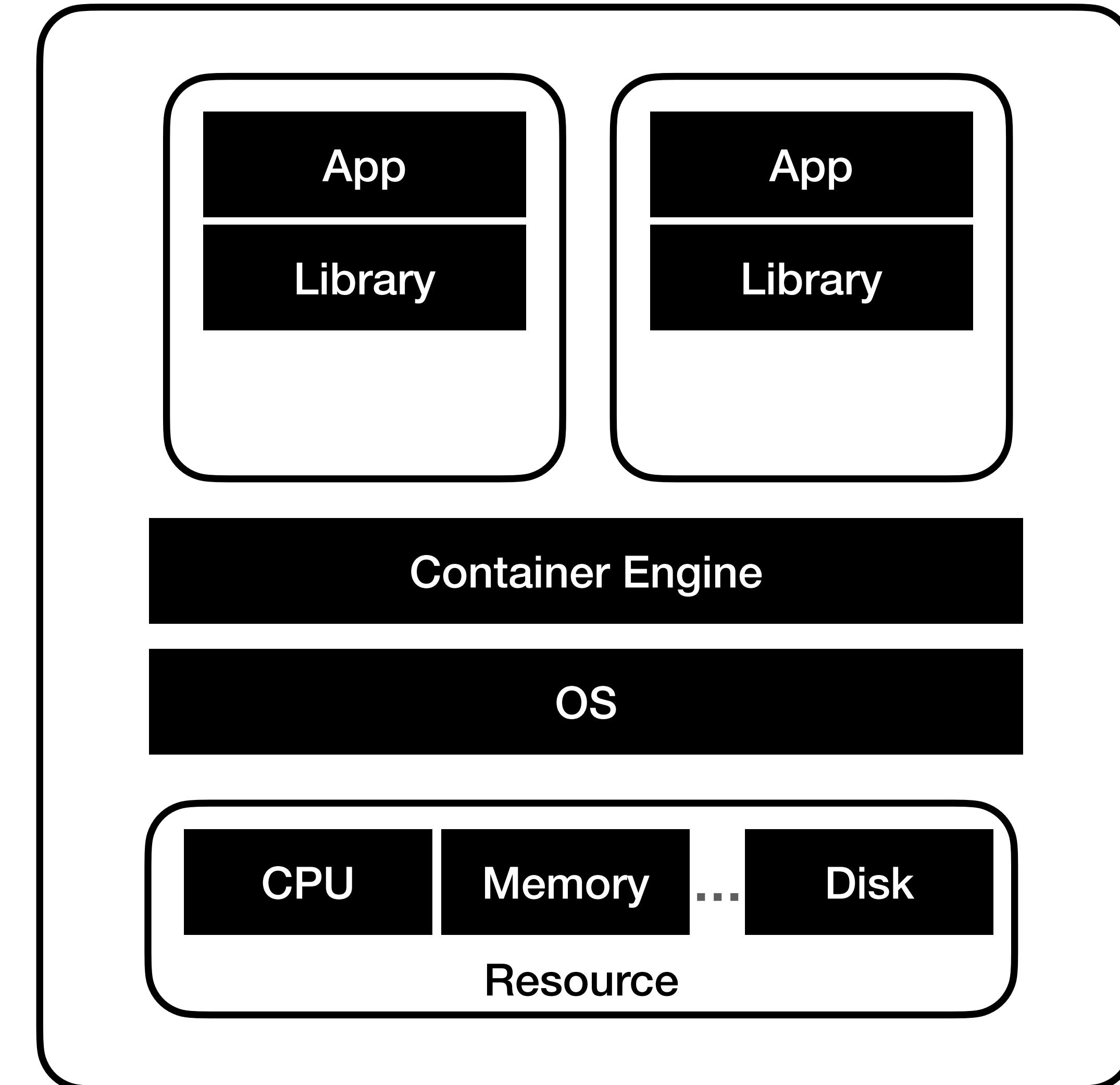
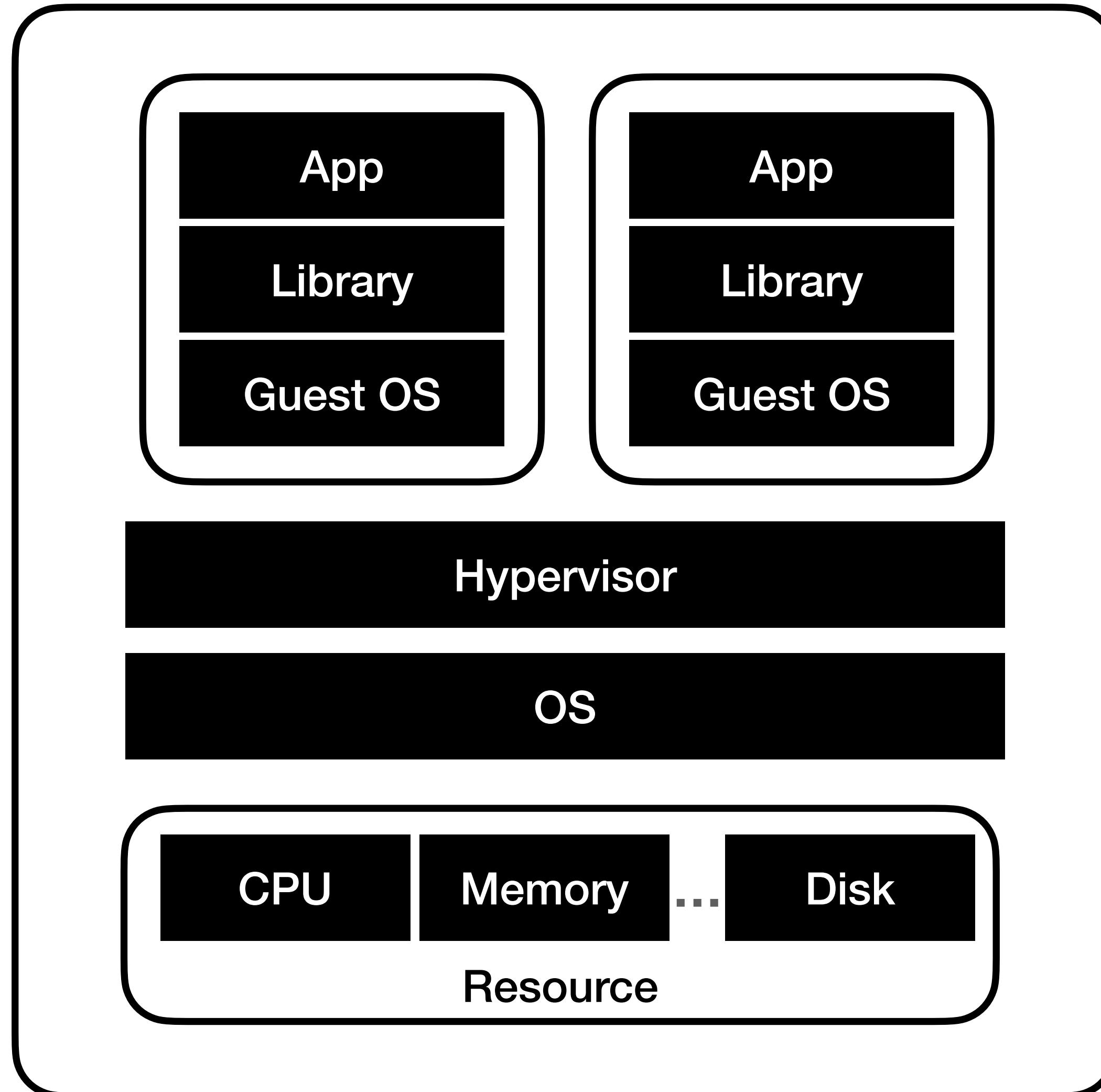
이를 통해 물리적인 컴퓨터 자원을 더욱 효율적으로 사용할 수 있으며,

서버나 애플리케이션 등을 운영하는데 있어 유연성과 안정성을 제공합니다.

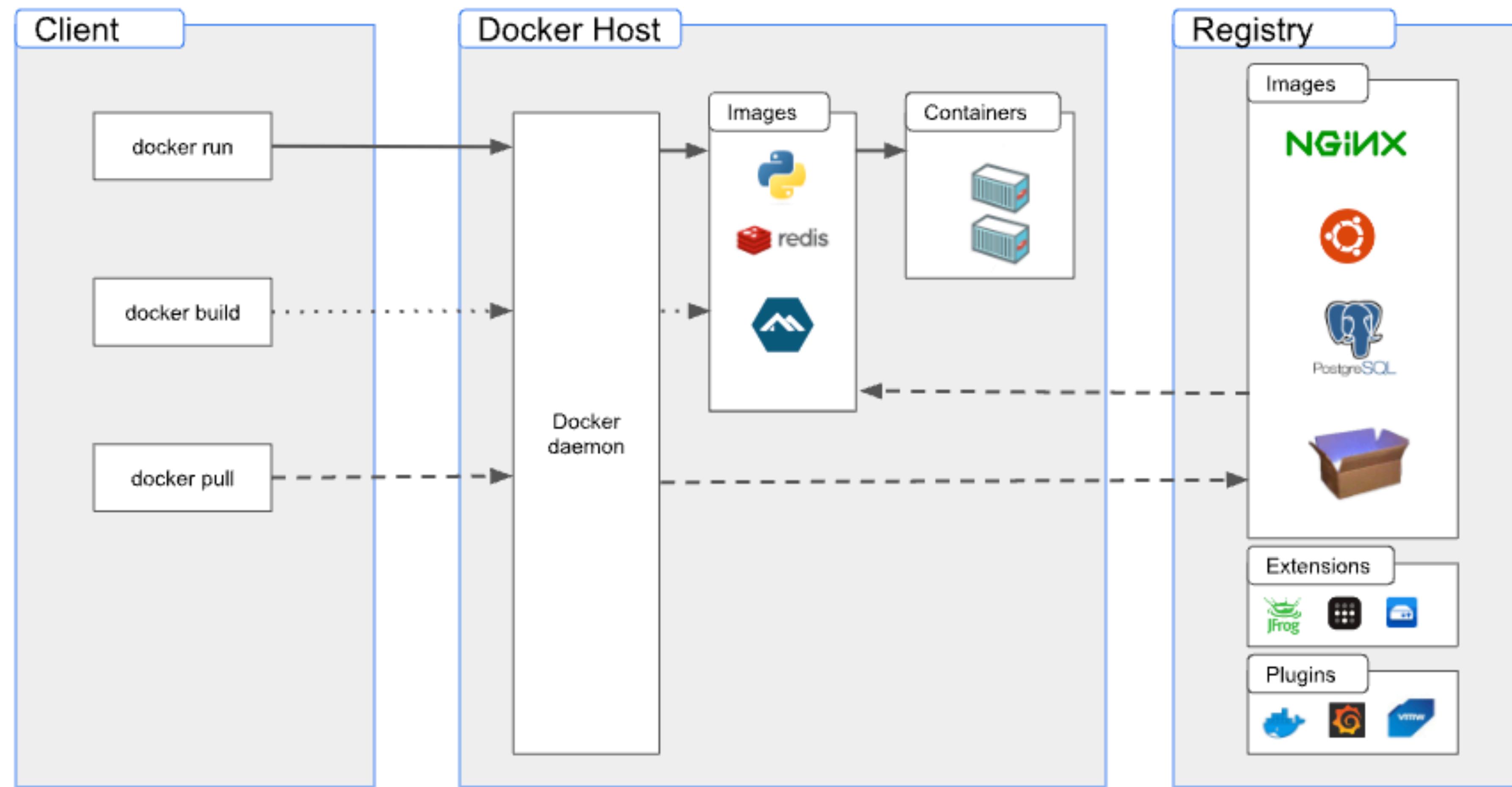
Container 란 무엇일까?

- 컨테이너는 가상화 기술 중 하나
- 호스트 운영체제 위에 여러 개의 격리된 환경을 생성
- 각각의 컨테이너 안에서 애플리케이션을 실행

Virtual Machine VS Container



Docker Architecture



Docker CLI

- **Download an image from a registry**
 - docker pull [OPTIONS] NAME[:TAG]@DIGEST
- **List images**
 - docker images [OPTIONS] [REPOSITORY[:TAG]]
- **Create and run a new container from an image**
 - docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
- **Stop one or more running containers**
 - docker stop [OPTIONS] CONTAINER [CONTAINER...]
- **Fetch the logs of a container**
 - docker logs [OPTIONS] CONTAINER
- **Remove one or more containers**
 - docker rm [OPTIONS] CONTAINER [CONTAINER...]
- **Remove one or more images**
 - docker rmi [OPTIONS] IMAGE [IMAGE...]

Docker CLI

3. Create and run a new container from an image

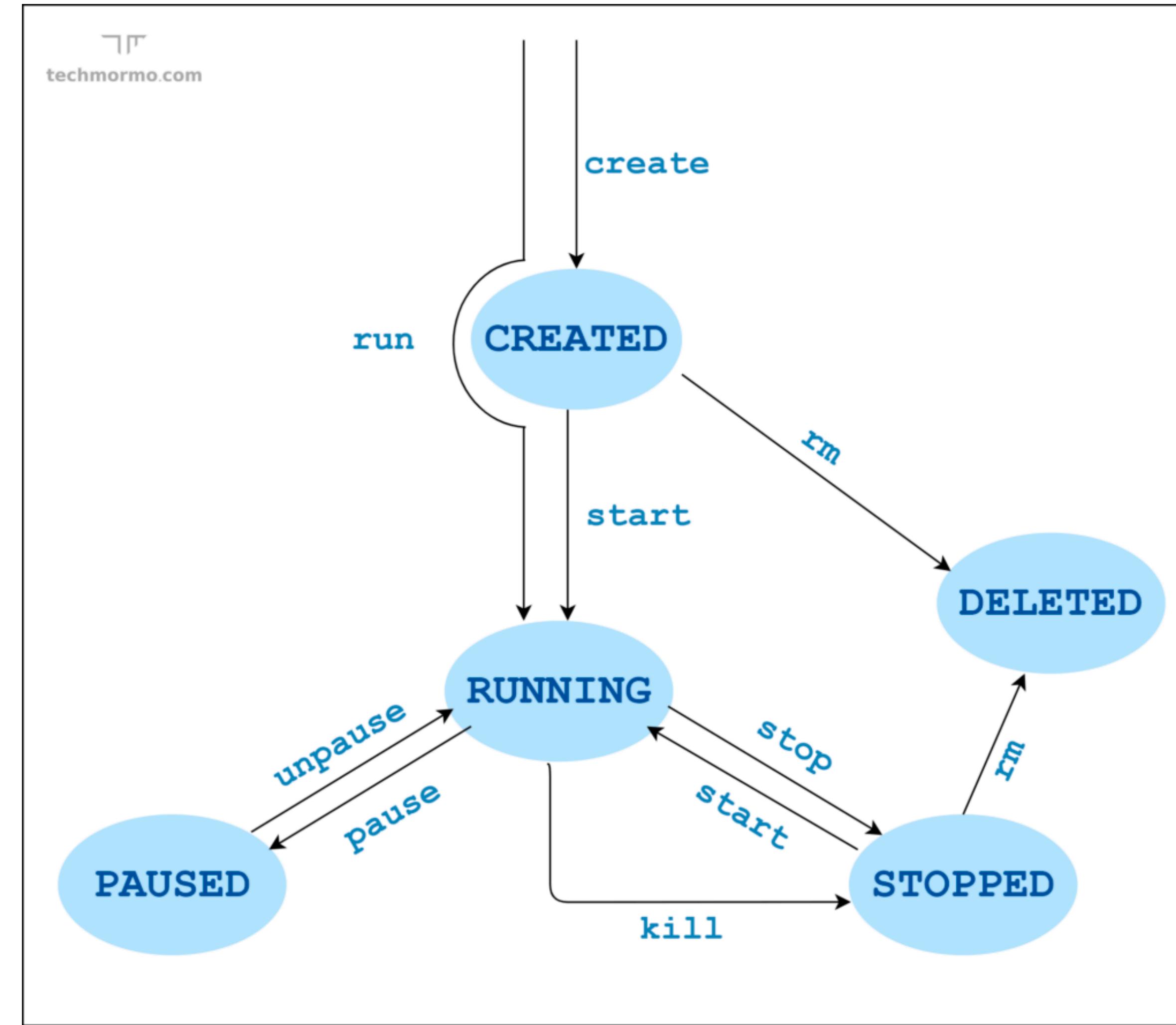
- 사용법

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

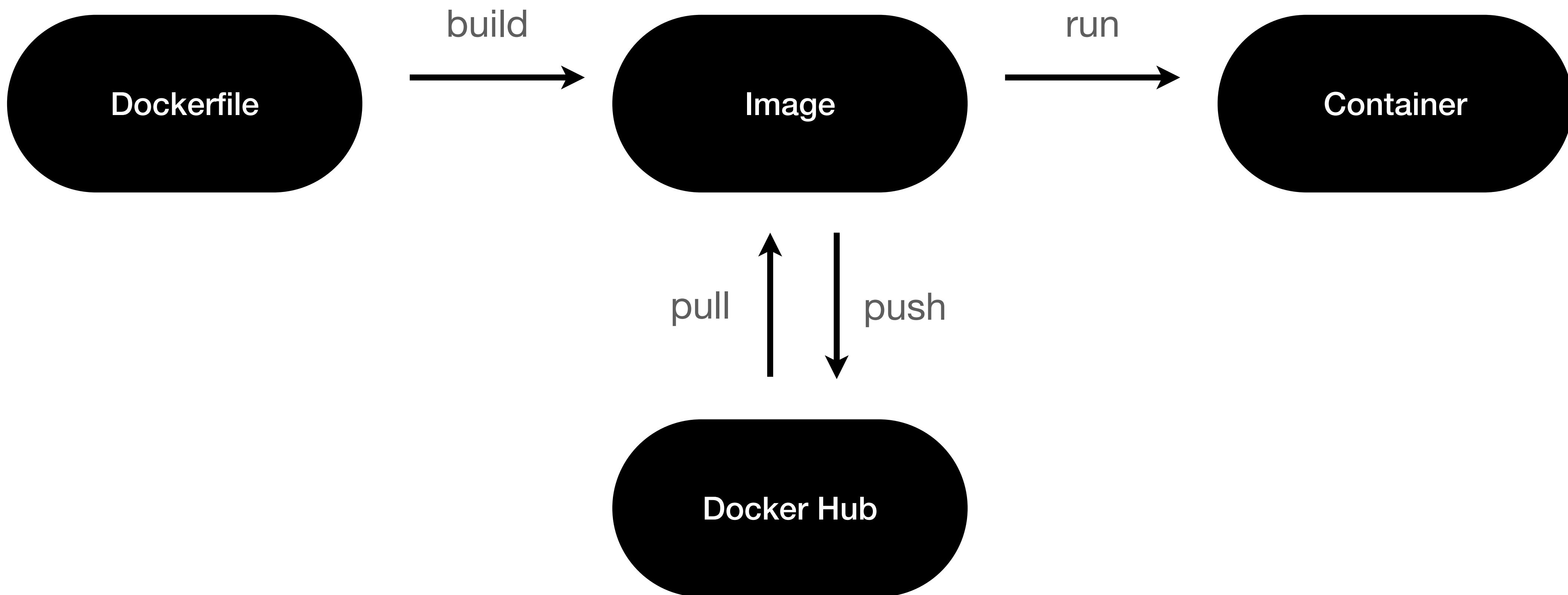
- 예제

```
docker run httpd
docker run --name secondContainer httpd
docker run -p 8888:80 -v ~/wanted/pre-mission/docker-pro-wanted/lecture-material:/usr/local/apache2/htdocs httpd
```

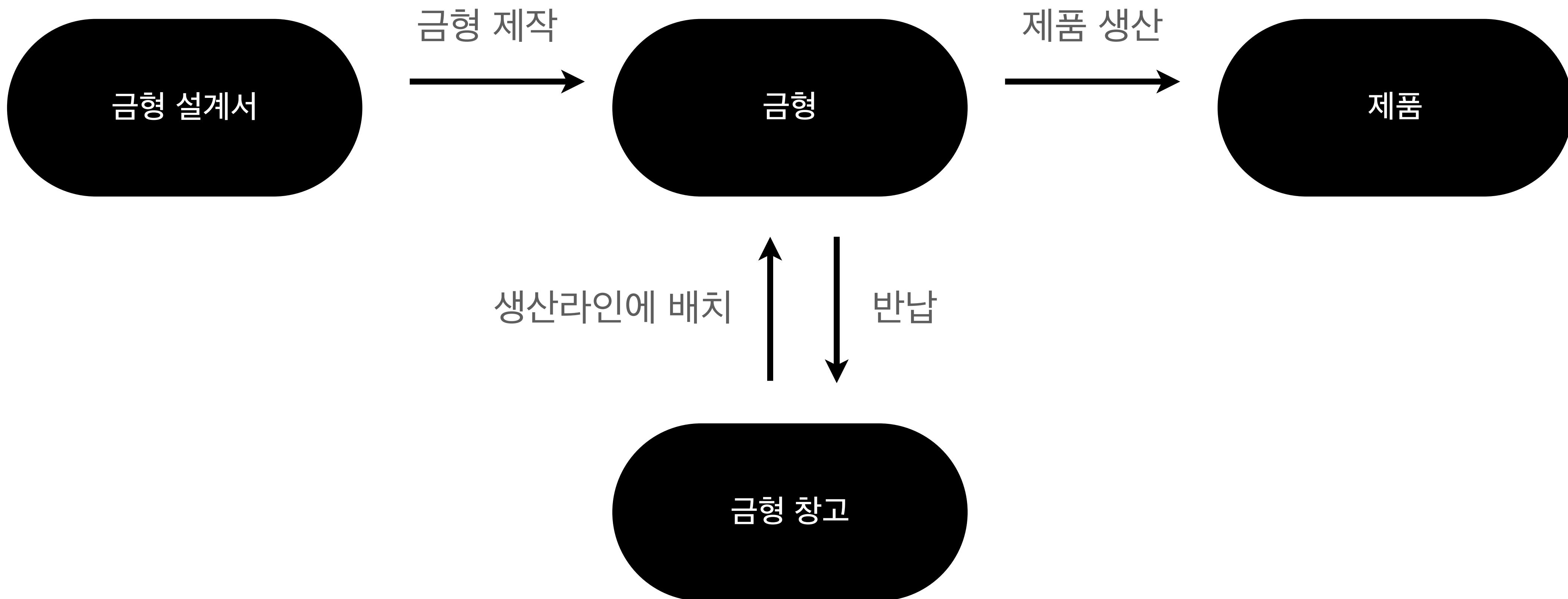
The Container Lifecycle



Dockerfile



Dockerfile



Dockerfile 활용

1. Dockerfile 예제

```
FROM httpd:latest
COPY index.html /usr/local/apache2/htdocs/index.html
EXPOSE 80
```

2. 이미지 만들기

```
docker build -t my-httpd .
```

3. 도커파일로 생성된 이미지로 컨테이너 실행하기

```
docker run -d -p 8888:80 my-httpd
```

[제 1 강 - 심화] 도커 파일, 도커 이미지, 도커 허브

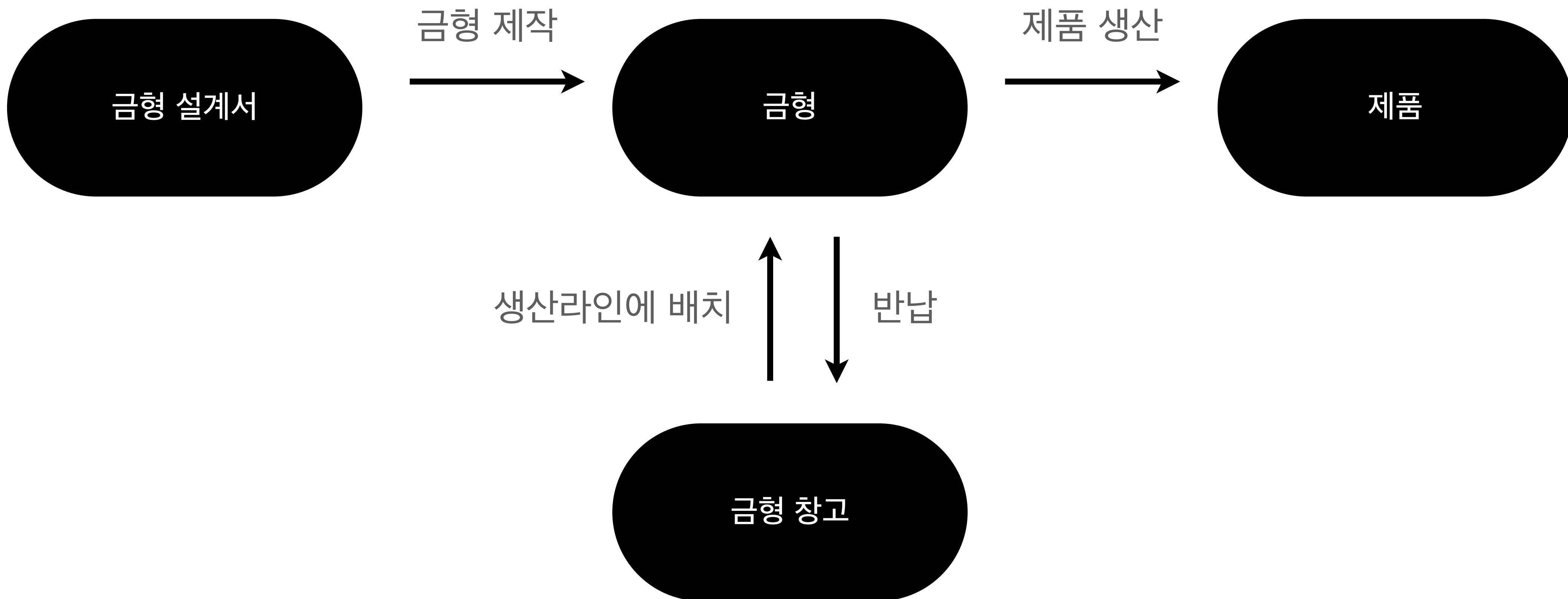
- 도커 이미지 빌드하기
- 도커 이미지를 도커 허브에 올리기
- 도커 네트워크

도커 이미지 빌드하기

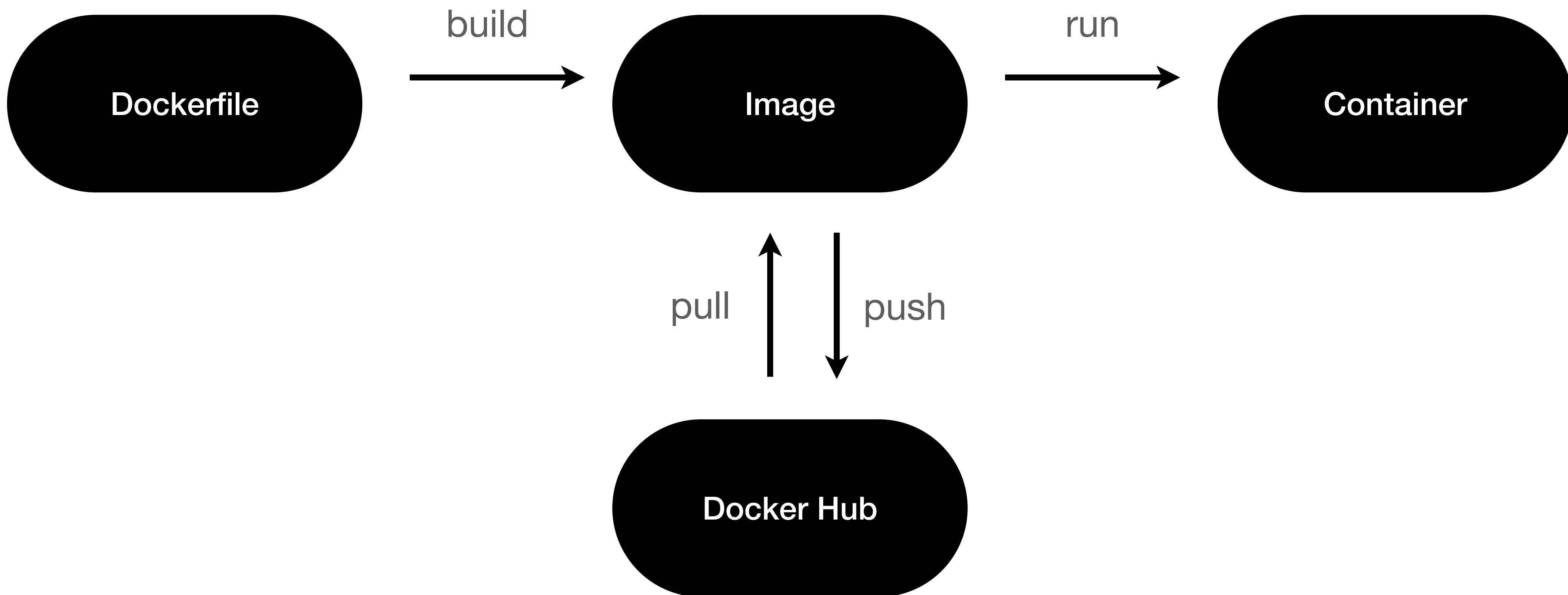
도커 파일이란?

- 도커 컨테이너를 빌드하고 설정하는 데 사용되는 텍스트 파일
- 도커 이미지를 어떻게 구성할지를 정의하는 파일
- 도커 이미지를 만들기 위한 명령어와 설정 정보를 담고 있음
 - 베이스 이미지
 - 빌드시 실행할 명령어
 - 복사할 파일
 - 환경 변수

Dockerfile



Dockerfile



도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **FROM:** 베이스 이미지 선정
 - **WORKDIR:** work directory 선정
 - **COPY:** 복사할 파일 선정 (예: 작업한 서비스 파일들)
 - **RUN:** (이미지 빌드 시)실행할 명령어
 - **CMD:** (컨테이너 생성 시) 실행할 명령어
 - **ENTRYPOINT:** (컨테이너가 생성되고 최초로) 실행할 명령어 (예: 서버 실행)

도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **FROM:** 베이스 이미지 선정
 - 반드시 있어야 하는 명령어
 - 여려겹의 이미지를 쌓아 가는데, 그 층 중 가장 기본이 되는 이미지

```
FROM alpine  
  
WORKDIR /usr/src/app  
  
COPY ./ ./  
  
CMD [ "echo", "hello" ]
```

도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **WORKDIR**: work directory 설정
 - 도커파일 뒤에 오는 모든 지시자(RUN, CMD, COPY, ADD 등)에 대한 작업 디렉토리를 설정
 - 리눅스 명령어의 cd와 비슷한 역할

```
FROM alpine

WORKDIR /usr/src/app

COPY ./ .

CMD [ "echo", "hello" ]
```

도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **COPY**: 복사할 파일 선정 (예: 작업한 서비스 파일들)
 - 현재 로컬 컴퓨터에 있는 파일을 도커 컨테이너 안으로 복사하는 기능

```
FROM alpine  
  
WORKDIR /usr/src/app  
  
COPY ./ ./  
  
CMD [ "echo", "hello" ]
```

도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **RUN:** (이미지 빌드 시)실행할 명령어
 - 현재 로컬 컴퓨터에 있는 파일을 도커 컨테이너 안으로 복사하는 기능

```
FROM python:3.7.9  
WORKDIR /usr/src/app  
COPY ./ ./  
RUN pip install -r requirements
```

도커 이미지 빌드하기

Dockerfile syntax

- 공식 문서
 - **CMD:** (컨테이너 생성 시) 실행할 명령어
 - CMD 명령어는 RUN 명령어가 이미지를 빌드할 때 실행되는 것과 달리,
 - 이미지로부터 컨테이너를 생성하여 최초로 실행할 때 수행됨

```
FROM ubuntu:16.04  
  
CMD ["echo", "CMD test"]
```

```
docker build . -f dockerfile_cmd  
  
docker run cmd_test
```

도커 이미지 빌드하기

Dockerfile syntax

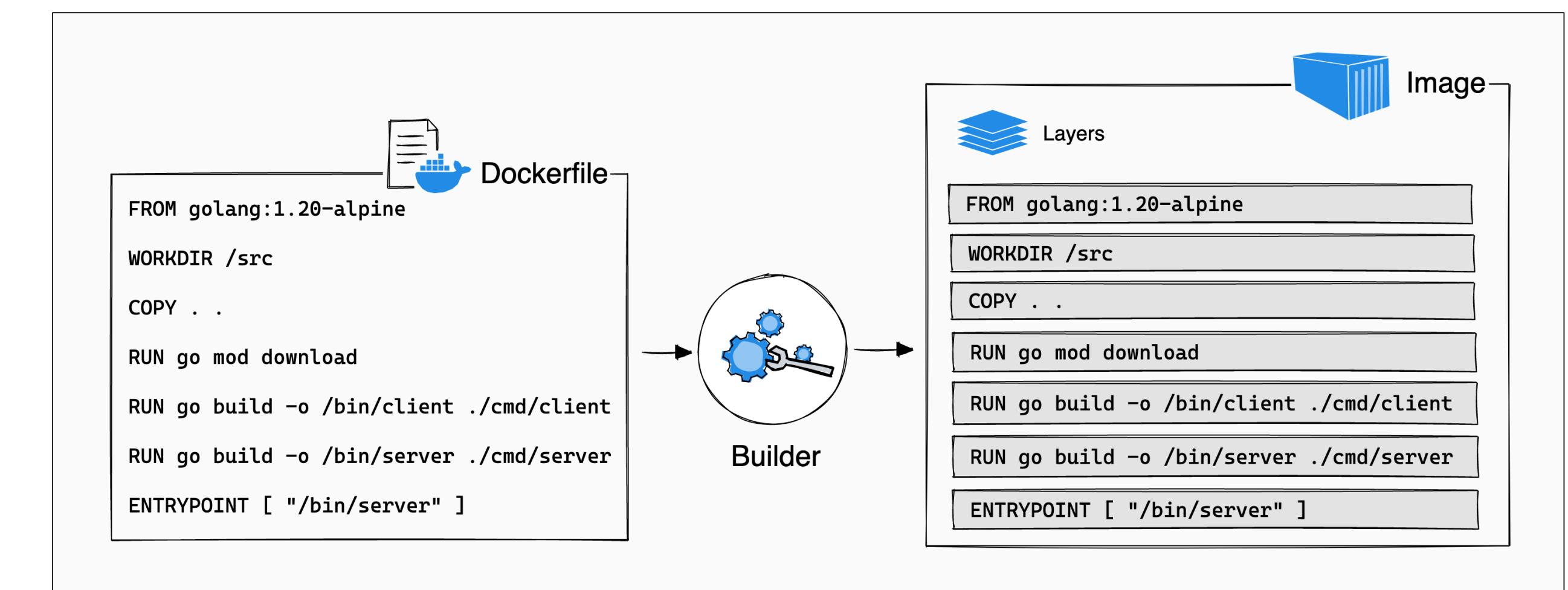
- 공식 문서
 - **ENTRYPOINT**: (컨테이너가 생성되고 최초로) 실행할 명령어 (예: 서버 실행) 현재 로컬 컴퓨터에 있는 파일을 도커 컨테이너 안으로 복사하는 기능

```
FROM alpine  
  
WORKDIR /usr/src/app  
  
COPY ./ ./  
  
CMD [ "echo", "hello" ]
```

도커 이미지 빌드하기

Layers in image

- Dockerfile을 빌드하여 Docker 이미지를 생성하면 Dockerfile의 명령별로 이미지를 생성한다.
- 작성된 여러 개의 이미지는 레이어 구조로 되어 있음.



[리스트 5.16] 이미지의 레이어 구조

```
$ docker build -t webap .
Sending build context to Docker daemon 1.307 GB
Step 1/4 : FROM ubuntu:latest
--> 0458a4468cbc ← 첫 번째 이미지 작성
Step 2/4 : RUN apt-get update && apt-get install -y -q nginx
~중략~
--> 9e6b06ec7d4f ← 두 번째 이미지 작성
Step 3/4 : COPY index.html /usr/share/nginx/html/
~중략~
--> d66d95434d9b ← 세 번째 이미지 작성
Step 4/4 : CMD nginx -g daemon off;
~중략~
--> a6f8f57f1588 ← 네 번째 이미지 작성
Successfully built a6f8f57f1588
Successfully tagged webap:latest
```

#STEP:1 Ubuntu (베이스 이미지)

FROM ubuntu: latest

STEP:2 Nginx 설치

RUN apt-get update && apt-get install -y -q

nginx

STEP : 3 파일 복사

COPY index.html /us/ share/nginx/html/

STEP:4 Nginx 시작

CMD ["nginx", "-g", "daemon off;"]

로그를 확인하면 Dockerfile의 명령 한 줄마다 이미지가 작성되는 것을 알 수 있습니다 (그림 5.5). 예를 들어 베이스 이미지를 Ubuntu의 latest로 지정하고 있는 FROM 명령을 실행하면 이미지 ID가 0458a4468cbc인 이미지가 생성됩니다.

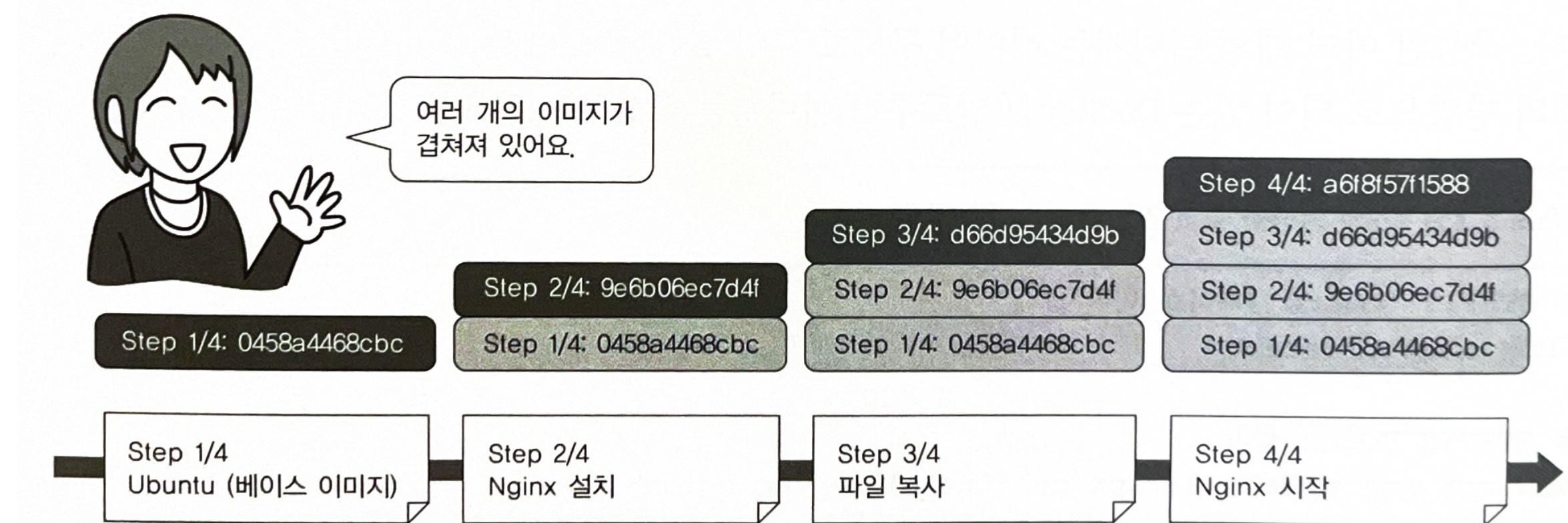


그림 5.5 Docker 이미지의 레이어 구조

도커 이미지를 도커 허브에 올리기

도커 이미지 만들기

- 이미지 태그 설정
 - Docker Hub에 이미지를 등록하려면 아래와 같은 규칙을 준수해야한다:
 - [Docker Hub 사용자명]/이미지명:[태그명]
 - 태그 방법
 - build 시: docker build -t my-httpd .
 - build 후: docker image tag [image name]

도커 이미지를 도커 허브에 올리기

도커 허브에 로그인하기

13. Log in to a registry

- 사용법

```
docker login [OPTIONS] [SERVER]
```



- 예제

```
docker login
```



14. Log out from a registry

- 사용법

```
docker logout [SERVER]
```



- 예제

```
docker logout
```



도커 이미지를 도커 허브에 올리기

도커 이미지 올리기

12. Upload an image to a registry

- 사용법

```
docker push [OPTIONS] NAME[:TAG]
```

- 예제

```
docker push drumgrammer/my-httpd:latest
```

<https://github.com/drum-grammer/docker-pro-2308/blob/main/lecture/1st/cli.md>

도커 네트워크

네트워크 드라이버

- docker network ls

```
→ docker-pro-wanted git:(main) ✘ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
1a27d994a2ff   bridge    bridge      local
f5ba5e642c94   host      host       local
98a646ac2502   none     null       local
```

- Network Drivers [[공식 문서](#)]

- bridge: 기본 네트워크 드라이버, 동일한 도커 호스트에서 컨테이너 간의 통신을 도와줌.
- host: 호스트의 네트워크를 직접 사용.
- overlay: 서로 다른 도커 호스트의 컨테이너 간 통신을 도와줌

도커 네트워크

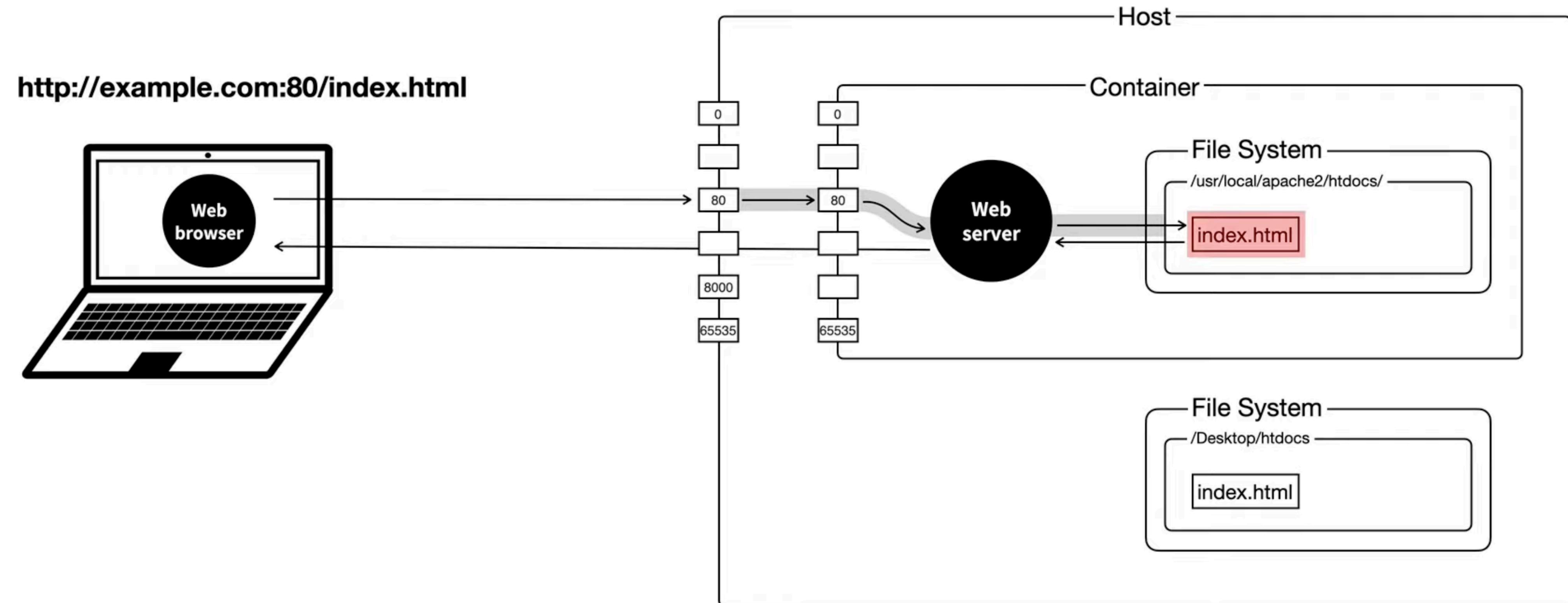
네트워크 드라이버

- docker network inspect bridge

```
"Containers": {  
    "275e19a4f245d59354bd399ee8f57577ba0e5ae9ddc082ca43e6159837d08aaf": {  
        "Name": "elated_khorana",  
        "EndpointID": "ac2a9913d7eb20728975f7953257881817eea602e87e0c59722e2d88a3b292a7",  
        "MacAddress": "02:42:ac:11:00:03",  
        "IPv4Address": "172.17.0.3/16",  
        "IPv6Address": ""  
    },  
    "321e578211885a6b4448350271fdf976d16449c124e37330dda50ab5c430a97c": {  
        "Name": "keen_jemison",  
        "EndpointID": "13da1d33320c7c3f9e2aa9209faee2dc704bb6f112c152976d11aaef19d55e96",  
        "MacAddress": "02:42:ac:11:00:02",  
        "IPv4Address": "172.17.0.2/16",  
        "IPv6Address": ""  
    },  
},
```

- 연결된 컨테이너 확인

Port Forwarding, Volume Mount





생각해보기

여러개의 컨테이너를 제어해야 한다면?

```
docker network create wordpress_net
```

```
docker \
run \
--name "db" \
-v "$(pwd)/db_data:/var/lib/mysql" \
-e "MYSQL_ROOT_PASSWORD=root_pass" \
-e "MYSQL_DATABASE=wordpress" \
-e "MYSQL_USER=docker_pro" \
-e "MYSQL_PASSWORD=docker_pro_pass" \
--network wordpress_net \
mysql:latest
```

```
docker \
run \
--name app \
-v "$(pwd)/app_data:/var/www/html" \
-e "WORDPRESS_DB_HOST=db" \
-e "WORDPRESS_DB_NAME=wordpress" \
-e "WORDPRESS_DB_USER=docker_pro" \
-e "WORDPRESS_DB_PASSWORD=docker_pro_pass" \
-e "WORDPRESS_DEBUG=1" \
-p 8000:80 \
--network wordpress_net \
wordpress:latest
```

[제 2 강] 로컬 환경에서 도커를 활용해보자!

- 도커 컴포즈란 무엇인가?
- 도커 컴포즈 설정 파일
- 도커 컴포즈를 활용하여 워드 프레스 서비스 실행하기

도커 컴포즈란?

공식 문서 확인

Screenshot of the Docker Docs website showing the "Compose specification / Overview" page.

The page includes a sidebar with navigation links for Reference documentation, Command-line reference, API reference, Dockerfile reference, Compose file reference, and Compose specification. The "Overview" link under Compose specification is currently selected.

Overview

Important

From the end of June 2023 Compose V1 won't be supported anymore and will be removed from all Docker Desktop versions.

Make sure you switch to [Compose V2](#) with the `docker compose` CLI plugin or by activating the **Use Docker Compose V2** setting in Docker Desktop.

For more information, see the [Evolution of Compose](#).

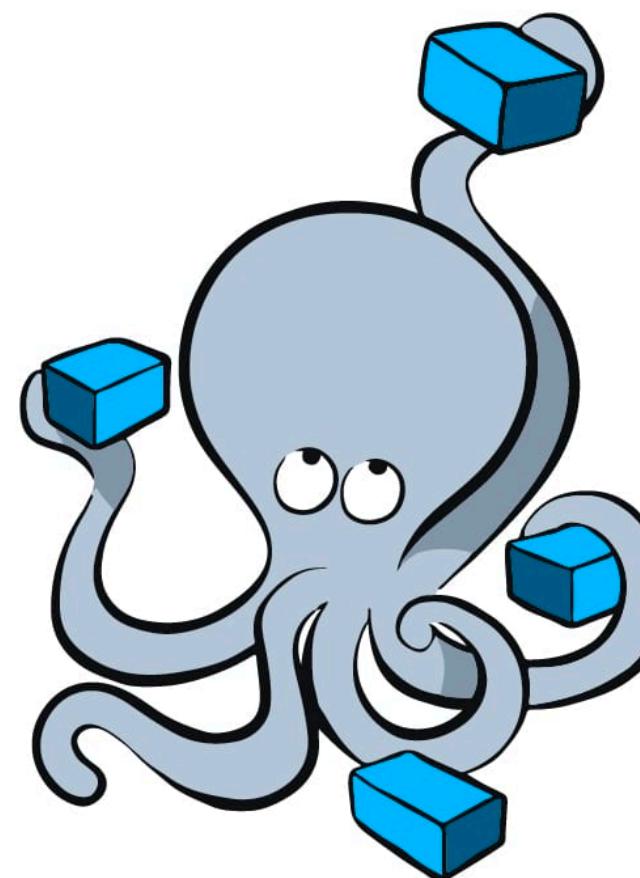
The Compose file is a [YAML](#) file defining services, networks, and volumes for a Docker application. The latest and recommended version of the Compose file format is defined by the [Compose Specification](#). The Compose spec merges the legacy 2.x and 3.x versions, aggregating properties across these formats and is implemented by [Compose 1.27.0+](#).

Use the links below to easily navigate key sections of the Compose specification.

- Status of the Specification** (Icon: Checkmark)
- The Compose application model** (Icon: Server)
- The Compose file** (Icon: Document)
- Version and name top-level element** (Icon: Document)
- Services top-level element** (Icon: Server)
- Networks top-level element** (Icon: Network)
- Volumes top-level element** (Icon: Volume)
- Configs top-level element** (Icon: Configuration)
- Secrets top-level element** (Icon: Secret)
- Fragments** (Icon: Document)
- Extensions** (Icon: Extension)
- Interpolation** (Icon: Interpolation)
- Compose file build** (Icon: Build)
- Compose file deploy** (Icon: Deploy)

도커 컴포즈란?

- 도커 컨테이너를 일괄적으로 정의하고 제어하는 도구
- 설정 파일을 도커 CLI로 번역하는 역할



docker
Compose

```
version: '3.1'

services:

  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html
```

、

사용 예제

- 회사에서 개발하고 있는 백엔드 서버에 필요한 인프라
 - Database
 - Redis
 - rabbitMQ
- 설정 파일을 도커 CLI로 번역하는 역할

도커 컴포즈를 활용한 예제

<https://github.com/drum-grammer/docker-pro-2308/blob/main/lecture/2nd/local-infra.yml>

33 lines (31 sloc) | 725 Bytes

```
1  version: '3.0'
2
3  services:
4    mariadb10:
5      image: mariadb:10
6      ports:
7        - "3310:3306/tcp"
8      environment:
9        - MYSQL_ROOT_PASSWORD=my_db_password
10       - MYSQL_USER=docker_pro
11       - MYSQL_PASSWORD=docker_pro_pass
12       - MYSQL_DATABASE=docker_pro
13    redis:
14      image: redis
15      command: redis-server --port 6379
16      restart: always
17      ports:
18        - 6379:6379
19    rabbitmq:
20      image: rabbitmq:3-management-alpine
21      container_name: 'rabbitmq'
22      ports:
23        - 5672:5672
24        - 15672:15672
25      volumes:
26        - ~/.docker-conf/rabbitmq/data/:/var/lib/rabbitmq/
27        - ~/.docker-conf/rabbitmq/log/:/var/log/rabbitmq
28      networks:
29        - rabbitmq_go_net
30
31    networks:
32      rabbitmq_go_net:
33        driver: bridge
```

```
→ docker-pro-wanted git:(main) docker-compose -f local-infra.yml up --build
[+] Running 5/5
  Network docker-pro-wanted_rabbitmq_go_net    Created      0.0s
  Network docker-pro-wanted_default           Created      0.0s
  Container rabbitmq                         Created      0.1s
  Container docker-pro-wanted-mariadb10-1     Created      0.1s
  Container docker-pro-wanted-redis-1         Created      0.0s
Attaching to docker-pro-wanted-mariadb10-1, docker-pro-wanted-redis-1, rabbitmq
docker-pro-wanted-mariadb10-1  | 2023-04-06 08:45:32+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1
:10.6.5+maria~focal started.
docker-pro-wanted-redis-1    | 1:C 06 Apr 2023 08:45:32.586 # o000o000o000o Redis is starting o000o000o000o
docker-pro-wanted-redis-1    | 1:C 06 Apr 2023 08:45:32.586 # Redis version=6.2.6, bits=64, commit=00000000, modifie
d=0, pid=1, just started
docker-pro-wanted-redis-1    | 1:C 06 Apr 2023 08:45:32.586 # Configuration loaded
docker-pro-wanted-redis-1    | 1:M 06 Apr 2023 08:45:32.587 * monotonic clock: POSIX clock_gettime
docker-pro-wanted-redis-1    | 1:M 06 Apr 2023 08:45:32.587 * Running mode=standalone, port=6379.
docker-pro-wanted-redis-1    | 1:M 06 Apr 2023 08:45:32.587 # Server initialized
docker-pro-wanted-redis-1    | 1:M 06 Apr 2023 08:45:32.589 * Ready to accept connections
docker-pro-wanted-mariadb10-1 | 2023-04-06 08:45:32+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
docker-pro-wanted-mariadb10-1 | 2023-04-06 08:45:32+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1
:10.6.5+maria~focal started.
docker-pro-wanted-mariadb10-1 | 2023-04-06 08:45:32+00:00 [Note] [Entrypoint]: Initializing database files
docker-pro-wanted-mariadb10-1 | 2023-04-06 08:45:32+00:00 [Warning] You need to use --log-bin to make --expire-logs-days
or --binlog-expire-logs-seconds work.
docker-pro-wanted-mariadb10-1 |
docker-pro-wanted-mariadb10-1 |
docker-pro-wanted-mariadb10-1 | PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !
docker-pro-wanted-mariadb10-1 | To do so, start the server, then issue the following command:
docker-pro-wanted-mariadb10-1 | '/usr/bin/mysql_secure_installation'
docker-pro-wanted-mariadb10-1 |
docker-pro-wanted-mariadb10-1 | which will also give you the option of removing the test
docker-pro-wanted-mariadb10-1 | databases and anonymous user created by default. This is
docker-pro-wanted-mariadb10-1 | strongly recommended for production servers.
docker-pro-wanted-mariadb10-1 |
docker-pro-wanted-mariadb10-1 | See the MariaDB Knowledgebase at https://mariadb.com/kb or the
docker-pro-wanted-mariadb10-1 | MySQL manual for more instructions.
```

도커 컴포즈 파일 구성

공식 문서

The Compose file

The Compose file is a [YAML](#) file defining:

- [Version](#) (Optional)
- [Services](#) (Required)
- [Networks](#)
- [Volumes](#)
- [Configs](#)
- [Secrets](#)

도커 컴포즈 파일 구성

- **version**
- **services**
- **network**
- **volume**
- **config**
- **secret**

```
version: '3.1'

services:
  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html
```

도커 컴포즈 파일 구성

<https://docs.docker.com/compose/compose-file/compose-versioning/>

- **version**

The screenshot shows the Docker Docs website with a blue header bar containing the Docker logo, 'docker docs', a search bar, and navigation links for 'Home', 'Guides', 'Manuals', 'Reference', 'Samples', and 'Contribute'. Below the header, a breadcrumb trail indicates the current page: Home / Reference / Compose file reference / Legacy versions / About versions and upgrading. On the left, a sidebar menu lists various documentation sections, with 'About versions and upgrading' highlighted by a blue border. The main content area features a large heading 'Compose file versions and upgrading'. Below it, a paragraph explains that the Compose file is a YAML file defining services, networks, and volumes for a Docker application. Another paragraph states that the Compose file formats are described in references specific to each version. A table then details the changes for different versions:

Reference file	What changed in this version
Compose Specification (most current, and recommended)	Versioning
Version 3	Version 3 updates
Version 2	Version 2 updates
Version 1 (Deprecated)	Version 1 updates

At the bottom, a note states: 'The topics below explain the differences among the versions, Docker Engine compatibility, and how to upgrade.'

도커 컴포즈 파일 구성

- **services**

- 실행하려는 컨테이너들을 정의하는 역할
- 이름, 이미지, 포트 매팅, 환경 변수, 볼륨 등을 포함
- 해당 정보를 가지고 컨테이너를 생성하고 관리

```
version: '3.1'

services:
  wordpress:
    image: wordpress
    restart: always
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD: examplepass
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress:/var/www/html
```

도커 컴포즈 파일 구성

- **services**
 - **image**: 컨테이너를 생성할 때 쓰일 이미지 지정
 - **build**: 정의된 도커파일에서 이미지를 빌드해 서비스의 컨테이너를 생성하도록 설정
 - **environment**: 환경 변수 설정, docker run 명령어의 --env, -e 옵션과 동일
 - **command**: 컨테이너가 실행될 때 수행할 명령어, docker run 명령어의 마지막에 붙는 커맨드와 동일
 - **depends_on**: 컨테이너 간의 의존성 주입, 명시된 컨테이너가 먼저 생성되고 실행

도커 컴포즈 파일 구성

- **services**
 - **ports**: 개방할 포트 지정, docker run 명령어의 -p와 동일
 - **expose**: 링크로 연계된 컨테이너에게만 공개할 포트 설정
 - **volumes**: 컨테이너에 볼륨을 마운트함
 - **restart**: 컨테이너가 종료될 때 재시작 정책
 - no: 재시작 되지 않음
 - always: 외부에 영향에 의해 종료 되었을 때 항상 재시작 (수동으로 끄기 전까지)
 - on-failure: 오류가 있을 시에 재시작

도커 컴포즈 명령어

docker-compose vs docker compose ?

docker-compose 명령어가 docker compose로 흡수되었음.

이전에는 Docker에서는 docker-compose 명령어가 별도로 설치되어야 했지만,

Docker 1.13 이후로는 docker-compose 명령어가 Docker CLI에 통합됨

도커 컴포즈 명령어

- **docker-compose -f local-infra.yml up -d**
 - **up**: 도커 컴포즈 파일로, 컨테이너를 생성하기
 - **-f**: 도커 컴포즈 파일 지정하기
 - **-d**: 백그라운드에서 실행하기

도커 컴포즈 명령어

공식문서

docker compose up

Create and start containers

Usage

```
$ docker compose up [OPTIONS] [SERVICE...]
```



Refer to the [options section](#) for an overview of available `OPTIONS` for this command.

도커 컴포즈 명령어

- **docker-compose -f local-infra.yml up -d**
 - **up**: 도커 컴포즈 파일로, 컨테이너를 생성하기
 - **-f**: 도커 컴포즈 파일 지정하기
 - **-d**: 백그라운드에서 실행하기

도커 컴포즈 실습

도커 CLI로 여러개 컨테이너 관리하기

도커 네트워크 리스트 조회

docker network ls

- **bridge**: 도커 엔진에 의해 자동으로 생성되는 가상 네트워크. 컨테이너끼리 연결되는 기본 네트워크
- **host**: 호스트 컴퓨터의 네트워크 인터페이스를 그대로 사용하는 네트워크
- **none**: 네트워크를 사용하지 않는 컨테이너

NETWORK ID	NAME	DRIVER	SCOPE
0e35967d48fd	bridge	bridge	local
f5ba5e642c94	host	host	local
98a646ac2502	none	null	local

도커 컴포즈 실습

도커 CLI로 여러개 컨테이너 관리하기

도커 네트워크 생성

docker network create wordpress_net

도커 네트워크를 생성하는 명령어 (네트워크의 이름을 지정)

도커 컴포즈 실습

도커 CLI로 여러개 컨테이너 관리하기

```
# mysql db container 생성
```

```
docker \
```

```
run \
```

```
--name "db" \
```

```
-v "$(pwd)/db_data:/var/lib/mysql" \
```

```
-e "MYSQL_ROOT_PASSWORD=root_pass" \
```

```
-e "MYSQL_DATABASE=wordpress" \
```

```
-e "MYSQL_USER=docker_pro" \
```

```
-e "MYSQL_PASSWORD=docker_pro_pass" \
```

```
--network wordpress_net \
```

```
mysql:latest
```

도커 컴포즈 실습

도커 CLI로 여러개 컨테이너 관리하기

```
# wordpress container 생성
```

```
docker \
```

```
run \
```

```
--name app \
```

```
-v "$(pwd)/app_data:/var/www/html" \
```

```
-e "WORDPRESS_DB_HOST=db" \
```

```
-e "WORDPRESS_DB_NAME=wordpress" \
```

```
-e "WORDPRESS_DB_USER=docker_pro" \
```

```
-e "WORDPRESS_DB_PASSWORD=docker_pro_pass" \
```

```
-e "WORDPRESS_DEBUG=1" \
```

```
-p 8000:80 \
```

```
--network wordpress_net \
```

```
wordpress:latest
```

도커 컴포즈 실습

도커 컴포즈로 여러개 컨테이너 관리하기

```
version: "3.0"
```

```
services:
```

```
db:
```

```
  image: mysql:latest
```

```
  volumes:
```

```
    - ./db_data:/var/lib/mysql
```

```
  restart: always
```

```
  environment:
```

```
    MYSQL_ROOT_PASSWORD: root_pass
```

```
    MYSQL_DATABASE: wordpress
```

```
    MYSQL_USER: docker_pro
```

```
    MYSQL_PASSWORD: docker_pro_pass
```

```
app:
```

```
depends_on:
```

```
  - db
```

```
image: wordpress:latest
```

```
volumes:
```

```
  - ./app_data:/var/www/html
```

```
ports:
```

```
  - "8000:80"
```

```
restart: always
```

```
environment:
```

```
    WORDPRESS_DB_HOST: db:3306
```

```
    WORDPRESS_DB_NAME: wordpress
```

```
    WORDPRESS_DB_USER: docker_pro
```

```
    WORDPRESS_DB_PASSWORD: docker_pro_pass
```

도커 컴포즈 실습

도커 컴포즈로 여러개 컨테이너 관리하기

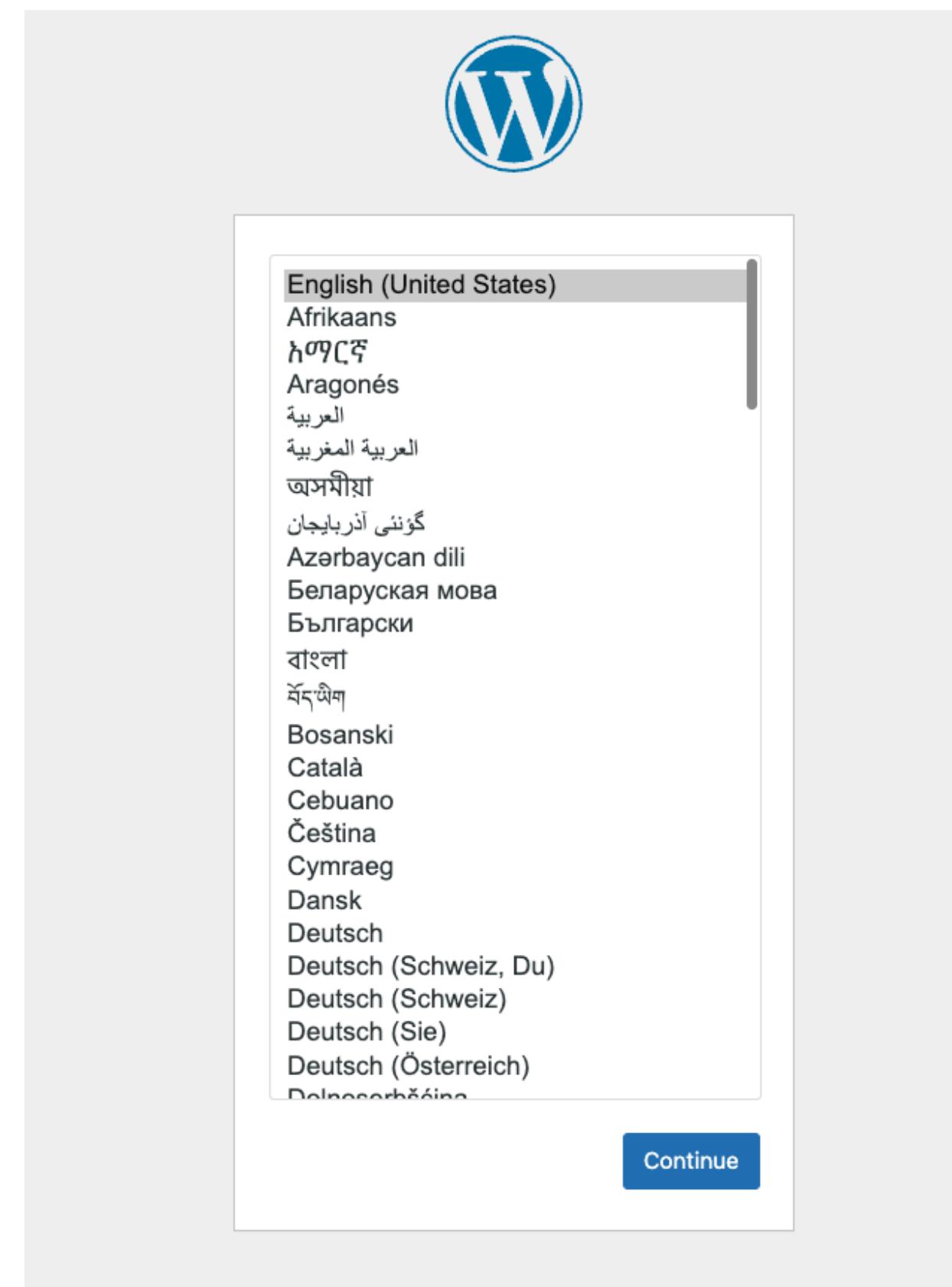
docker compose -f docker-compose.yml up --build

```
→ second git:(main) ✘ docker-compose -f docker-compose.yml up --build
[+] Running 2/0
  :: Container second-db-1  Created                                     0.0s
  :: Container second-app-1 Created                                     0.0s
Attaching to second-app-1, second-db-1
second-db-1  | 2023-04-06 10:28:44+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.32-1.el8 started.
second-app-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.0.3. Set the 'ServerName' directive globally to suppress this message
second-app-1 | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 192.168.0.3. Set the 'ServerName' directive globally to suppress this message
second-app-1 | [Thu Apr 06 10:28:45.229046 2023] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.56 (Debian) PHP/8.0.28 configured -- resuming normal operations
second-app-1 | [Thu Apr 06 10:28:45.229083 2023] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
second-db-1  | 2023-04-06 10:28:45+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
second-db-1  | 2023-04-06 10:28:45+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.32-1.el8 started.
second-db-1  | '/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'
second-db-1  | 2023-04-06T10:28:45.743704Z 0 [Warning] [MY-011068] [Server] The syntax '--skip-host-cache' is deprecated and will be removed in a future release. Please use SET GLOBAL host_cache_size=0 instead.
second-db-1  | 2023-04-06T10:28:45.747880Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.32) starting as process 1
second-db-1  | 2023-04-06T10:28:45.752933Z 0 [Warning] [MY-010159] [Server] Setting lower_case_table_names=2 because file system for /var/lib/mysql/ is case insensitive
second-db-1  | 2023-04-06T10:28:45.761606Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
second-db-1  | 2023-04-06T10:28:46.355031Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
second-db-1  | 2023-04-06T10:28:46.750322Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
second-db-1  | 2023-04-06T10:28:46.750366Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
second-db-1  | 2023-04-06T10:28:46.754179Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
second-db-1  | 2023-04-06T10:28:46.897479Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
second-db-1  | 2023-04-06T10:28:46.897597Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.32'  socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
```

도커 컴포즈 실습

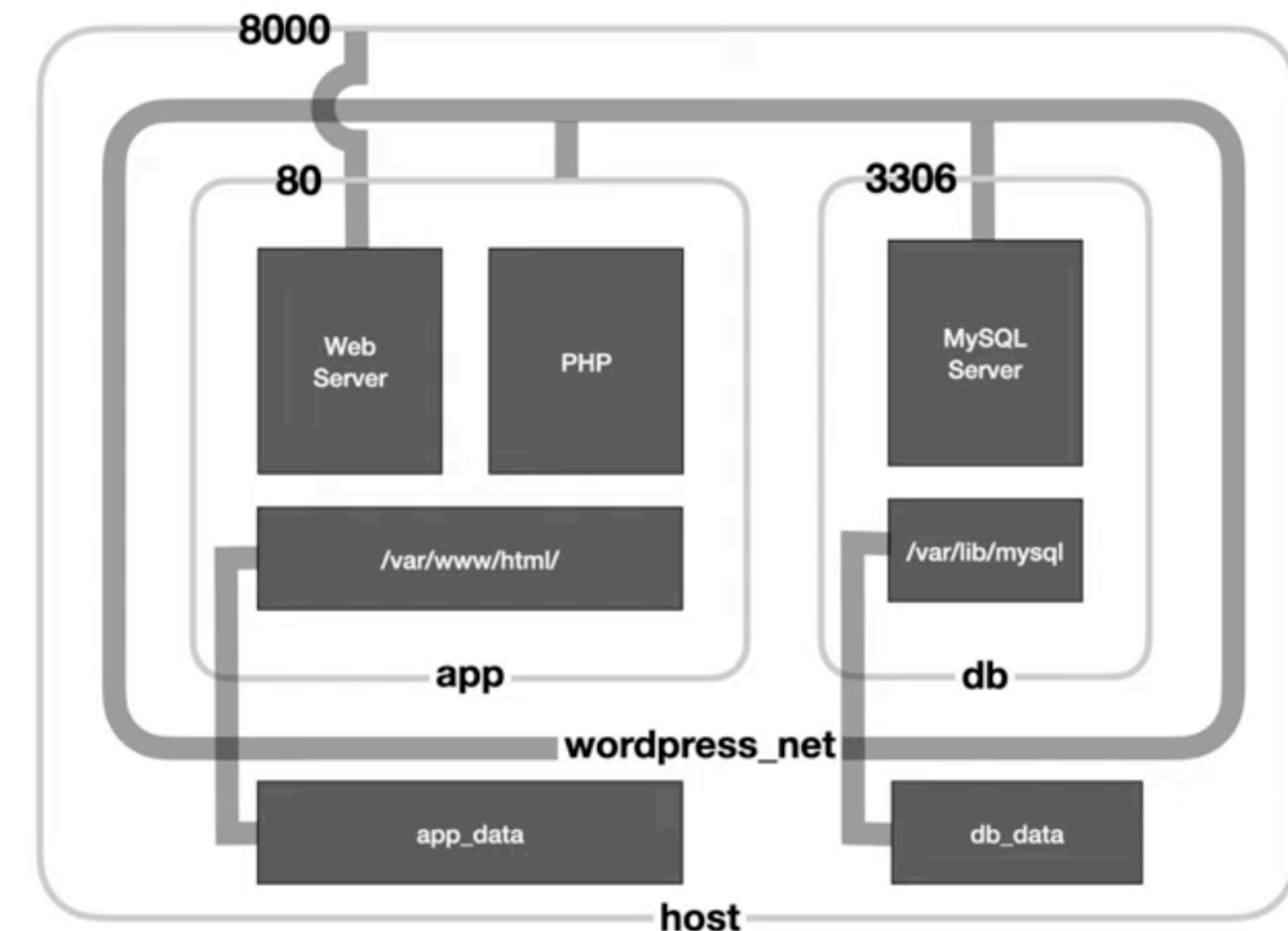
도커 컴포즈로 여러개 컨테이너 관리하기

docker-compose -f docker-compose.yml up --build



도커 컴포즈 실습

도커 컴포즈로 여러개 컨테이너 관리하기



나만의 도커 컴포즈 파일 콘테스트



- 개발 / 일상생활에 필요했던 나만의 도커 컴포즈 설정 파일을 정의하고 올려봐요
- 기한
 - 12/11(월) 15:00 까지 제출
 - 12/11(월) 수업시간에 시연
- 선정
 - 시연자 중 가장 많은 표를 받은 두분을 선정
 - 상품
 - 스타벅스 기프트 쿠폰



수고하셨습니다!