6 주차 암기

도커 이미지 커밋과 빌드

Dockerfile

- 도커 이미지를 빌드하기 위한 텍스트 기반의 스크립트 또는 설정 파일
- docker 스크립트를 재료 폴더에 컨테이너에 넣을 파일과 함께 두어 이미지 생성
- docker build -t ex22_original2 [재료_폴더_경로] (t옵션은 태그도 같이 저장하게 해줌)

```
nano ~/apa_folder/dockerfile

#dockerfile 내용
FROM httpd

COPY index.html /usr/local/apache2/htdocs/

# /tmp 디렉토리를 생성하고 권한을 설정합니다.
RUN mkdir -p /tmp && chmod 1777 /tmp

# 패키지 관리자를 업데이트하고 패키지를 설치합니다.
RUN apt-get update && \
    apt-get install -y apt-utils


#Dockerfile로 이미지 생성
#docker build -t [만들 이미지_이름_지정] [재료_폴더_경로]
docker build -t ex22_original2 ~/apa_folder
```

RUN mkdir -p /tmp && chmod 1777 /tmp 의 의미

- tmp 디렉토리를 생성하고 권한을 설정합니다.
- 특수권한 Sticky bit(1xxx)를 사용해 디렉토리 내에서 파일을 생성한 사용자만 해당 파일을 삭제할 수 있습니다.

이미지 생성시 커밋(Commit)과 빌드(Build)의 차이

Commit(커밋)

현재 실행중인 컨테이너의 상태, 파일 시스템, 변경사항을 그대로 새로운 이미지에 반영하여 저장한다.
변경 사항이 즉시 이미지에 저장되지만, 변경 이력 관리나 기록이 어렵다.
재현 가능성은 떨어지며, 체계적인 관리가 어렵다.
주로 컨테이너 내부에 작업한 내용을 즉시 저장하고 이후에 복원하고 싶을 때 사용한다.

Build(빌드)

Dockerfile을 사용하여 명시된 명령어들을 순차적으로 실행해 새로운 이미지를 생성한다.
현재 컨테이너의 상태, 파일 시스템과 무관하게 Dockerfile에 명시된 내용으로 이미지가 생성된다.
Dockerfile을 통해 이미지를 체계적으로 관리하고 재생산 할 수 있다. 빌드 과정이 Dockerfile에 기록되어 재현 가능성이 높다.

주로 일괄된 개발 환경을 유지할 때, 새롭게 이미지를 만들거나 여러 환경에서 동일한 설정을 배포할 때 사용한다.

- **주의 : 두 방식 모두 마운트 된 파일이나 폴더는 이미지에 포함시키지 않음.**

▼ 원본 글

- 커밋 (Commit)
  - 사용 목적: 현재 실행 중인 컨테이너의 상태를 새로운 이미지로 저장하는 과정.
  - 동작 방식: 현재 컨테이너의 파일 시스템과 그 상태(설치된 패키지, 생성된 파일 등)를 그대로 이미지에 반영.
  - 컨테이너 상태: 컨테이너 내부에서 일어난 변경 사항이 이미지에 포함됨.
  - 파일 포함: 실행 중인 컨테이너에서 발생한 모든 변경 사항이 이미지에 포함됨.
  - 관리 가능성: 변경 사항이 즉시 이미지에 저장되지만, 변경 이력 관리나 기록이 어려움.
  - 재현 가능성: 재현 가능성은 떨어지며, 체계적인 관리가 어렵다.
  - 주요 사용 상황:
    - 컨테이너 내부에서 작업한 내용을 즉시 저장하고 싶을 때,
    - 컨테이너 상태를 보존하고 이후에 동일 상태로 복원하고 싶을 때.
  - **요약**:
    - 현재 실행중인 컨테이너의 상태, 파일 시스템, 변경사항을 그대로 새로운 이미지에 반영하여 저장한다.
    - 변경 사항이 즉시 이미지에 저장되지만, 변경 이력 관리나 기록이 어렵다.
    - 재현 가능성은 떨어지며, 체계적인 관리가 어렵다.
    - 주요 사용 상황:
      - 컨테이너 내부에서 작업한 내용을 즉시 저장하고 싶을 때,
      - 컨테이너 상태를 보존하고 이후에 동일 상태로 복원하고 싶을 때.

- 빌드 (Build)
  - 사용 목적: Dockerfile을 사용하여 새로운 이미지를 체계적으로 생성하는 과정.
  - 동작 방식: Dockerfile에 명시된 명령어들을 순차적으로 실행하여 이미지를 빌드함.
  - 컨테이너 상태: 컨테이너의 현재 상태와는 무관하게, Dockerfile에 명시된 내용만으로 이미지가 생성됨.
  - 파일 포함: Dockerfile에 명시된 파일 복사(COPY), 패키지 설치(RUN apt-get install) 등의 작업만 이미지에 포함됨.
  - 관리 가능성: Dockerfile을 통해 이미지를 체계적으로 관리하고, 동일한 환경에서 이미지를 재생산할 수 있음.
  - 재현 가능성: 빌드 과정이 Dockerfile에 명확히 기록되므로 재현 가능한 이미지 생성이 가능.
  - 주요 사용 상황:
    - 일관된 개발 환경을 유지해야 할 때,
    - 새롭게 이미지를 만들거나 여러 환경에서 동일한 설정을 배포할 때.
  - **요약**

    Dockerfile을 사용하여 명시된 명령어들을 순차적으로 실행해 새로운 이미지를 생성한다.
    현재 컨테이너의 상태, 파일 시스템과 무관하게 Dockerfile에 명시된 내용으로 이미지가 생성된다.
    Dockerfile을 통해 이미지를 체계적으로 관리하고 재생산 할 수 있다. 빌드 과정이 Dockerfile에 기록되어 재현 가능성이 높다.

    주로 일괄된 개발 환경을 유지할 때, 새롭게 이미지를 만들거나 여러 환경에서 동일한 설정을 배포할 때 사용한다.
- **주의 : 두 방식 모두 마운트 된 파일이나 폴더는 이미지에 포함시키지 않음.**

도커 컨테이너 개조

- 컨테이너를 실행 중인 상태에서 컨테이너 내부의 파일 시스템, 환경 변수, 네트워크 설정 등을 변경하는 작업

컨테이너를 개조하는 방법

- 마운트와 파일 복사를 이용
- 컨테이너에서 리눅스 명령어 실행

컨테이너에서 명령어를 실행하려면 셸이 필요

- 컨테이너는 가장 일반적으로 사용되는 셸인 'bash'가 설치 되어 있음
- 컨테이너를 아무 설정 없이 실행하면 bash가 동작하지 않는 상태로 실행되기 때문에 bash를 실행해 명령을 입력받을 수 있는 상태로 만들어야 함
- 'bash'를 실행하려면 '/bin/bash' 인자를 전달해야 함
- docker run 또는 docker exec 커맨드와 함께 사용

docker exec

- 컨테이너 속에서 명령어를 실행하는 커맨드
- 'bash' 없이 어느 정도 명령을 직접 전달할 수는 있지만 초기 설정이 없어 동작하지 않는 경우도 있기 때문에 기본적으로는 셸을 통해 명령을 실행
- docker exec -it [컨테이너_이름] /bin/bash
- 컨테이너가 가지고 있는 bash 셸이 실행된다.

docker run

- docker run -dit [이미지_이름] /bin/bash
- 위 명령어 실행 시 컨테이너는 실행되지만 기본 패지키 (아파치)는 실행되지 않는다.
- 따라서 기본 패키지를 수동으로 실행해야 한다.
  - ex) service apahce2 start, apachectl start
  - 보통은 그냥 ubuntu에서 docker restart [컨테이너명] 한다.
- docker run 명령어를 사용해 bash 실행 시, 컨테이너는 실행 중이지만, 기본 패키지(예: Apache 등)는 실행되지 않음.
- 수동으로 기본 패키지를 실행하여야 함.
  - 예, service apache2 start 또는 apachectl start (이미지와 배포판에 따라 명령어가 다를 수 있음)
- docker run은 기본 사용만 권장, bash 실행이 필요할 경우 docker exec 사용여 컨테이너 내부로 접속 권장

도커 엔진 명령(도커 엔진 관리 범위)

- 도커 엔진의 시작/종료
  → 우분투에 의해서 관리되는 도커 엔진 명령어
  - sudo systemctl start docker
  - sudo systemctl stop docker 등

- 네트워크, 디스크 설정, 실행중인 컨테이너 목록 확인 등 컨테이너 전체에 대한 관리 작업 → 도커 엔진에 의해서 관리되는 명령어
    - docker network create
    - docker volume create
    - docker ps 등

도커 허브

도커 허브

도커 허브

- 도커 제작사에서 운영하는 공식 도커 레지스트리

도커 레지스트리

- 도커 이미지를 저장하고 관리하는 중앙 저장소로서 동작하는 서버
- 도커 이미지를 업로드, 다운로드, 검색, 삭제 등의 작업을 수행
- 도커 이미지를 공유하고 배포하기 위해 사용

리포지토리

- 도커 이미지의 집합을 나타내는 공간
- 이미지의 다양한 버전을 관리하고 구분

태그

- 도커 이미지의 버전을 식별하기 위한 라벨
- 이미지의 특정 버전을 구분하고 관리하는 데 사용
- 레지스트리_주소(도커 허브는 ID)/리포지토리_이름:버전

이미지에 태그를 부여해 복제하는 명령어

- docker tag [기존_이미지_이름] [레지스트리_주소]/[리포지토리_이름]:[버전]
- 명령어 실행 후 기존 이미지와 태그가 부여된 이미지가 둘 다 존재

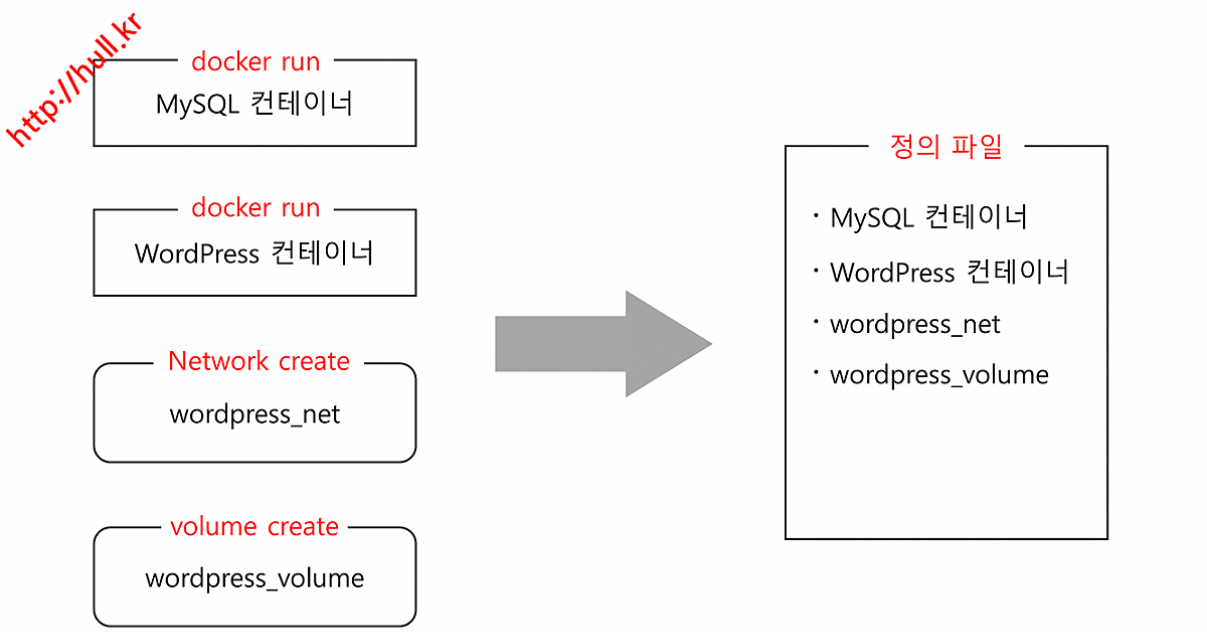
이미지를 업로드하는 명령어

- docker push [레지스트리_주소]/[리포지토리_이름]:[버전]
- 리포지토리는 처음 업로드할 때는 존재하지 않고, push 커맨드를 실행하며 만들어짐

도커 컴포즈

도커 컴포즈

- 시스템 구축과 관련된 명령어를 하나의 텍스트 파일(정의 파일)에 기재해 명령어 한번에 시스템 전체를 실행하고 종료와 폐기까지 한번에 하도록 도와주는 도구



도커 컴포즈의 구조

- 시스템 구축에 필요한 설정을 YAML(YAML Ain't a Markup Language) 포맷으로 기재한 정의 파일을 이용해 전체 시스템을 일괄 실행(run) 또는 일괄 종료 및 삭제(down)할 수 있는 도구
- 정의 파일에는 컨테이너나 볼륨을 '어떠한 설정으로 만들지 ' 에 대한 항목이 기재
- up 커맨드 → 도커 컴포즈 파일을 실행
  - docker run 커맨드와 비슷
  - 정의 파일에 기재된 내용대로 이미지를 내려받고 컨테이너를 생성 및 실행
  - 정의 파일에는 네트워크나 볼륨에 대한 정의도 기재할 수 있어 주변 환경을 한꺼번에 생성 가능
- down 커맨드
  - 컨테이너와 네트워크를 정지 및 삭제
  - 볼륨과 이미지는 삭제하지 않음

도커 컴포즈 사용법

- 호스트 컴퓨터에 폴더를 만들고 이 폴더에 정의 파일(YAML 파일)을 배치

도커 컴포즈 파일

네트워크나 볼륨은 하나가 아닌 여러개를 지정할 수 있으므로 배열 형식으로 관리된다.
이럴 경우 하이픈(-)을 사용한다.

MySQL에는 포트포워딩 정보가 X
워드프레스와 MySQL은 같은 네트워크로 레벨 2(스위치)에서 연결되므로 포트포워드가 필요X
단 IP주소, 컨테이너 명을 알아야한다.

```
~/compose

nano ~/compose/docker-compose.yaml
```

```
version: "3"
services:
  # MySQL 컨테이너 설정
  mysql000ex11:
    image: mysql:5.7  # MySQL 5.7 이미지 사용
    networks:
      - wordpress000net1  # wordpress000net1 네트워크 연결
    volumes:
      - mysql000vol11:/var/lib/mysql  # MySQL 데이터 저장을 위한 볼륨 마운
    restart: always  # 컨테이너 재시작 설정: 컨테이너가 죽으면 항상 재시작
    environment:
      MYSQL_ROOT_PASSWORD: myrootpass  # MySQL 루트 비밀번호 설정
      MYSQL_DATABASE: wordpress000db  # WordPress 데이터베이스 생성
      MYSQL_USER: wordpress000kun  # WordPress 사용자 생성
      MYSQL_PASSWORD: wkunpass  # WordPress 사용자 비밀번호 설정

  # WordPress 컨테이너 설정
  wordpress000ex12:
    depends_on:
      - mysql000ex11  # mysql000ex11 컨테이너가 먼저 실행되어야 함
    image: wordpress  # WordPress 이미지 사용
    networks:
      - wordpress000net1  # wordpress000net1 네트워크 연결
    volumes:
      - wordpress000vol12:/var/www/html  # WordPress 웹 애플리케이션 파일
    ports:
      - 8085:80  # 호스트 포트 8085를 컨테이너의 80번 포트로 매핑
    restart: always  # 컨테이너 재시작 설정: 컨테이너가 죽으면 항상 재시작
```

```
    environment:
      WORDPRESS_DB_HOST: mysql000ex11  # MySQL 호스트 설정: 컨테이너 명을
      WORDPRESS_DB_NAME: wordpress000db  # WordPress 데이터베이스 이름 설
      WORDPRESS_DB_USER: wordpress000kun  # WordPress 데이터베이스 사용자
      WORDPRESS_DB_PASSWORD: wkunpass  # WordPress 데이터베이스 사용자 비

# 컨테이너 간 통신을 위한 네트워크 설정
networks:
  wordpress000net1:
      #driver: bridge # 네트워크 드라이버 가 기본 설정이면 본 내용 생략 가능

# 데이터 저장용 볼륨 설정
volumes:
  mysql000vol11:  # MySQL 데이터 저장용 볼륨
      #driver: local # 볼륨 드라이버가 기본 설정이면 본 내용 생략 가능
  wordpress000vol12:
```