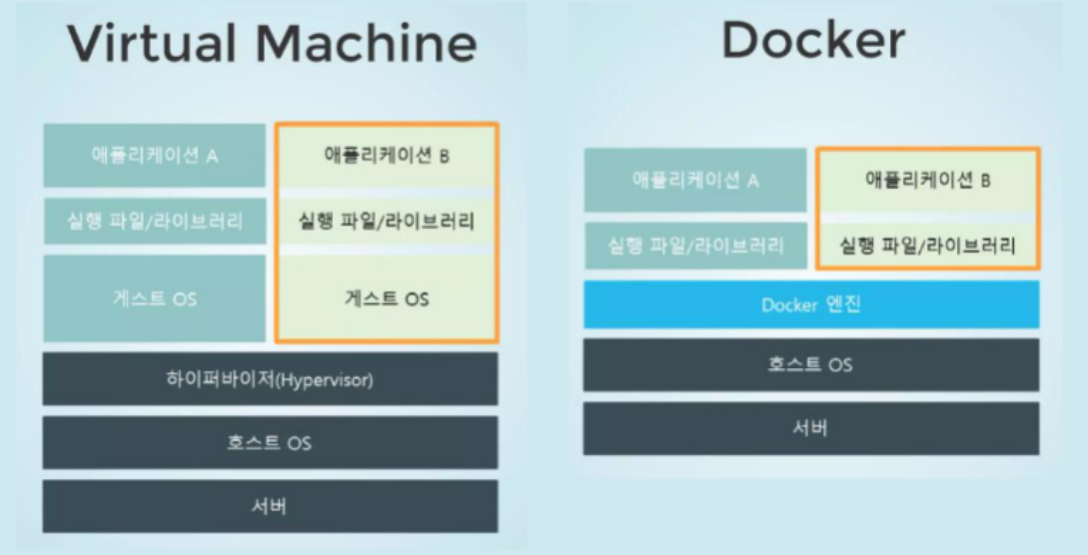
도커

- **도커(Docker)**
  - 도커는 애플리케이션을 컨테이너에 담아 실행할 수 있게 하는 컨테이너 기반 가상화 기술
  - 개발자가 만든 애플리케이션을 컨테이너에 넣으면 어디서나 동일한 환경에서 실행
- **컨테이너(Container)**
  - 애플리케이션과 그 실행에 필요한 모든 것을 포함한 독립적인 실행환경
  - 컨테이너는 OS 커널을 공유하며 앱마다 격리된 환경을 제공함
  - 가상머신(VM)보다 가볍고 빠름

VM과 도커



도커 구성요소

- **이미지(Image)**
  - 컨테이너 실행에 필요한 파일과 설정을 묶어둔 파일
  - 운영체제의 **프로그램**과 비슷한 형태
- **컨테이너(Container)**
  - 이미지를 실행한 상태로 독립적인 애플리케이션 실행 환경
  - 운영체제의 **프로세스**와 비슷한 형태
- **도커파일(Dockerfile)**
  - 도커 이미지를 생성하기 위한 설정파일
  - 빌드 스크립트와 비슷한 형태
- **도커 컴포즈(Docker Compose)**
  - 여러 컨테이너를 정의하고 실행하는 도커 유틸리티
- **도커 레지스트리(Docker Registry)**
  - 도커 레지스트리는 도커 이미지를 저장함
  - 도커는 기본적으로 Docker Hub라는 공개 레지스트리에서 이미지를 검색함

예제 `docker run` 명령

다음 명령은 `ubuntu` 컨테이너를 실행하고 로컬 명령줄 세션에 대화형으로 연결하여 실행합니다 `/bin/bash`.

```
$ docker run -i -t ubuntu /bin/bash
```

이 명령을 실행하면 다음 작업이 수행됩니다(기본 레지스트리 구성을 사용하는 경우).

1. 로컬에 이미지가 없으면 `ubuntu` Docker는 수동으로 실행한 것처럼 구성된 레지스트리에서 이미지를 가져옵니다 `docker pull ubuntu`.

2. Docker는 마치 수동으로 명령을 실행한 것처럼 새로운 컨테이너를 만듭니다 `docker container create` .
3. Docker는 컨테이너에 읽기-쓰기 파일 시스템을 최종 계층으로 할당합니다. 이를 통해 실행 중인 컨테이너는 로컬 파일 시스템에서 파일과 디렉토리를 만들거나 수정할 수 있습니다.
4. Docker는 네트워크 옵션을 지정하지 않았기 때문에 컨테이너를 기본 네트워크에 연결하기 위한 네트워크 인터페이스를 만듭니다. 여기에는 컨테이너에 IP 주소를 할당하는 것이 포함됩니다. 기본적으로 컨테이너는 호스트 머신의 네트워크 연결을 사용하여 외부 네트워크에 연결할 수 있습니다.
5. Docker는 컨테이너를 시작하고 .을 실행합니다 `/bin/bash` . 컨테이너가 대화형으로 실행되고 터미널에 연결되어 있기 때문에(`-i` 및 `-t` 플래그로 인해) Docker가 터미널에 출력을 기록하는 동안 키보드를 사용하여 입력을 제공할 수 있습니다.
6. `exit` 명령을 종료하기 위해 실행하면 `/bin/bash` 컨테이너는 중지되지만 제거되지 않습니다. 다시 시작하거나 제거할 수 있습니다.

도커 기본 명령어

- 이미지 관리

$ docker pull 이미지	도커 허브에서 이미지를 다운로드
$ docker images	로컬에 저장된 이미지 조회
$ docker rmi 이미지	로컬에 저장된 이미지 삭제

- 컨테이너 관리

$ docker run [옵션] [이미지] [커맨드]	컨테이너를 생성하고 실행 (예: docker run ubuntu ls) 주요 옵션: -it : 터미널에서 상호작용 가능하게 실행 -d : 백그라운드에서 실행 --name : 컨테이너에 이름 부여 -p [host port]:[container port] : 포트매핑
$ docker ps	현재 실행 중인 컨테이너 조회
$ docker start [컨테이너]	컨테이너 시작 (마찬가지로 stop, restar가 있음)
$ docker rm [컨테이너]	종료된 컨테이너 삭제
$ docker exec –it [컨테이너] /bin/bash	실행 중인 컨테이너에 접속
$ docker logs [컨테이너]	컨테이너 로그 확인

- 그 외

$ docker build –t [이미지] .	dockerfile을 기반으로 이미지를 생성
$ docker network ls	도커에서 사용 중인 네트워크 확인
$ docker network create [네트워크]	새로운 네트워크를 생성
$ docker network connect [네트워크] [컨테이너]	컨테이너를 네트워크에 연결
$ docker volume create [볼륨]	도커 볼륨을 생성
$ docker volume ls	도커 볼륨 목록을 확인
$ docker run –v [볼륨]:[컨테이너 경로] [이미지]	컨테이너 실행 시 볼륨(또는 host 디렉터리)을 연결 예: docker run –v my-volume:/data/ ubuntu

---

도커 기본 명령어 코드 실습

- 이미지 관리

```
# 도커 허브에서 이미지를 다운로드
$ docker pull [image]

# 로컬에 저장된 이미지 조회
$ docker images

# 로컬에 저장된 이미지 삭제
$ docker rmi [image]
```

- 컨테이너 관리

```
# 컨테이너를 생성하고 실행
$ docker run [option] [image] [cmd]
# 주요 옵션:
        # 터미널에서 상화작용 가능하게 실행
```

```
        -it
        # 백그라운드에서 실행
        -d
        # 컨테이너에 이름 부여
        --name
        # 포트매핑
        -p [host]:[container]

# 현재 실행 중인 컨테이너 조회
$ docker ps

# 컨테이너 시작(start), 중지(stop), 재시작(restar)
$ docker start [name]

# 종료된 컨테이너 삭제
$ docker rm [name]

# 실행 중인 컨테이너에 접속
$ docker exec [name] [image]
```

- 그 외

```
# dockerfile을 기반으로 이미지를 생성
$ docker build -t [image] .

# 도커에서 사용 중인 네트워크 확인
$ docker network ls
# 새로운 네트워크 생성
$ docker network create [network]
# 컨테이너를 네트워크에 연결
$ docker network connect [network] [name]

# 도커 볼륨을 생성
$ docker volume  create [volume]
# 도커 볼륨 목록을 확인
$ docker volume ls
# 컨테이너 실행 시 볼륨을 연결
$ docker run -v [volume]:[containerPATH] [image]
```

도커 만들기

- 플라스크 프로젝트가 가상환경에서 실행 중이라고 가정
- 플라스크 프로젝트 경로에서

```
$ pip freeze > reuqirements.txt
```

- 플라스크 프로젝트 경로에서 Dockerfile을 생성
- 도커파일이 위치한 디렉터리에서 도커 이미지를 빌드

```
$ docker build -t flask-pjk_241110 .
```

- 도커 컨테이너 실행

```
$ docker run -p 15001:15001 flask-pjk_241110
```

- 도커 파일(Dockerfile)은 내 프로젝트를 도커 이미지로 만들기 위한 일종의 **빌드 스크립트**
- 유명 오픈소스 등에서 찾을 수도 있음

```
# 1. 베이스 이미지 선택
FROM python:3.9-slim
# 2. 컨테이너 내부의 작업 디렉터리 생성 및 설정
WORKDIR /app
```

```
# 3. 의존성 파일 복사
COPY requirements.txt requirements.txt
# 4. 의존성 설치
RUN pip install --no-cache-dir -r requirements.txt
# 5. 애플리케이션 코드 복사
COPY . .
# 6. Flask 실행 포트 노출
EXPOSE 15022
# 7. 애플리케이션 실행 명령
CMD ["python", "app.py"]
```

- 도커 컴포즈(Docker compose)
  - yml(야믈) 파일로 여러 개의 컨테이너를 동시에 관리할 수 있는 유틸리티
  - 앞의 Dockerfile이 있다는 가정하에, 아래와 같이 **docker-compose.yml** 파일 작성

```
version: '3.8'
services:
        web:
                bulid:
                ports:
                        - "15022:15022"
```

- 도커 컴포즈 명령어

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