12주차

리눅스 시스템

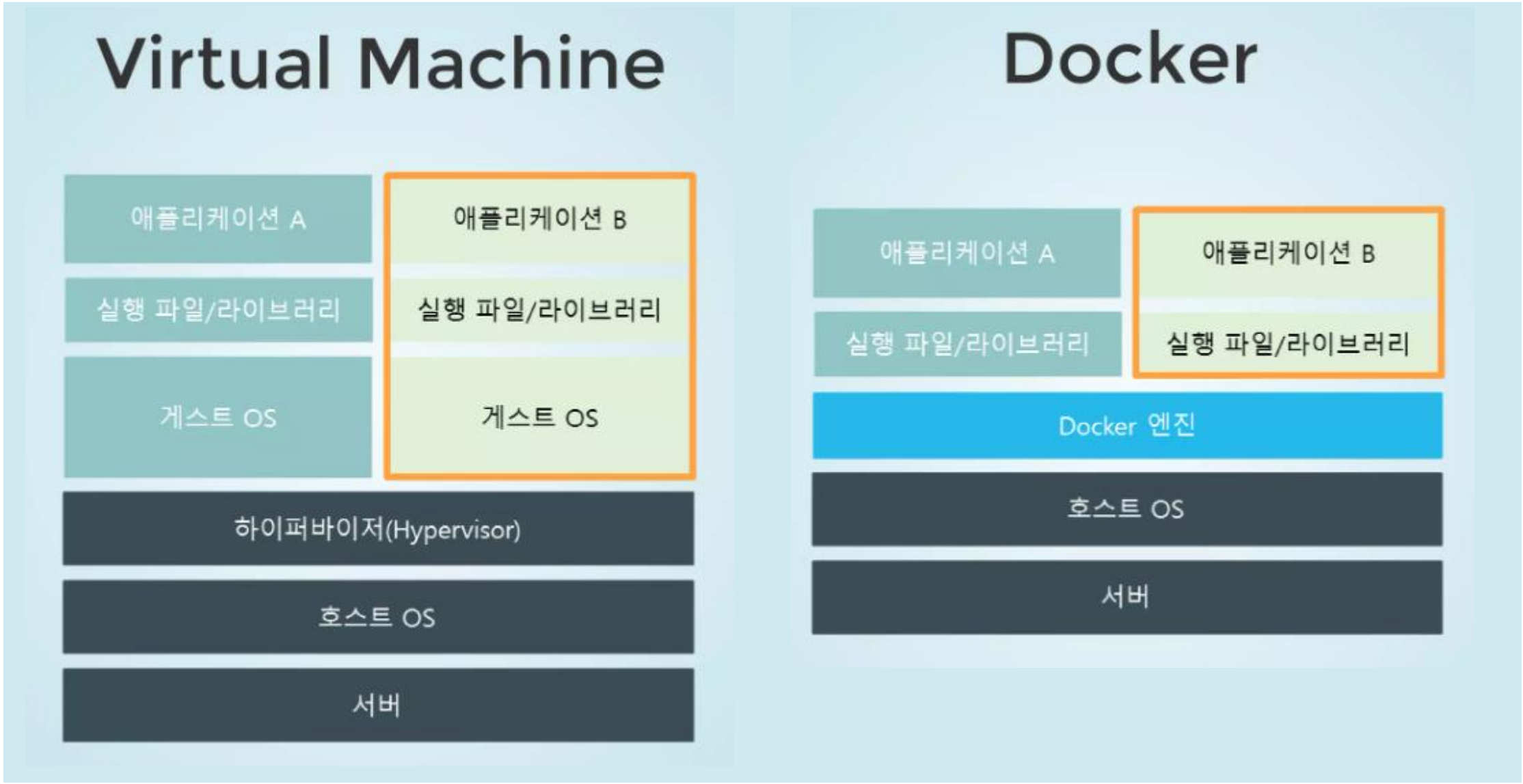
2024.동계계절학기

CONTENTS

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 실습환경
  - 컨테이너는 OS 커널을 공유하며 앱마다 격리된 환경을 제공함
  - 가상머신(VM)보다 가볍고 빠름

VM과 도커


Virtual Machine
애플리케이션 A
실행 파일/라이브러리
게스트 OS
애플리케이션 B
실행 파일/라이브러리
게스트 OS
하이퍼바이저(Hypervisor)
호스트 OS
서버
Docker
애플리케이션 A
실행 파일/라이브러리
애플리케이션 B
실행 파일/라이브러리
Docker 엔진
호스트 OS
서버

도커 구성요소

- 이미지(Image)
  - 컨테이너 실행에 필요한 파일과 설정을 묶어둔 파일
  - 운영체제의 프로그램과 비슷한 형태
- 컨테이너(Container)
  - 이미지를 실행한 상태로 독립적인 애플리케이션 실행 환경
  - 운영체제의 프로세스와 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

도커 실행 예시

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

1. 로컬에 이미지 가 없으면 ubuntu Docker는 수동으로 실행한 것처럼 구성된 레지스트리에서 이미지를 가져옵니다.
2. Docker는 마치 수동으로 명령을 실행한 것처럼 새로운 컨테이너를 만듭니다.
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. exit명령을 종료하기 위해 실행하면 /bin/bash컨테이너는 중지되지만 제거되지 않습니다. 다시 시작하거나 제거할 수 있습니다.

참조: https://docs.docker.com/get-started/docker-overview/

도커 기본 명령어

- **이미지 관리**

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

도커 기본 명령어

- **컨테이너 관리**

명령어	내용
$ 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

도커 기본 명령어

- 그 외

명령어	내용
$ 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 만들기 1 (1/3)

- 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

<- 현재 경로를 뜻하는 . 을 빠트리면 안됨

도커 만들기 1 (2/3)

- 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 15001

# 7. 애플리케이션 실행 명령
CMD ["python", "app.py"]
```

도커 만들기 1 (3/3)

- **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:
    build: .
    ports:
      - "15001:15001"
```

- **도커 컴포즈 명령어**
  - **$ docker compose up -d**
  - **$ docker compose down**

참고: 초기에는 Docker compose 명령어가 "$ docker-compose up" 이었음.
현재는 Docker compose가 도커의 정식 유틸리티로 채용되며 명령어 호출 구조가 조금 바뀜

도커 만들기 2 (1/4)

- **플라스크 프로젝트 + mysql 연결**
  - **두개의 컨테이너를 생성해야함**
  - **플라스크 프로젝트의 Dockerfile은 앞과 동일**
  - **도커 컴포즈를 사용하여 컨테이너 생성 및 실행**
  - **두개의 컨테이너를 연결하기 위해 도커 네트워크 정보 필요**
  - **mysql 컨테이너는 초기화(테이블 생성) 작업 필요**
  - **dockerfile**

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 15001

CMD ["python", "app.py"]
```

***서비스를 껐다켜면 DB가 지워지는 등 문제가 있음**
****현재 인프라를 당장 건드릴 수 없어 실습은 못하고 참고만 하길 바람**

도커 만들기 2 (2/4)

- **docker-compose.yml**

```yaml
version: '3.8'
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: flask-pjk_241110
    ports:
      - "15001:15001"
    depends_on:
      - db
    networks:
      - app-network

  db:
    image: mysql:latest
    container_name: mysql-pjk_241110
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: course_management
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro  # 초기화 SQL 파일 마운트
    ports:
      - "13306:3306"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

도커 만들기 2 (3/4)

- **init.sql**

```sql
CREATE DATABASE IF NOT EXISTS course_management;
USE course_management;

-- Courses 테이블
CREATE TABLE IF NOT EXISTS courses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    instructor VARCHAR(255),
    location VARCHAR(255),
    time VARCHAR(255)
);

-- Students 테이블
CREATE TABLE IF NOT EXISTS students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    student_id VARCHAR(50) UNIQUE NOT NULL
);

-- Enrollments 테이블 (수업과 학생 간 Many-to-Many 관계)
CREATE TABLE IF NOT EXISTS enrollments (
    course_id INT,
    student_id INT,
    PRIMARY KEY (course_id, student_id),
    FOREIGN KEY (course_id) REFERENCES courses(id) ON DELETE CASCADE,
    FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);
```

도커 만들기 2 (4/4)

- 파이썬 소스 코드 수정

```python
# MySQL 데이터베이스 연결 설정
db_config = {
    'host': 'db',
    'user': 'root',
    'password': 'example',
    'database': 'course_management'
}
```

```yaml
version: '3.8'
services:
  db:
    image: mysql:latest
    container_name: mysql-pjk_241110
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: course_management
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro  # 초기화 SQL 파일 마운트
    ports:
      - " 241110:3306"
    networks:
      - app-network
```

도커 만들기 3

- 플라스크 프로젝트는 도커로, mysql은 host로 그대로 쓰는 방법
  - 컨테이너로 만드는 소스코드에서 'localhost'는 컨테이너 자신을 가르키므로 host 머신을 가르키는 방법이 필요함
- ** 현재 host 머신의 mysql은 외부 포트를 다 막고 있으므로 실행은 안 됨

```
# MySQL 데이터베이스 연결 설정
db_config = {
    'host': 'host.docker.internal',
    'user': 'root',
    'password': 'example',
    'database': 'course_management'
}
```