1. Docker?

하드웨어를 가상으로 분리하여 여러개의 독립된 환경으로 쓸 수 있게 해주는 기술

참고) https://velog.io/@markany/%EB%8F%84%EC%BB%A4%EC%97%90-%EB%8C%80%ED%95%9C-%EC%96%B4%EB%96%A4-%EA%B2%83-1.-%EB%8F%84%EC%BB%A4%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80

2. Virtual Machine vs Docker

일반적인 가상화 머신을 통한 분할은 app 을 올리기 위해 각 VM 마다 guest os 단을 새롭게 생성해야하지만, Docker 는 host os 를 공유하기 때문에 app 만 독립적으로 올리면 된다.(다만, 이로 인해, Docker 로 생성한 가상환경에서는 다른 os 의 app 은 실행 불가능)

Docker 컨테이너의 동작 방식

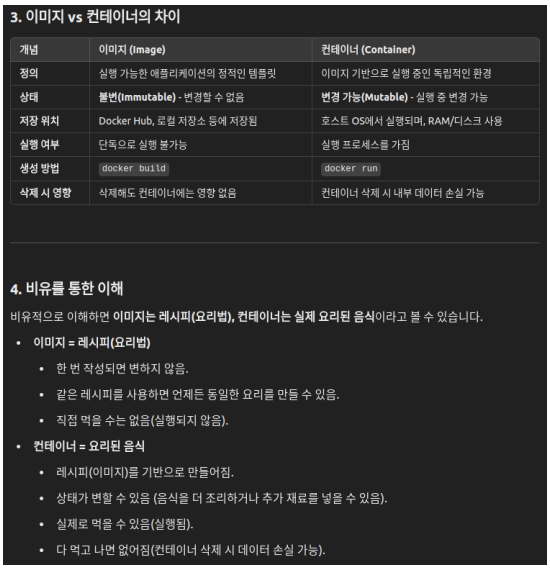
- Docker 컨테이너는 **각각 독립적인 파일 시스템, 라이브러리, 종속성을 포함**하지만, OS 커널은 공유합니다.
- 따라서, VM처럼 **별도의 OS를 완전히 가동하는 것이 아니라** 동일한 OS 커널에서 독립된 환경을 제공하는 방식입니다.

서로 다른 OS의 애플리케이션 실행 문제

- 같은 호스트 OS(예: Ubuntu)에서 실행되는 Docker 컨테이너는 **모두 동일한 커널을 공유**해야 합니다.
- 즉, **Linux 커널에서 실행되는 Docker 컨테이너는 Windows 앱을 실행할 수 없고**, 반대로 **Windows의 Docker에서 직접 Linux 바이너리를 실행할 수 없습니다.**
- 하지만, **WSL2 기반의 Docker Desktop을 사용하면 Windows에서 Linux 컨테이너를 실행할 수 있고**, Windows Server의 "Windows Containers" 모드를 사용하면 Windows 컨테이너를 실행할 수 있습니다.

3. image vs container

image 는 container 를 생성하기 위한 설계도, container 는 그 설계도를 본따 만든 실물(class - instance 의 관계와 유사하나, image 는 class 와 달리 실행 가능하다는 점을 유의)



3. 이미지 vs 컨테이너의 차이

개념	이미지 (Image)	컨테이너 (Container)
정의	실행 가능한 애플리케이션의 정적인 템플릿	이미지 기반으로 실행 중인 독립적인 환경
상태	**불변(Immutable)** - 변경할 수 없음	**변경 가능(Mutable)** - 실행 중 변경 가능
저장 위치	Docker Hub, 로컬 저장소 등에 저장됨	호스트 OS에서 실행되며, RAM/디스크 사용
실행 여부	단독으로 실행 불가능	실행 프로세스를 가짐
생성 방법	`docker build`	`docker run`
삭제 시 영향	삭제해도 컨테이너에는 영향 없음	컨테이너 삭제 시 내부 데이터 손실 가능

4. 비유를 통한 이해

비유적으로 이해하면 **이미지는 레시피(요리법), 컨테이너는 실제 요리된 음식**이라고 볼 수 있습니다.

- **이미지 = 레시피(요리법)**
  - 한 번 작성되면 변하지 않음.
  - 같은 레시피를 사용하면 언제든 동일한 요리를 만들 수 있음.
  - 직접 먹을 수는 없음(실행되지 않음).
- **컨테이너 = 요리된 음식**
  - 레시피(이미지)를 기반으로 만들어짐.
  - 상태가 변할 수 있음 (음식을 더 조리하거나 추가 재료를 넣을 수 있음).
  - 실제로 먹을 수 있음(실행됨).
  - 다 먹고 나면 없어짐(컨테이너 삭제 시 데이터 손실 가능).

4. Docker 설치(Ubuntu 22.04)

참고) https://velog.io/@osk3856/Docker-Ubuntu-22.04-Docker-Installation