9주차(1)_Docker 이론

📖 **목차**

Docker란?

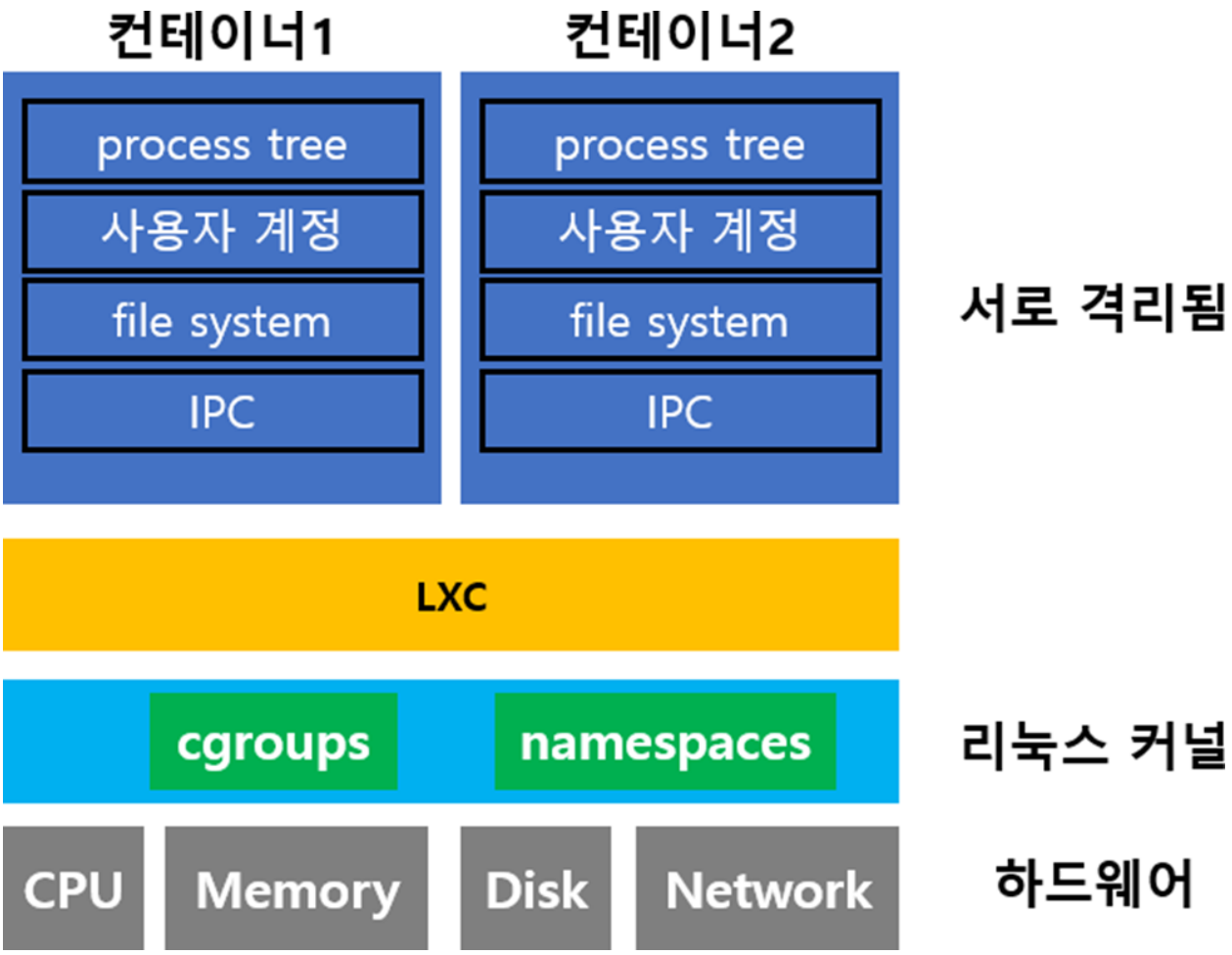
Docker는 **컨테이너** 기술을 사용하여 어플리케이션에 필요한 환경을 구축하고, 테스트 및 배포를 할 수 있게 해주는 플랫폼이다.

기존에 사용했던, Virtual Machine 처럼 사용자의 로컬 환경과 독립된 새로운 환경을 만들어주는 도구라고 볼 수 있다.

가상 머신 위에 운영체제를 깔아주는 도구는 VMWare, Virtual Box 같은 프로그램이 있다.

Docker의 경우, 운영체제가 호스트 컴퓨터에 실제로 설치되지 않기 때문에, 독립된 환경에서 돌아가며, 가상머신을 여러 개 돌렸을 때도 메모리 점유율, 성능 면에서 효율적임.

- **Host** : 운영체제가 설치된 computer
- **Container** : host에서 실행되는 각각의 격리된 실행 환경. application과 application을 구동하는 환경을 의미하며, ‘Host OS’와 격리되어 있음.



❓ Web을 만드려고 하면 Web Server, Database 등등을 일일히 설치해야하는데.. 설치하기 까다롭고 시간도 많이 걸리고, 한 번에 제대로 되지도 않고..

누가 설치한 걸 그대로 사용할 수 있으면 얼마나 좋을까?

Docker를 사용하는 이유

Docker 는 운영체제를 가상화한 container로 만들기 때문에 Docker를 사용하면 윈도우에서 리눅스, OSX 를 사용할 수 있고 리눅스와 OSX 에서도 다른 운영체제를 사용할 수 있게 해준다.

기존에도 Virtual Machine이라는 SW를 사용해서 여러 운영체제를 사용할 수 있었으나 매우 무거운 방식이라서 성능저하가 크다.

하지만, 컨테이너를 사용하면 **어느 운영체제나 어느 환경에서든 Docker만 깔려있으면 똑같은 실행 환경을 구축**할 수 있다. 또한 호스트 운영체제의 핵심부(Kernel)을 그대로 사용하는 방식으로 지원하기 때문에 성능저하와 오버헤드가 적다.

정리하자면,

1. **빠른 설치**: 애플리케이션의 이미지만 있다면 다운로드(pull)기능을 이용해 쉽고 빠르게 설치할 수 있다.
2. **애플리케이션 이식성**: 호스트OS나 플랫폼 버전에 상관없이 애플리케이션과 설치 파일들을 사용 가능하다.
3. **버전제어**: 컨테이너 버전 제어가 쉽다.
4. **쉬운 유지 관리**: 디펜던시에 관한 문제들을 관리하는 것이 쉽다.

서버 운영에서의 이점

대부분의 서버는 여러 대의 컴퓨터로 구성되어 있기 때문에 여러 서버에서 동일한 소프트웨어를 실행하면서 서비스를 제공한다.

서버 또한 우리가 사용하는 컴퓨터처럼 서버 가동을 위해서 여러 소프트웨어를 설치한다.

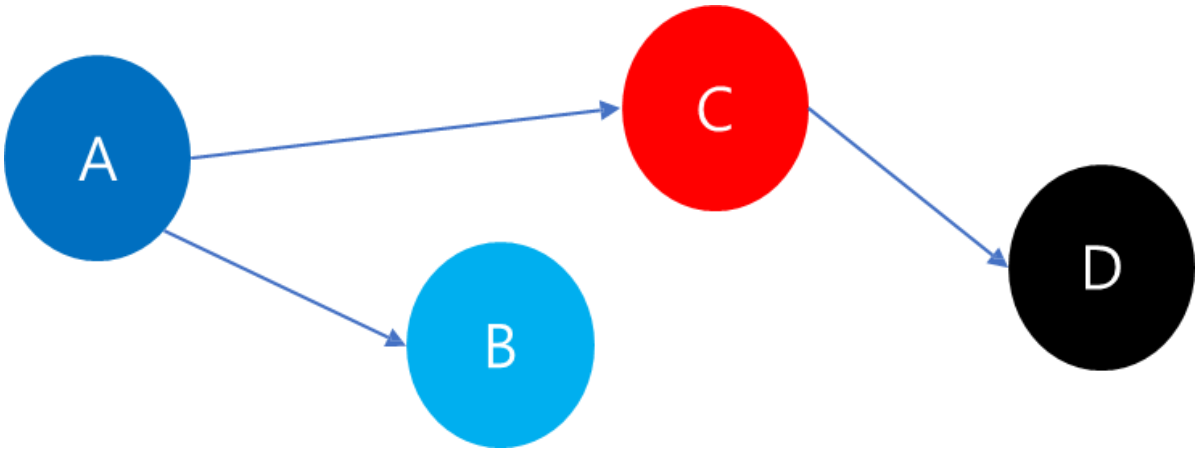
BUT 소프트웨어는 각각 **version**과 **dependency**가 존재한다.

- version이란?

  : 소프트웨어는 업데이트를 하면서 진화하고, 각 과정은 버전으로 구분한다.

- dependency이란?

  : 서로 다른 패키지가 기능 사용을 위해 다른 패키지에 의존한다.

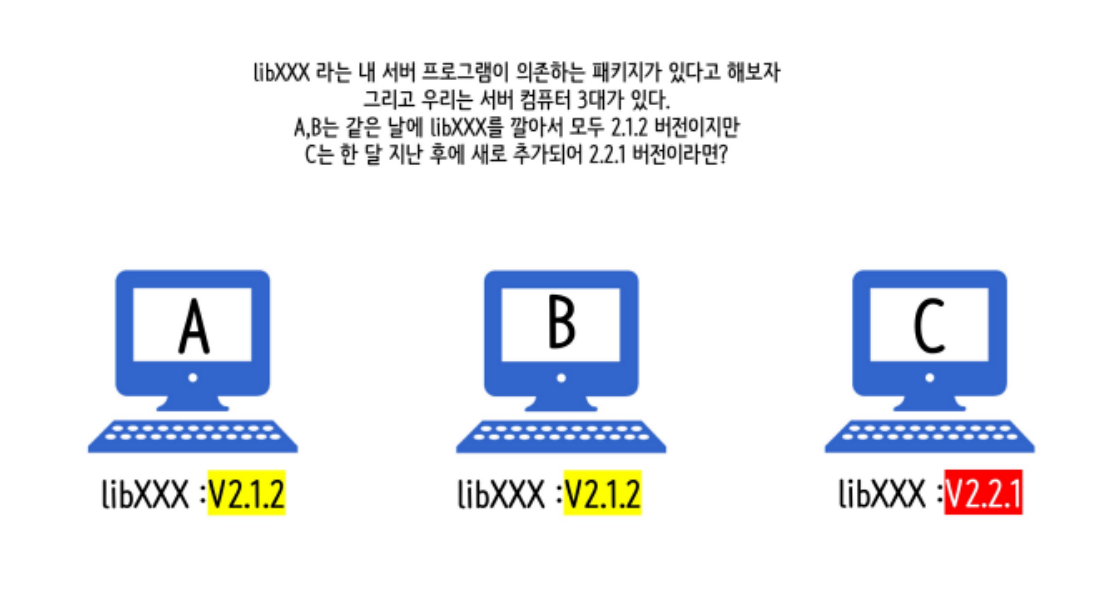


서버에는 **수 천 가지의 패키지가 설치**되고 여러 대의 서버를 **동일한 목적**을 위해 사용한다.

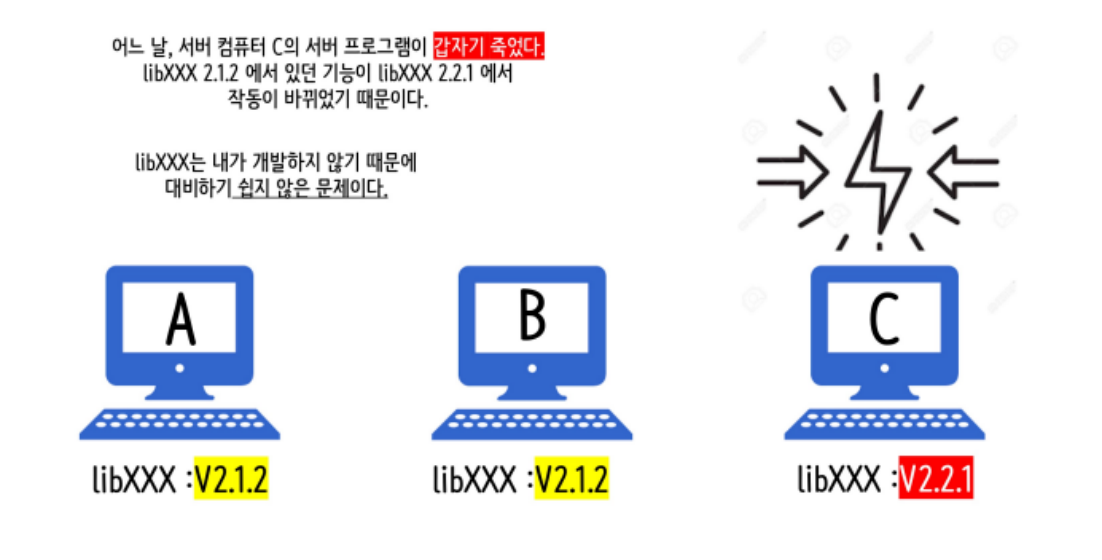
또한, 서버에서 서비스되는 프로그램은 이 수천 개의 패키지에 의존한다

그런데, 패키지는 시간이 지남에 따라 업데이트 된다.

❓ ex1) libXXX 라는 내 서버 프로그램이 의존하는 패키지가 있다고 해보자 그리고 우리는 서버 컴퓨터 3대가 있다. A,B는 같은 날에 libXXX를 깔아서 모두 2.1.2 버전이지만 C는 한 달 지난 후에 새로 추가되어 2.2.1 버전이라면?



❓ ex2) 어느 날, 서버 컴퓨터 C의 서버 프로그램이 갑자기 죽었다. libXXX 2.1.2 에서 있던 기능이 libXXX 2.2.1 에서 작동이 바뀌었기 때문이다. libXXX는 내가 개발하지 않기 때문에 대비하기 쉽지 않은 문제이다.

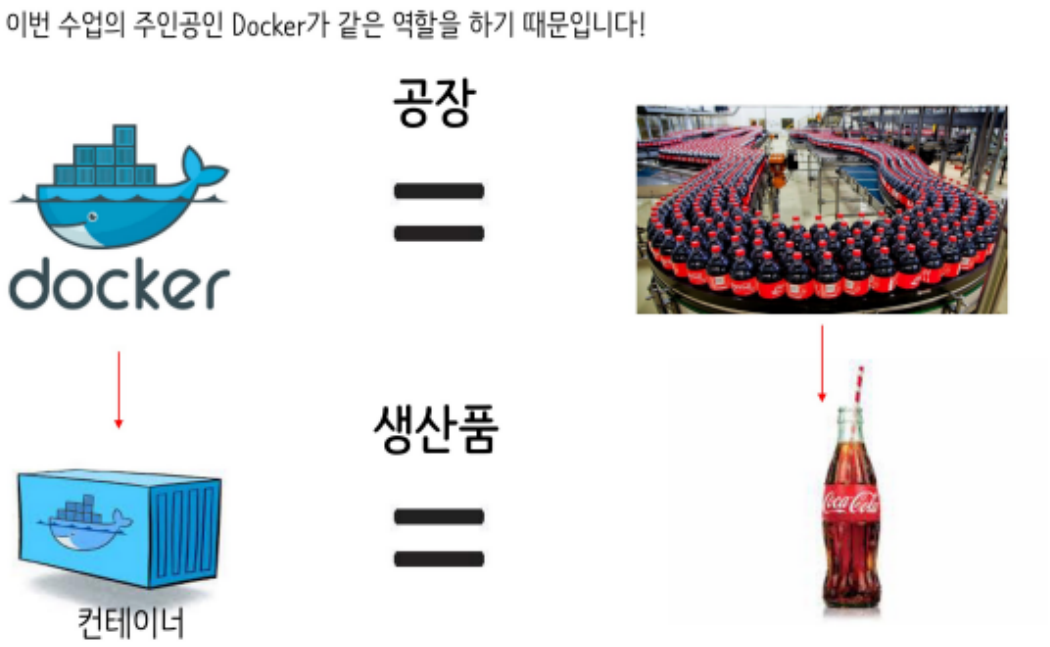


이런 일이 수 천, 수 만 개의 패키지에서 언제든지 일어날 수 있다.
패키지 뿐만 아니라, 운영체제의 버전도 큰 영향을 끼친다.
예시로 A,B 컴퓨터에서는 되는데 C컴퓨터에는 정상적으로 작동하지 않을 수도 있다.

즉, **조금씩 다른 서버**가 만들어지게 된다!

지금까지 설명한 관리방식은 비유하자면 수공예 작업에 가깝다 큰 틀은 같지만 조금씩 다른 서버의 상태가 만들어지게 된다. 그런데, 작은 차이가 모든 서비스를 중단시킬 수 있다는 것!

그래서 우리가 원하는 것은 **모든 서버가 공장에서 찍어낸 듯 동일한 상태**이다!!



💡 **즉,**
Image = 콜라의 레시피
Docker = 공장
Container = 만들어진 콜라

오픈소스에서의 이점

오픈소스는 많은 사람들이 수정하고 사용할 수 있는 소프트웨어이다.

그러면 아래와 같은 상황이 빈번할 것 같은데..?

❓ **이 사람 컴퓨터에는 되는 것 같은데 왜 내 컴퓨터에서는 안되는 거지..?**

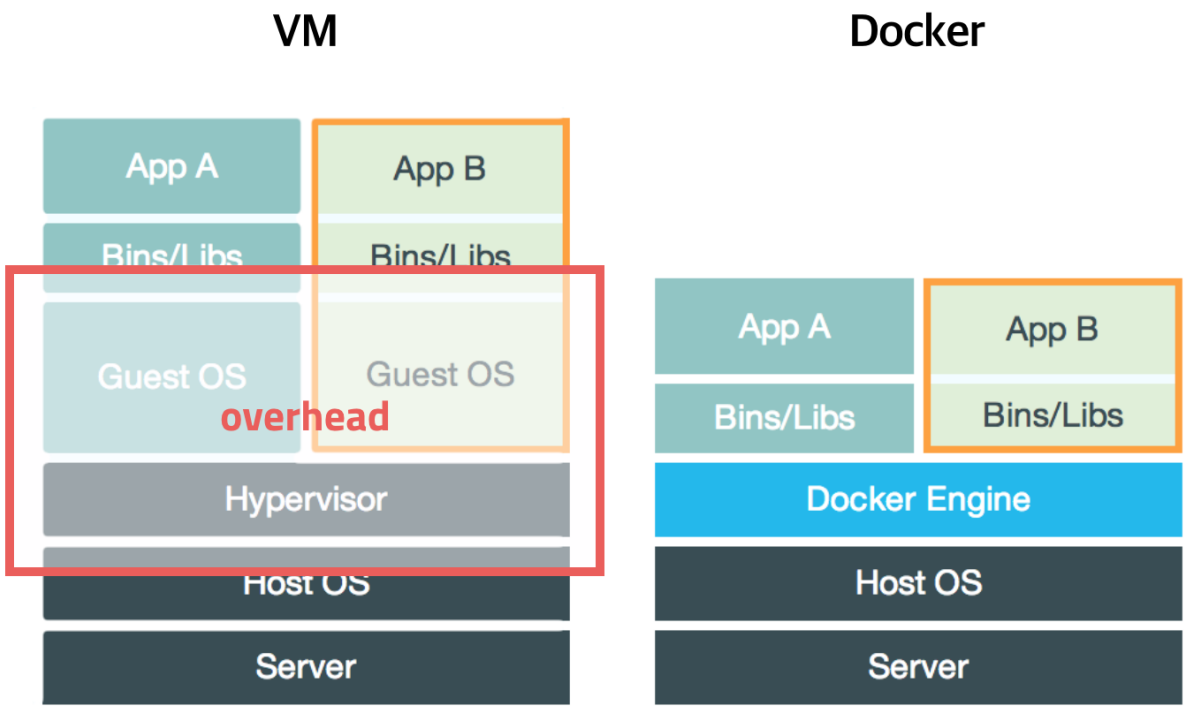
a) OS가 다르거나 version이 다름
b) 패키지 설치가 안되어있거나 version이 다름
c) 환경변수 설정이 다르게 되어있음
d) 프로그램이 참조하는 폴더 및 파일이 없음
등등..

안되는 이유는 매우 다양함!!!!

⇒ 즉, 이 프로그램은 항상 이 환경에서 실행하면 좋겠다!를 도커 이미지로 제공하면 항상 동일한 환경에서 실행 가능함!

💡 **Docker를 사용하면?**

Docker vs Virtual Machine



- **Virtual Machine** : 호스트 가상화 방식
  - 호스트 OS위에 게스트 OS를 올려놓는 방식으로 구현하기 때문에 가상 컴퓨터인 게스트 OS가 물리적 자원을 분할해서 사용한다.
  - CPU, 디스크, 메모리를 필요 이상으로 사용하면서 overhead가 발생한다.
  - BUT 호스트와 게스트가 완전히 분리되어 있어 **보안 측면**에서 좋다.

- **Docker** : 컨테이너 가상화 방식
  - 호스트 OS에 독립적인 컨테이너를 만들고, 애플리케이션이 동작하는 데 필요한 라이브러리와 애플리케이션 등을 컨테이너 안에 포함시킨다.
  - 게스트 OS없이 실행 환경만 독립적으로 돌리면 되기 때문에 자원적인 측면에서 좋고, **가볍고 빠르다.**