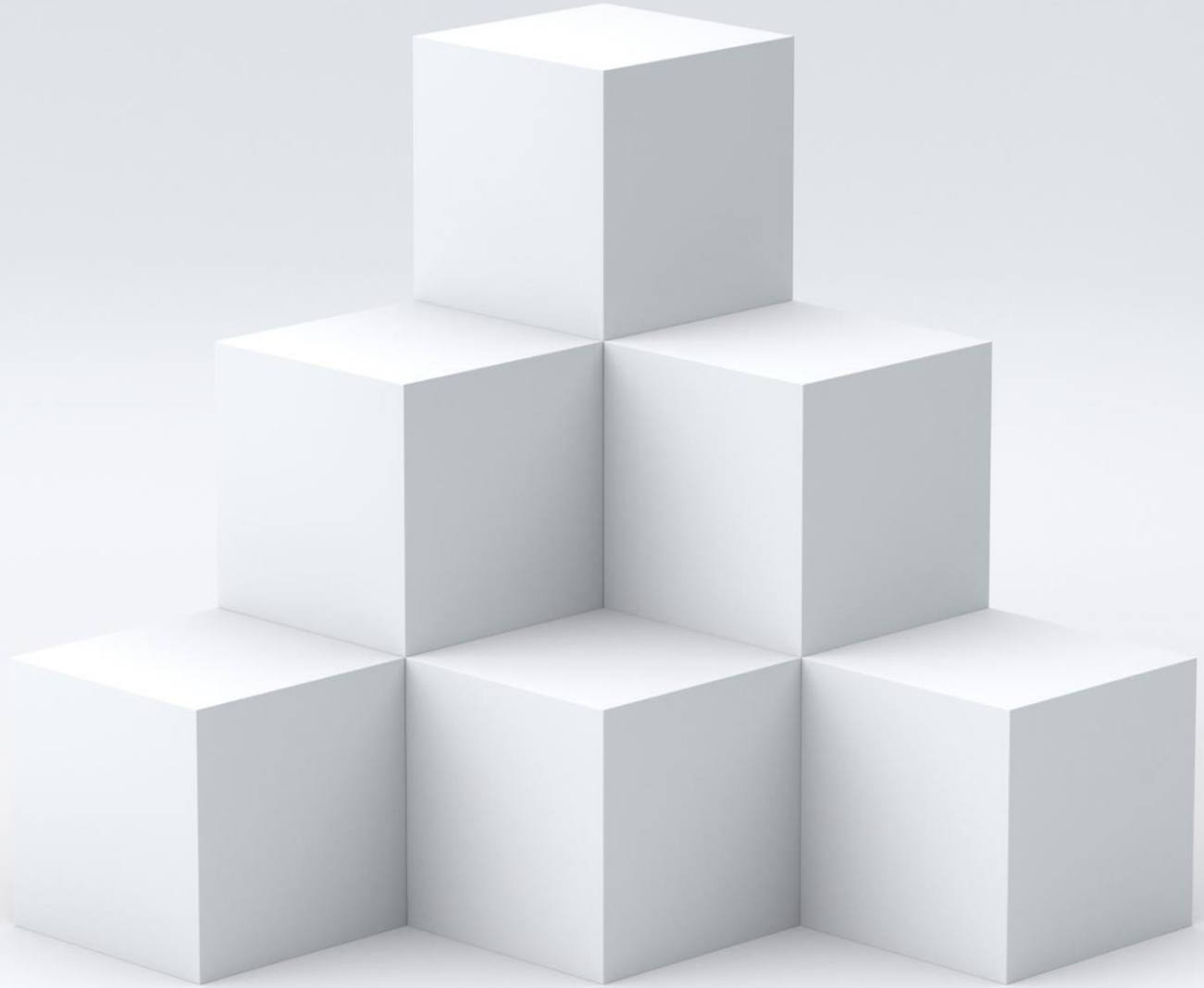
Software Engineering

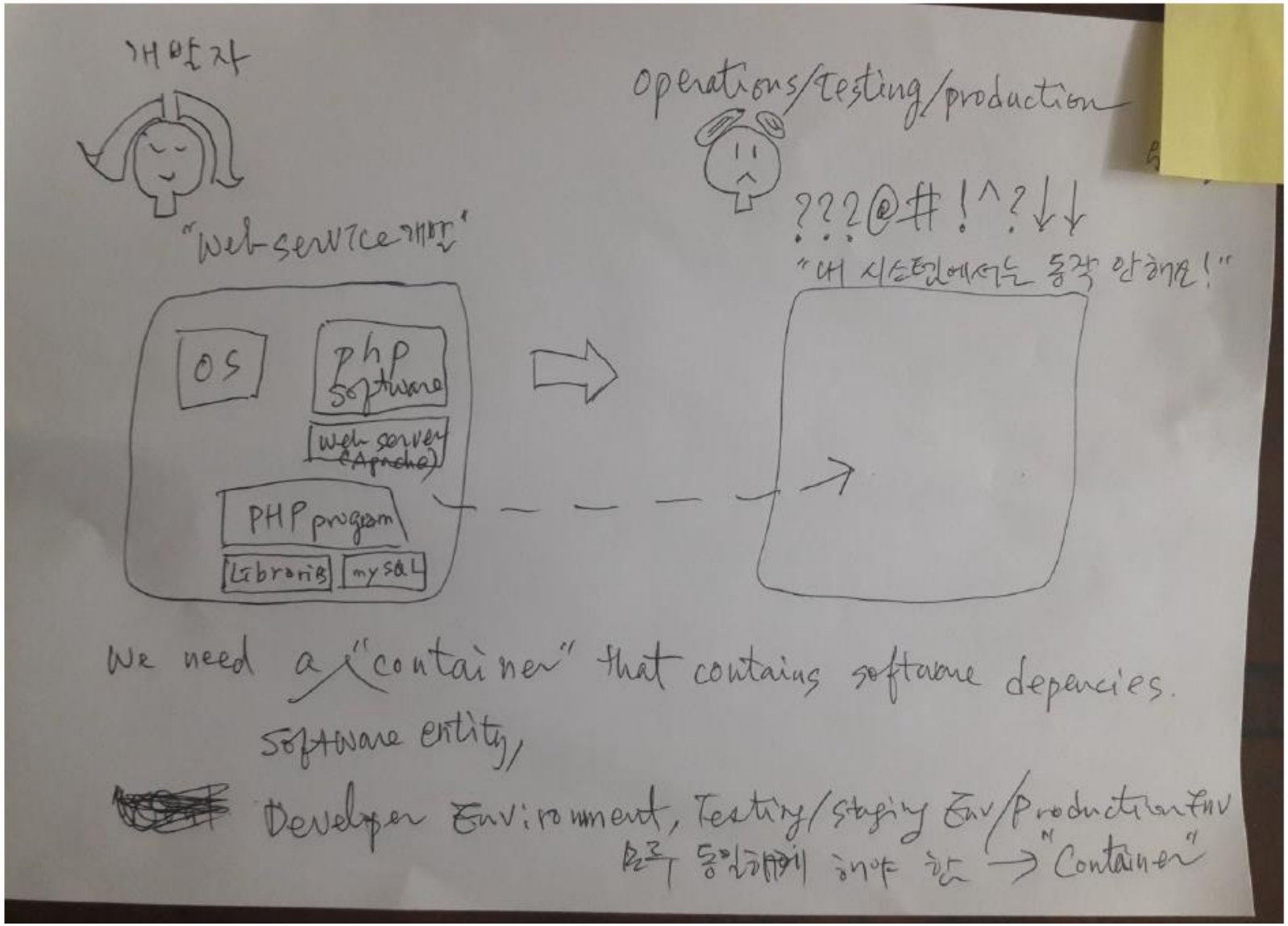
- Container -

Professor Han-gyoo Kim

2022



Real Life Example



Big Issues

1) Discrepancy in Dev Env. and Ops (testing/deploy) Env.
   - VM – Abstraction of Infrastructure (H/W)
   - VM's with same OS may have different environment for apps such as different set of libraries and dependencies
2) Lightweight Executable SW package
   - VM images are dull and heavy occupying large amount of DRAM of order of GB
   - High throughput system = apps should reside on DRAM (without swap) -> apps should occupy as small amount of DRAM as possible -> CPU has less impact than the size of DRAM to overall performance
3) Answer => Containerization

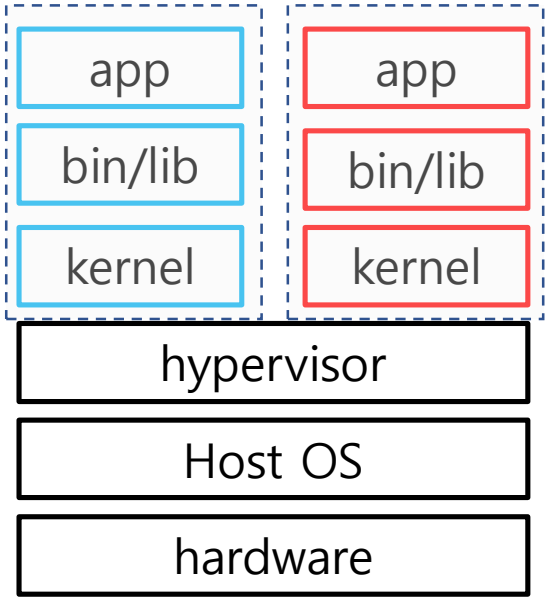
Docker

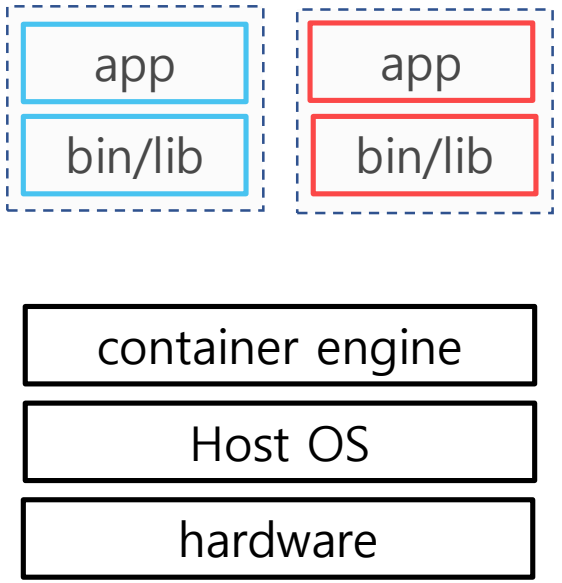
- Container engine을 만든 대표적인 회사로서 docker사의 container engine을 지칭. 즉 “container”를 실행시키는 platform
- Container engine = A set of Platform as a Service (PaaS) that uses OS level virtualization to deliver software packages called containers
- 2005, 2007, 2008 Linux “container” projects
- Docker, Inc. established in 2013
- Instead of running apps on top of different VMs which include OS kernels, why not running apps on top of “container” that interfaces to OS
- VM image = order of GB in size
- Container image = order of 50MB in size

화물 실어 나르는 컨테이너 배?

- 동일한 개념
- 임의의 다양한 화물들을 패키지로 묶어 일정한 규격(인터페이스)의 컨테이너에 적재하여 보내면 컨테이너를 내린 항구(VM)에서 규격 컨테이너를 부리고(discharge, container 엔진) 원하는 작업을 수행(app 실행)하는 개념

VM vs Container





VM	Container
과도한 메모리 점유	낮은 메모리 점유
느린 부팅 시간	빠른 부팅 시간
여러 VM 실행하면 급속히 성능 저하	훨씬 많은 수의 container 실행
Portability 제한	Platform에 좌우되지 않음
VM 사이에서 데이터 공유 불가능	여러 container가 데이터 공유 가능
낮은 효율	높은 효율

Container

- App과 app 실행에 필요한 모든 dependency 를 묶은 package
- 지정된 OS 위에서는 container가 실행될 수 있도록 “container engine”을 기계에 장착 – container engine은 결국 OS level에서의 virtualization을 app에 제공하는 것
- Container engine은 OS 별로 제공됨 (OS level virtualization)
- App을 container로 만드는 것을 containerization이라 하며, app은 어떤 dependency 모듈들을 호출하든 결국 app user process에 정의되어 있는 모듈과 OS에서 제공된 system call을 호출하므로 container는 container engine을 통해 해당 system call 등을 호출할 수 있도록 container 엔진과 인터페이스 되도록 만들어진 패키지이며 container 엔진은 OS 별로 제공되어 장착된다
- 당연히 container 엔진은 H/W 기계는 물론 VM 위에 장착되며 VM의 OS 종류에 따라 엔진이 제공되어야 한다
- 만일 host 기계의 OS와 다른 OS 위에서 containerized app을 실행하려면 hypervisor 위에 target OS용 container 엔진을 장착한다. 이와 같은 경우에도 hypervisor 위에 VM을 설치하여 app을 실행하는 것보다 hypervisor 위에 container 엔진을 설치하고 실행하는 것이 메모리 점유가 적고 실행 속도가 빠르다

Container Engine Architecture

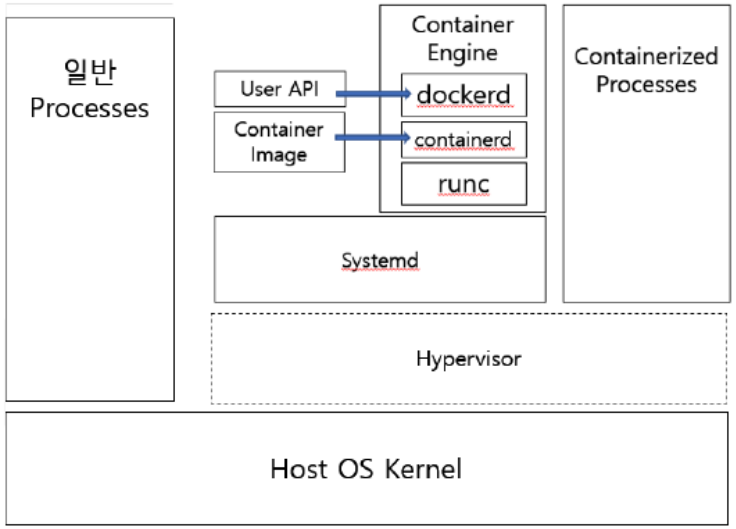
- Linux namespace, cgroups, runC 참조

아래 그림에서 보듯이 container는 물론 container engine에도 OS kernel은 없습니다. 사용자 명령을 처리하는 dockerd(docker daemon)와 container의 lifecycle을 관리하는 daemon인 containerd와 lightweight universal container runtime인 runC로 구성되어 있습니다. runC는 container를 set up합니다. 즉 container가 실행되도록 machine instruction/software로 변환하는 모듈입니다.

이를테면 사용자가 내린 docker run 명령을 dockerd가 받아 containerd가 해당 container image를 가져와 runC가 container process로 만들어 실행하는 것입니다.

만일 Ubuntu OS 위에서 실행되는 app을 container로 만들었는데 Windows 서버 기계에서 실행시키고 싶은 경우에는 Host OS인 Windows 위에 Windows 용 docker container를 설치하는 것은 물론 Ubuntu의 App을 Windows에서 실행시킬 수 있도록 Hyper-V (Microsoft의 hypervisor)를 설치해야 합니다.

같은 계열의 OS를 사용하는 호스트에서 container를 실행하는 데에는 당연히 hypervisor가 필요 없는 것은 물론이지만 hypervisor가 필요한 호스트 기계의 OS와 container app의 OS가 서로 다른 경우에도 Hypervisor 위에 VM을 설치하여 app을 실행하는 것에 비해 hypervisor를 설치하여 container를 실행하는 것이 수업에서 언급한 것처럼 빠르고 메모리 점유가 상대적으로 매우 적어서 효율적입니다.



Container의 일생