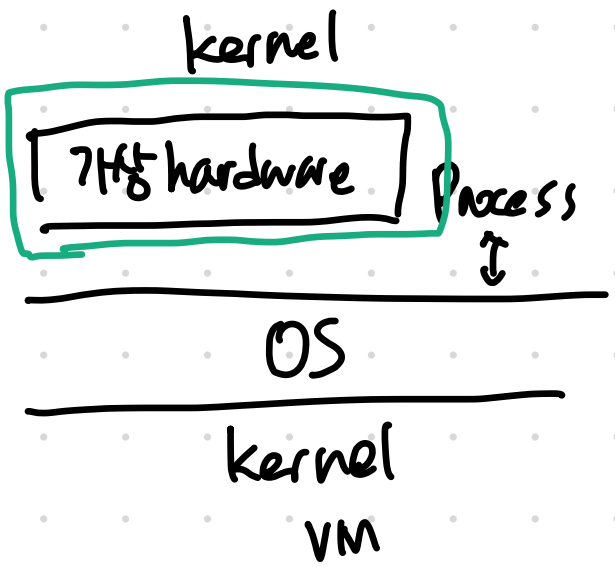
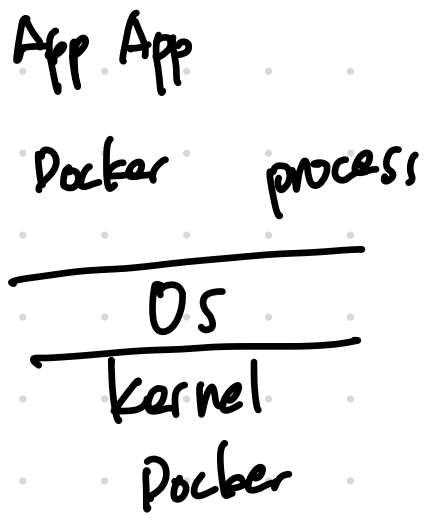
Chapter 1.

도커 . → 리눅스 컨테이너 ?
→ 이미지 ?

도커 : 이미지를 통하여 다양한 환경에서 실행되는 실행된다.
→ 컨테이너

VM → 다양한 OS 에서 실행. 소프트웨어로 구현된 하드웨어
컨테이너 → 하드웨어의 가상화 X . 프로세서 (격리된 환경에서 실행되는)

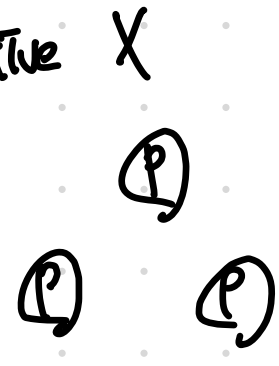




컨테이너 : 하드웨어의 가상화 필요없이 격리된 환경에서 실행되는 프로세스

ex) MacOS

Hive X



chroot → 루트디렉토리를 바꿔주는 것.

→ 이때한 디렉토리를 루트 " 처럼 인식하게 하는것.

→ 바깥의 디렉토리 접근 불가.

도커

lib 컨테이너 ↓ libvirt ↓ lxc ↓ systemdnspawn ↓

리눅스

즉, 도커는 리눅스커널에 격리된 프로세스를 만드는 기능을 함.

이미지 : 특정 프로세스가 실행되는 환경 (파일의 집합)

실제 도커 컨테이너 배포는 리눅스에서 함.

Configuration Management 등의 작업을 → Local에서 환경을 만든 후 실행.
⇓
미리구축된 환경사용.

도커를 쓰는 이유? → 컴퓨터 환경은 보편화 X. (서버관리가 힘듦)

→ 쉬운 길을 만드는 환경을 제공. (깨끗한)

도커 → 강력한 포터블 앱

재현성 Good. (이미지)

이미지 → 최소한의 파일을 제공.
→ 작동하는 걸 보장하는 상태.
→ 같은 환경.

버추얼ENV → python ↓ 개발환경

Note

컨테이너 (프로세스) → Host에 접근이 불가. 보안성 Good.

Ch1. container 사용 이유

1. 개발환경 - global dependency를 막음.

2. VMware, VBox

Hypervisor : 호스트 컴퓨터에서 다수의 OS를 동작하기 위한 논리적 플랫폼.

→ VMM

2가지 ┌ Type 1: 하드웨어에서 OS처럼 직접 실행
       └ type 2: 호스트 운영체제에서 실행됨.

↳ 대표적인, VMware 등.

Guest.

Hypervisor.

Host.

3. Container

Bin Bin Bin

Container Engine

OS

Host

⇒ Hypervisor의 단점인 Guest OS에서 App을 실행시키는 오버헤드 X.

↳ OS만 공유.

4. Container 원리.

4.1 C-group : 프로세스 자원 관리 / 제한

4.2 Linux Namespace : Linux 오브젝트의 이름을 독립적 관리. → 충돌방지

Ch2. Development Environment.

1. Monolithic architecture

전통 아키텍처, 서비스가 하나의 거대한 아키텍처를 가질때.

Application UI.
|
Business Logic
|
DB

장점: 단순아키텍처 → 다루기 간편.
End to End 서비스 용이
처음 ~ 끝 서비스 제공.

단점: QA가 어렵다. 기능별로 알맞은 언어·프레임워크 선택이 까다로움.

2. Microservice Architecture

API를 통해 통신하는 소규모 독립적인 서비스.

ApUI
/ | \
BL BL BL
↑ | |
DB

장점: QA 쉬움, 컨테이너를 이용하여 알맞는 언어·프레임워크 사용.
단점: 모니터링 어려움 (잘게 나뉘어져 있어서), 통신관련 오류,
테스트 불편

Ch3. Docker Installation

1. Docker 이미지 : 필요한 프로그램, 라이브러리, 소스 → 합친 파일
   컨테이너 : 이미지를 격리하여 돌아가는 가상 environment.
   의존성 및 파일 시스템까지 패키징해서 빌드, 배포, 실행을 단순화함.
   cgroup : 가상화, Window는 hypervisor로 Docker를 리눅스
   Linux가 아닌것의 작업환경.

2. 클라우드 모델 (플랫폼)
   1. IaaS : 서비스로서의 인프라 2. PaaS : 플랫폼 구축, 웹서버X 서비스로서
   3. SaaS : 서비스로서의 소프트웨어

3. Docker의 환경.
   서비스가 커지면 커질수록 관리해야하는 컨테이너 양 大.
   → 내부적 : 도커 외부적 : 도커스웜, 쿠버네티스 활용.