VIRTUALIZATION & DOCKER

panya

가상화

- 리소스를 추상화 (큰 의미)
  - 프로세서(CPU), 메모리(Memory), 스토리지(Storage), 네트워크(Network) 등
  - 물리적인 자원들을 숨기고 논리적인 자원을 만들어내는 것
- 하나의 컴퓨터로 여러 개의 컴퓨터를 사용 (매우 좁은 의미)

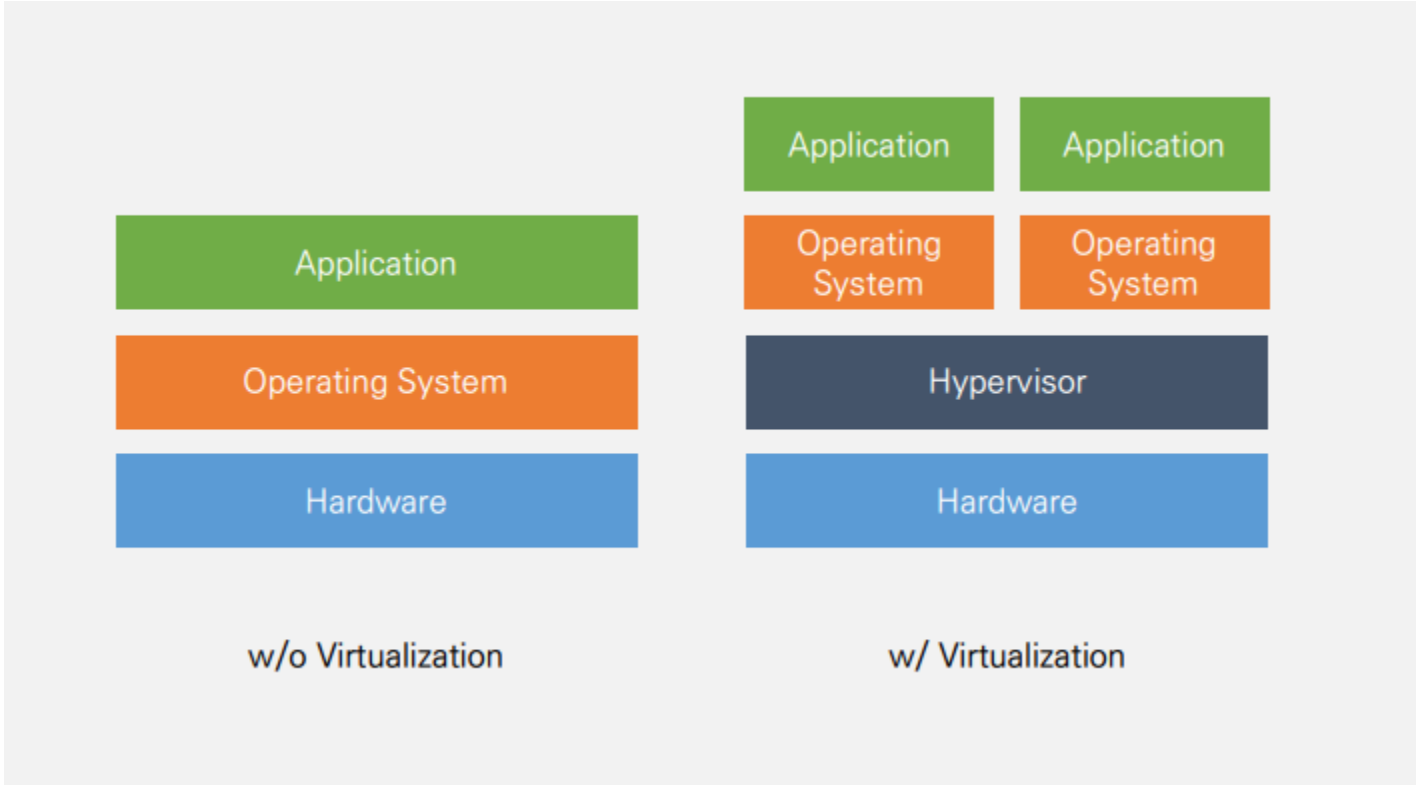
가상화를 사용하는 이유

자원의 효율적 사용

상황에 유연하게 대처할 수 있음

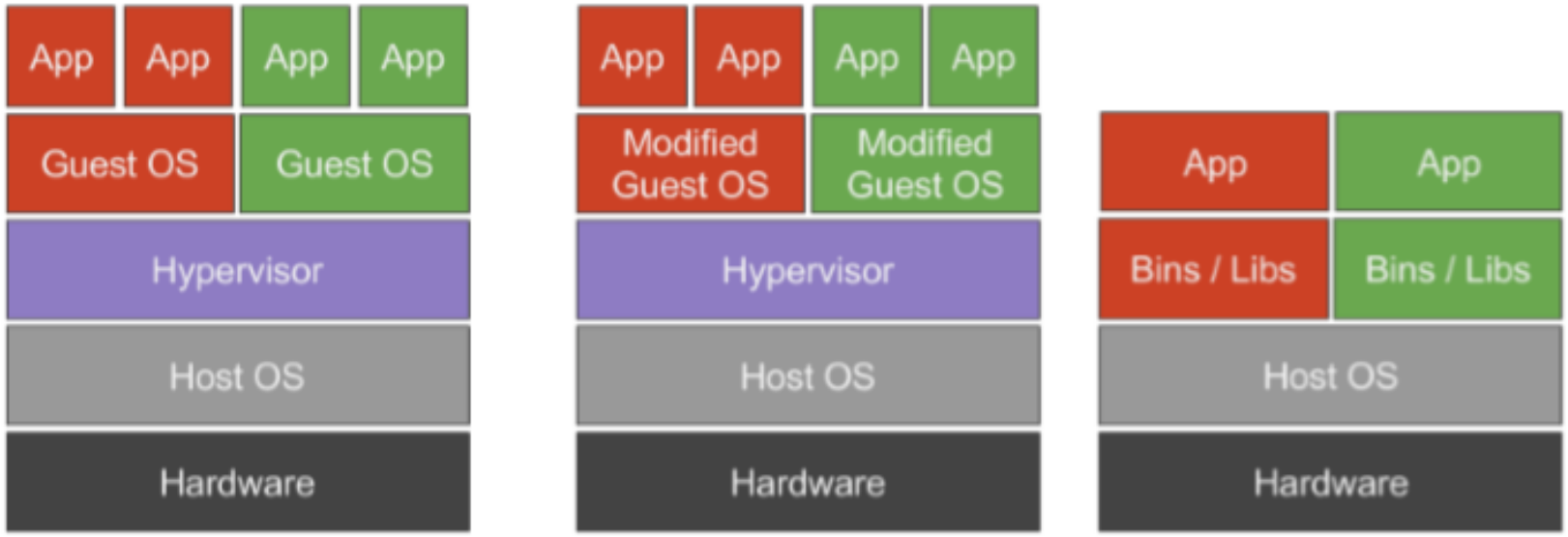
캡슐화 / 격리, 획일화된 환경

서버 가상화


Application
Operating System
Hardware
w/o Virtualization
Application
Application
Operating System
Operating System
Hypervisor
Hardware
w/ Virtualization

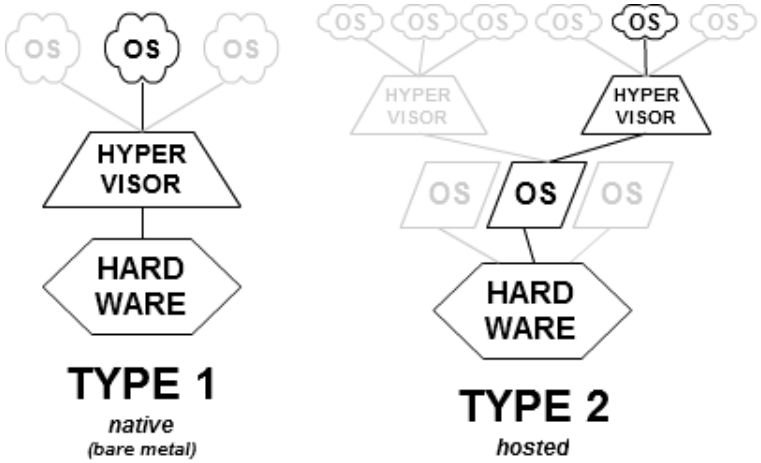
Virtualization

종류



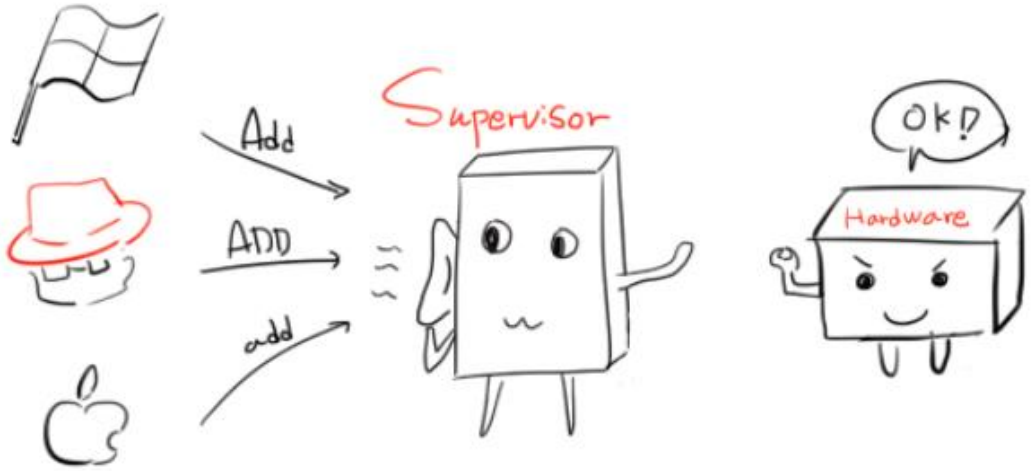
하이퍼바이저(Hypervisor)

- 호스트 컴퓨터에서 다수의 OS를 동시에 실행하기 위한 논리적 플랫폼
- 하드웨어로부터 제공되는 물리적인 레이어를 추상화하고,
  가상머신을 통해 기능들을 사용하도록 함
- HostOS(machine) -> Hypervisor -> GuestOS



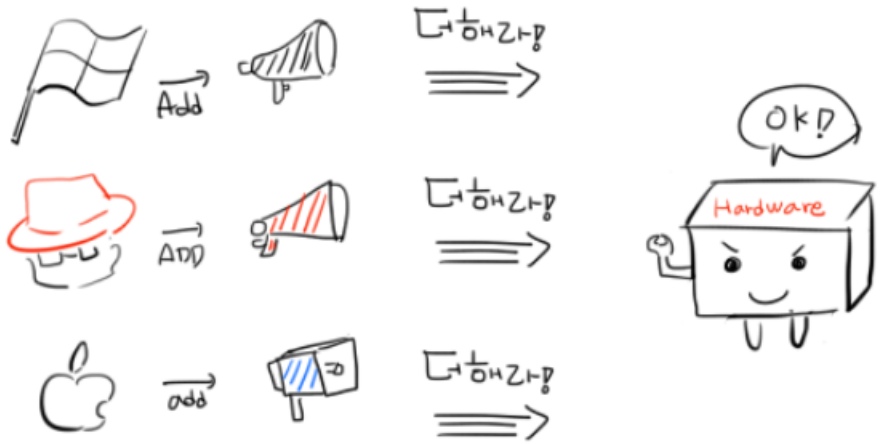
Full Virtualization (전가상화)

- 완전히 가상화, 언제나 개입
- OS는 하이퍼바이저를 모름
- CPU가 전가상화를 지원하는 VT(Virtualization Technology)를 지원해야함
- 모든 명령을 중재하기 때문에 비교적 느림
  - Dom0 (관리용 가상 머신)



Para Virtualization (반가상화)

- OS를 적당히 수정해서 하이퍼바이저에게 직접 요청 (Hyper Call)
- 전가상화에 비해 퍼포먼스가 좋다
- 게스트 OS가 오픈소스가 아니라면...



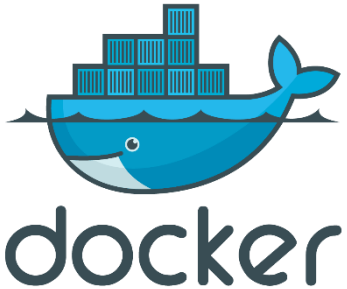
가상화 솔루션

- Xen (garam에서 사용중임)
- ESXi
- KVM
- 등등 ...

OS-Level Virtualization

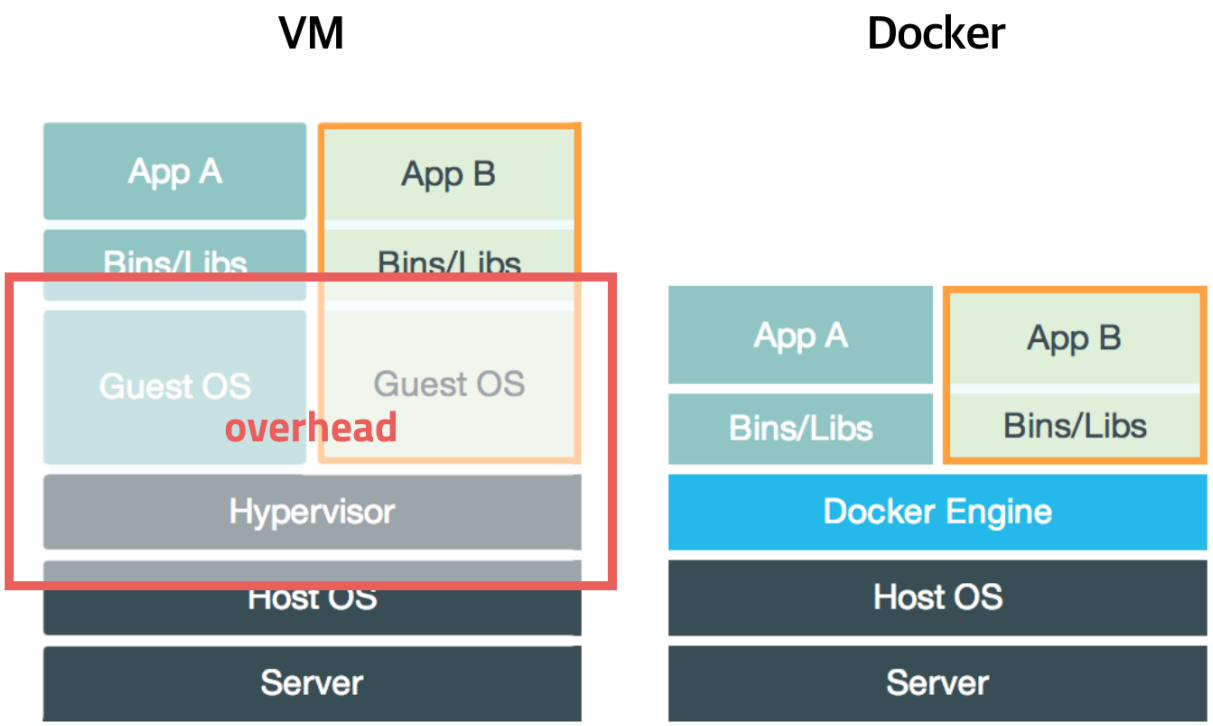
- OS까지는 같이 쓰고 그 위에서 가상화를 하자!
- 전체 OS를 설치하는 것이 아니기 때문에, 상대적으로 가벼움
- LXC (LinuX Containers)
  - 단일 리눅스 시스템에 동작하고 있는 프로세스를 격리시켜
    각 프로세스마다 독자적인 리눅스 시스템 환경을 구축
  - Chroots : 프로세스의 루트 디렉토리 변경
  - Cgroups : 프로세스 묶음이 접근할 수 있는 시스템 자원 제한 및 관리
    (CPU, 메모리, 디스크 I/O 등 격리)
  - Namespaces : 다른 namespace의 resource를 열람할 수 없도록 함
    (PID, hostname, 네트워크, IPC, 파일시스템 등 격리)
- Libcontainer : Linux의 virtualization feature들을 이용하기 쉽도록 묶은 라이브러리

Docker



- 컨테이너 기반의 오픈소스 가상화 플랫폼
- OS-Level Virtualization
- 컨테이너
  - 직육면체의 규격화된 사이즈 -> 동일한 인터페이스 제공
  - 화물이 들어간다 -> 프로그램 실행 환경

Docker


VM
App A
App B
Bins/Libs
Bins/Libs
Guest OS
Guest OS
overhead
Hypervisor
Host OS
Server
Docker
App A
App B
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

장점

- 가볍다
  - 퍼포먼스가 좋음
  - 전체 OS를 깔 필요가 없어 비효율을 줄일 수 있음
- DockerHub에서 이미지를 가져올 수 있음
- 그 외 수많은 장점들이 있음
  - 빌드용으로 도커를 쓰면 디펜던시 관리로 인한 탈모 예방
  - Snowflake Servers

Docker

Image

- 시스템의 스냅샷
- 실행 파일, 라이브러리, 기타 파일 등을 하나로 묶은 것
- 컨테이너 실행에 필요한 것

Container

- 이미지를 실행시킨 상태
- 격리된 한 환경의 단위 –> 독립적인 Linux 기기와 매우 유사하게 작동

Docker Commands

Docker 설치 https://docs.docker.com/install/

$ sudo apt install docker.io

$ docker --version

$ docker ps /docker container list : 현재 실행중인 컨테이너 출력

-a / --all : 실행중이 아닌 컨테이너까지 모두 출력

$ docker search [image] : 이미지를 검색

$ docker pull [image] : remote 저장소에서 [image]를 가져옴

[repository]:[tag]의 형태로 이름을 가짐

:[tag] 를 제공하지 않으면 자동으로 latest 사용

Docker – Exercise

도커 설치 확인

```
$ docker --version
$ docker ps
```

Sudo 없이 사용하기 (현재접속중인 사용자에게 권한 주기)

```
$ sudo usermod –aG docker $USER
```

Bionic(18.04) 버전의 'ubuntu' 이미지를 다운로드

```
$ docker search ubuntu
$ docker pull ubuntu:bionic
$ docker images
```

Docker Commands

새로운 container를 만들어 실행

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

옵션	설명
-d	detached mode 흔히 말하는 백그라운드 모드
-p	호스트와 컨테이너의 포트를 연결 (포워딩)
-v	호스트와 컨테이너의 디렉토리를 연결 (마운트)
-e	컨테이너 내에서 사용할 환경변수 설정
-name	컨테이너 이름 설정
-rm	프로세스 종료시 컨테이너 자동 제거
-it	-i와 -t를 동시에 사용한 것으로 터미널 입력을 위한 옵션
-link	컨테이너 연결 [컨테이너명:별칭]

Docker Commands

$ docker [status] [name]

- start: stopped 컨테이너를 run
- stop: running 컨테이너를 stop
- pause: running 컨테이너를 pause
- unpause: paused 컨테이너를 run
- kill: 컨테이너 상태와 관계없이 모든 프로세스를 kill, 컨테이너 stop
- restart: 컨테이너를 재시작

Docker - Exercise

$docker run -i -t -d --name hello ubuntu:bionic /bin/bash

hello라는 이름으로 ubuntu 컨테이너를 만들고 /bin/bash 실행

I: interactive, t:tty, d:daemon

$ docker ps -a

모든 컨테이너 목록 출력

Docker command에서 컨테이너를 가리킬 때 container id, 컨테이너 이름 모두 가능

Docker Commands

Attach / detach : 현재 stdin, stdout, stderr을 컨테이너의 것과 연결/연결 해제

$docker attach [name]

[name] 컨테이너와 attach

ctrl + p > ctrl + q

컨테이너에서 detach

$ docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

실행 중인 container에 명령어 전달

Docker - Exercise

$ docker attach hello

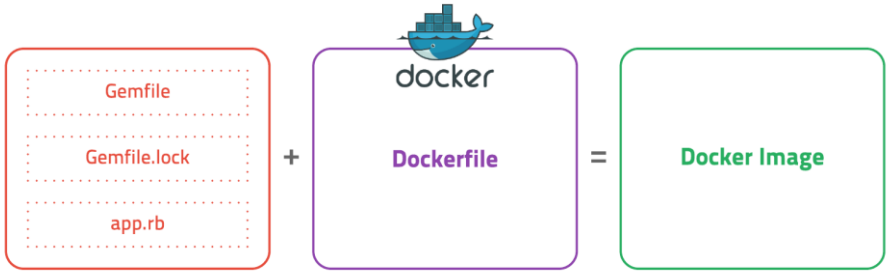
hello에 stdio 연결 (ctrl + p > ctrl + q 로 detach)

만약 exit, ctrl+d를 입력하면 컨테이너 정지

$docker exec -i -t hello /bin/bash

hello에서 /bin/bash를 실행하고 stdio 연결

Image 생성



- Dockerfile : 이미지를 생성하기 위한 파일
  - FROM (이미지명) : 특정 이미지로부터 시작
  - COPY (소스) (타겟) : 폴더/파일을 소스에서 파일로 복사
  - RUN (명령어) : 명령어를 실행
  - WORKDIR (폴더) : 특정 폴더로 이동
  - ENV (키)=(값) : 환경변수를 설정
  - ENTRYPOINT [“echo”, “hello”] : 컨테이너 구동 시 echo hello 실행

Image 생성

$ docker build --tag 이름:버전 .

이름:버전으로 도커 이미지 생성

$ docker commit (컨테이너) (이미지)

변경 사항을 이미지로 저장

$ docker images

이미지 확인

DockerHub

도커 이미지들이 올라오는 곳

https://hub.docker.com

매우 많은 이미지들이 미리 올라와있다!

Docker – Exercise

$ mkdir example

$ cd example

$ vim Dockerfile

```
FROM ubuntu:14.04
MAINTAINER Foo Bar foo@bar.com

RUN apt-get update
RUN apt-get install -y nginx
RUN echo "₩ndaemon off;" >> /etc/nginx/nginx.conf
RUN chown -R www-data:www-data /var/lib/nginx

VOLUME ["/data", "/etc/nginx/site-enabled", "/var/log/nginx"]

WORKDIR /etc/nginx

CMD ["nginx"]

EXPOSE 80
EXPOSE 443
```

Docker – Exercise

```
$ sudo docker build --tag hello:0.1 .
$ sudo docker images
$ sudo docker run --name hello-nginx -d -p 80:80 -v /root/data:/data hello:0.1
$ sudo docker ps
```

http://<호스트 IP>:80 접속!

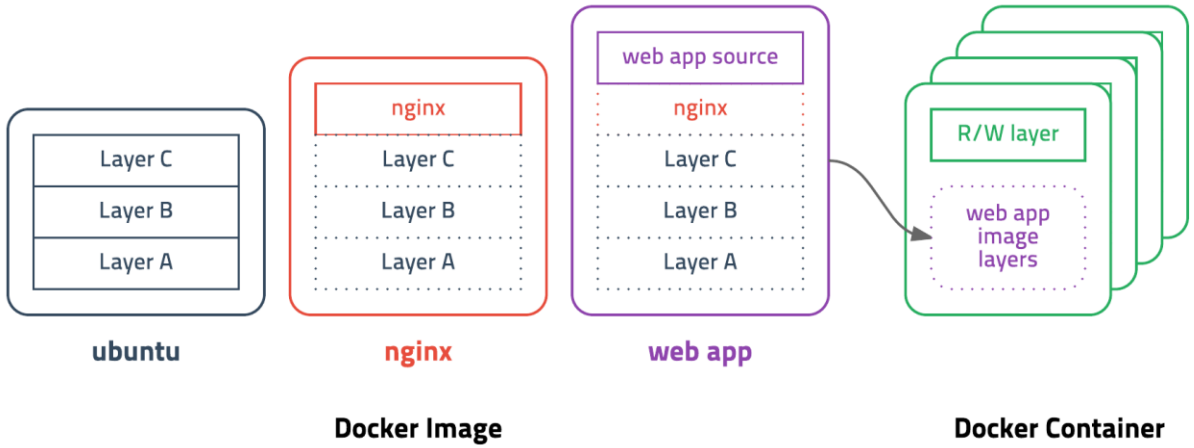
Union Mounting

Read-only 파일시스템들의 Layer를 쌓는 느낌

그러나 하나의 파일시스템처럼 보이게 됨

만약 소스가 수정되었다면 수정된 이후의 레이어만 다운받으면 됨 -> 효율적

ex) Alpine Linux + nodejs + ...



Docker

컨테이너가 삭제될 때 기본적으로 데이터도 삭제됨

따라서 특정 디렉토리를 호스트에 저장하는 옵션을 제공함 (-v)

Volume

도커가 관리하는 볼륨이 생성돼 데이터가 저장됨

/var/lib/docker/volume/ 에 존재

$ docker volume

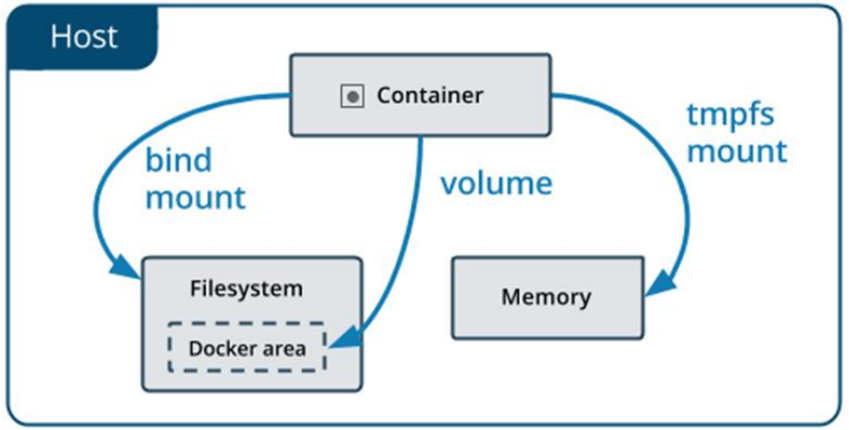
Non-Docker 프로세스들이 수정 불가

Bind mount

그 외 호스트의 어딘가를 컨테이너에 마운트함

Tmpfs mount

Host System의 Memory에만 Data가 저장



Networking

- bridge
  - 격리된 네트워크, docker0 브릿지 (기본)
  - expose : 포트를 노출 (-p 8080:80 : 8080포트를 80포트로 노출)
- host (--net=host)
  - 호스트와 같은 네트워크 사용
- container (--net=container:(hash))
  - 다른 컨테이너와 같은 네트워크 사용
- none
  - 격리만 된 상태

Reboot Policy

- 자동으로 컨테이너를 실행할 수 있음 (--restart)
- no
  - 자동 재시작을 사용하지 않음 (default)
- on-failure
  - 오류로 죽으면 재시작 (exit code 0이 아님)
- always
  - 항상 재시작 (직접 죽이면 도커 재시작시 같이 재시작)
- unless-stopped
  - always랑 같지만 stop되면 도커 재시작해도 같이 재시작 X

Docker Compose

Background

도커 아규먼트가 주렁주렁 달림

컨테이너 간의 의존성이 생김

설정파일을 하나 만들어서 그걸로 관리하자!

Docker Compose

How To

- yaml 파일을 하나 만들어서 설정을 저장
  - docker-compose.yml
- $ docker-compose up
  - $ docker-compose up -d : 데몬으로 실행
- $ docker-compose down

Docker Compose

```yaml
version: '3'
services:
        pydio:
                image: pydio/cells
                restart: always
                environment:
                        CELLS_BIND: 0.0.0.0:8080
                        CELLS_EXTERNAL: cloud.nenw.dev
                        CELLS_NO_SSL: 0
                ports:
                        - 3003:8080
                volumes:
                        - pydio:/root/.config/pydio/cells
                        - static:/root/.config/pydio/cells/static/pydio
                        - data:/root/.config/pydio/cells/data
                depends_on:
                        - db
                networks:
                        pydio:
                                aliases:
                                        - pydio

        db:
                image: mariadb
                restart: always
                environment:
                        MYSQL_ROOT_PASSWORD: [redacted]
                        MYSQL_DATABASE: pydio
                        MYSQL_USER: pydio
                        MYSQL_PASSWORD: [redacted]
                command: [mysqld, --character-set-server=utf8mb4, --collation-server=utf8mb4_unicode_ci]
                expose:
                        - "3306"
                volumes:
                        - db:/var/lib/mysql
                networks:
                        pydio:
                                aliases:
                                        - db

volumes:
        pydio:
        data:
        static:
        db:

networks:
        pydio:
```

Docker Compose - Exercise

설치

```
$ sudo apt install docker-compose
```

```
$ mkdir example2
$ cd example2
$ vim docker-compose.yml
```

```
$ docker-compose up -d
```

http://<호스트 IP> 접속!

```
version: '2'

services:
    wordpress:
        image: wordpress
        ports:
            - 80:80
        environment:
            WORDPRESS_DB_PASSWORD: example

    mysql:
        image: mariadb
        environment:
            MYSQL_ROOT_PASSWORD: example
```

Container Orchestration

- 컨테이너 배포 관리
  - 여러 컨테이너의 배포 프로세스 최적화
- Docker Swarm
- Kubernetes
- Apache Mesos