Container

- 특정 애플리케이션을 실행시키는데 필요한 모든 것들을 담아 놓은 소프트웨어
- 실행 환경에 상관없이 일관된 방식으로 애플리케이션을 배포하고 실행하는데 용이
- Namespace와 cgroup(컨트롤 그룹)을 기반으로 애플리케이션을 가상화하여 실행하는 방법
  cgroup : 프로세스별로 CPU 시간이나 메모리 사용량 같은 자원을 감시하고 제한
  Namespace 종류와 역할

네임스페이스	의미	역할
pid	PID: Process ID	리눅스 커널의 프로세스 ID 분리
net	NET: Networking	네트워크 인터페이스(NET) 관리
ipc	IPC: Inter Process Communication	프로세스 간 통신(IPC) 접근 관리
mnt	MNT: Mount	파일 시스템의 마운트 관리
uts	UTS: Unix Timesharing System	커널과 버전 식별자 분리

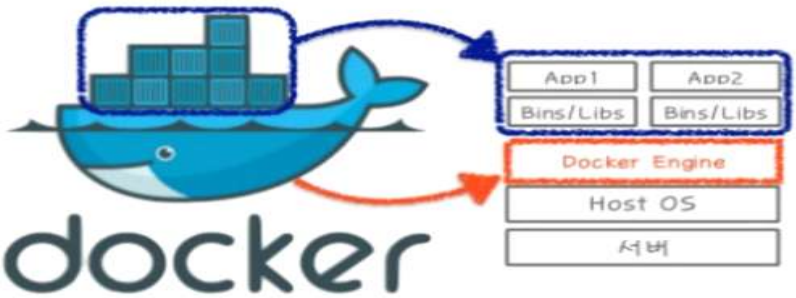
도커

- 도커(Docker)는 리눅스 컨테이너에 리눅스 어플리케이션을 프로세스 격리 기술을 사용하여 더 쉽게 컨테이너로 실행하고 관리할 수 있게 해주는 오픈소스 프로젝트이다.
- 도커는 일반적으로 도커 엔진(Docker Engine) 혹은 도커에 관련된 모든 프로젝트를 말한다.
- 애플리케이션과 해당 애플리케이션에 종속성을 포함하는 실행 환경을 제공한다.
- Dockerfile을 사용해서 라이브러리, 패키지, 코드 등과 함께 이미지를 생성한다.
- Docker Hub와 같은 컨테이너 레지스트리를 통해서 이미지를 관리 및 공유한다.

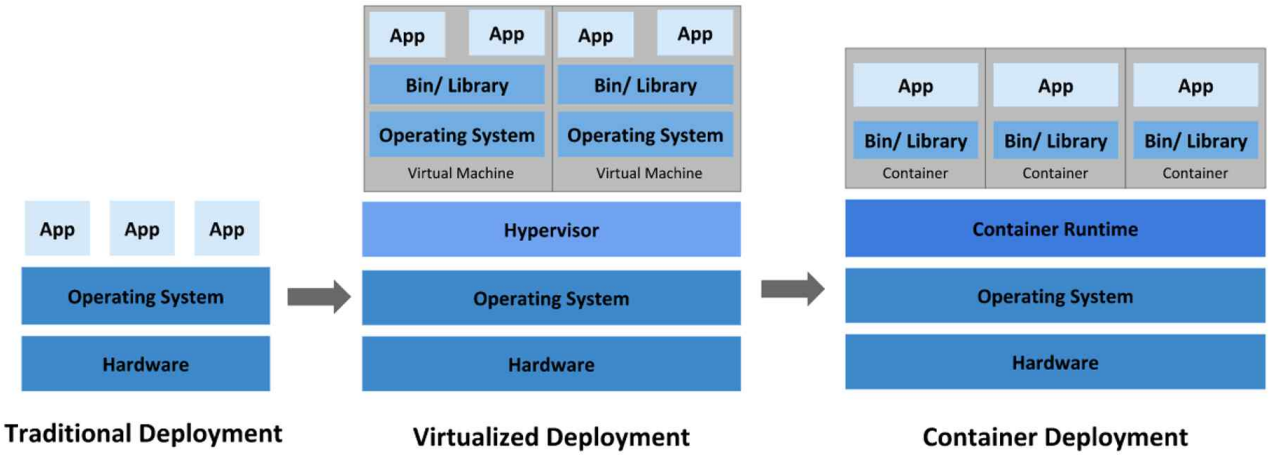
도커 엔진(Docker Engine)

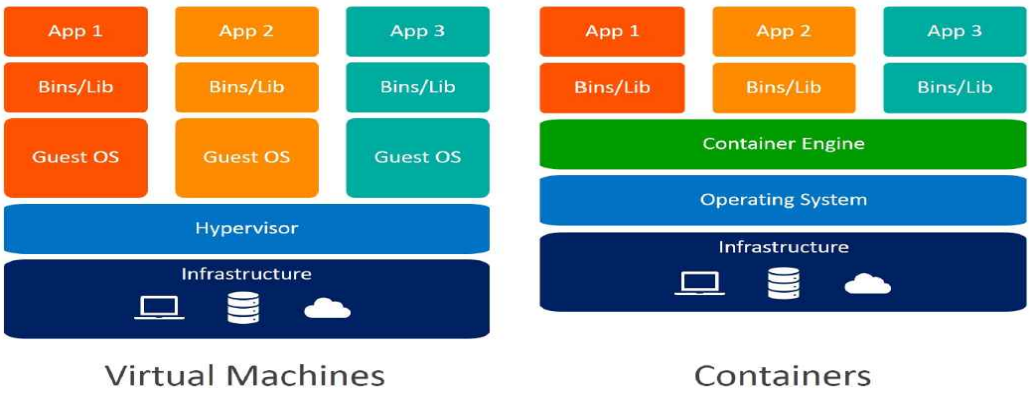
- 도커 엔진(Docker Engine)은 컨테이너를 생성하고 관리하는 주체로서 이 자체로도 컨테이너를 제어할 수 있고 다양한 기능을 제공하는 도커의 프로젝트이다.

도커 생태계에 있는 여러 프로젝트는 도커 엔진을 좀 더 효율적으로 사용하기 위한 것에 불과하므로 도커의 핵심은 도커 엔진이라고 할 수 있다.



Virtual Machine(가상머신) vs Docker Container(도커 컨테이너)





Virtual Machine(가상머신)

- 가상머신은 Hypervisor를 통해 여러 개의 운영체제를 생성하고 관리한다. (Guest OS)
- 시스템 자원을 가상화하고 독립된 공간을 생성하는 작업은 HyperVisor를 거치므로 성능 손실이 크다.
- 가상머신은 Guest OS를 사용하기 위한 라이브러리, 커널 등을 포함하므로 배포할 때 용량이 크다.

Docker Container(도커 컨테이너)

- 도커 컨테이너는 가상화된 공간을 생성할 때 리눅스 자체 기능을 사용하여 프로세스 단위의 격리 환경을 만들므로 성능 손실 없다.
- 가상머신과 달리 커널을 공유해서 사용하므로, 컨테이너에는 라이브러리 및 실행파일만 있으므로 용량이 작다.
- 위의 이유로 컨테이너를 이미지로 만들었을 때 배포하는 시간이 가상 머신에 비해 빠르며, 사용할 때의 성능 손실 또한 거의 없다.

Docker Client

- 도커를 설치하면 그것이 Client며 build, pull, run 등의 도커 명령어를 수행한다.

DOCKER HOST

- 도커가 띄워져 있는 서버를 의미한다. DOCKER HOST에서 컨테이너와 이미지를 관리하게 된다.

Docker daemon

- 도커 엔진이다.

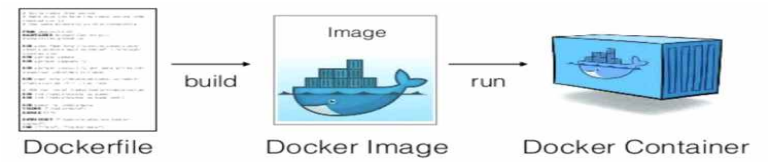
Registry

- 외부(remote) 이미지 저장소이다.
  다른 사람들이 공유한 이미지를 내부(local) 도커 호스트에 pull 할 수 있다.
  이렇게 가져온 이미지를 run하면 컨테이너가 됩니다.

public 저장소 : Docker Hub, QUAY
private 저장소 : AWS ECR 혹은 Docker Registry를 직접 띄워서 비공개로 사용하는 방법 등이 존재

도커 이미지와 도커 컨테이너

- 도커 엔진에서 사용하는 기본 단위는 이미지와 컨테이너이며 도커 엔진의 핵심이다.
- 도커 이미지와 컨테이너는 1:N 관계입니다.
  객체지향 프로그래밍에서의 클래스와 인스턴스의 관계와 비슷하다고 생각하면 된다.
- Docker File은 도커 이미지를 만들 때 사용하는 파일이다.
  docker build 명령어를 실행시키면 도커 이미지를 만들 수 있다.
- Docker Image를 docker run 명령어로 실행시키면 Docker Container를 만들 수 있다.



Dockerfile

- Dockerfile은 컨테이너 이미지를 만들기 위한 설명서
- Dockerfile에 기술된 명령어를 차례대로 실행해서 컨테이너 이미지를 생성
- 작성된 Dockerfile은 'docker build' 명령어에 의해서 사용
- 'docker push' 명령어를 통해서 컨테이너 레지스트리에 업로드

도커 이미지

- 도커 이미지(Docker Image)는 컨테이너를 생성할 때 필요한 요소이며, 가상 머신을 생성할 때 사용하는 iso 파일과 비슷한 개념이다.
- 이미지는 컨테이너를 생성하고 실행할 때 읽기 전용으로 사용되며 여러 계층으로 된 바이너리 파일로 존재한다.
- 도커에서 사용하는 이미지의 이름은 기본적으로 아래의 형태로 구성된다.

[형식]

[저장소 이름]/[이미지 이름]:[태그]

저장소 이름: 이미지가 저장된 장소.

저장소 이름이 명시되지 않은 이미지는 도커 허브의 공식 이미지를 뜻한다.

이미지 이름: 해당 이미지가 어떤 역할을 하는지 나타내며 필수로 설정해야 함.

ubuntu:latest -> 우분투 컨테이너를 생성하기 위한 이미지라는 것을 나타낸다.

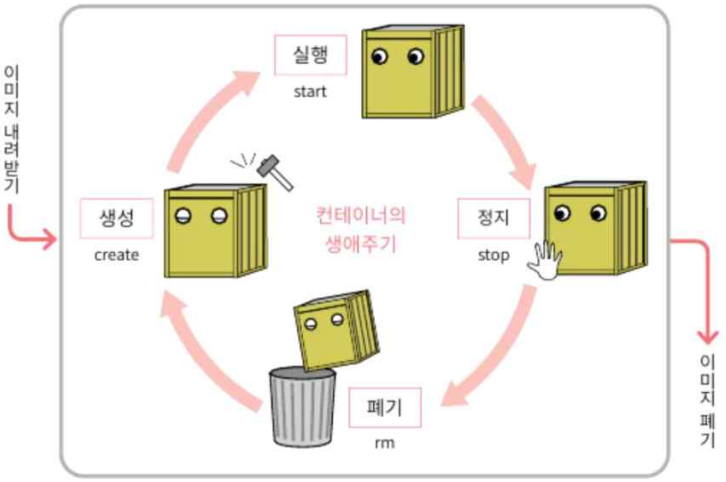
태그: 이미지의 버전을 나타낸다.

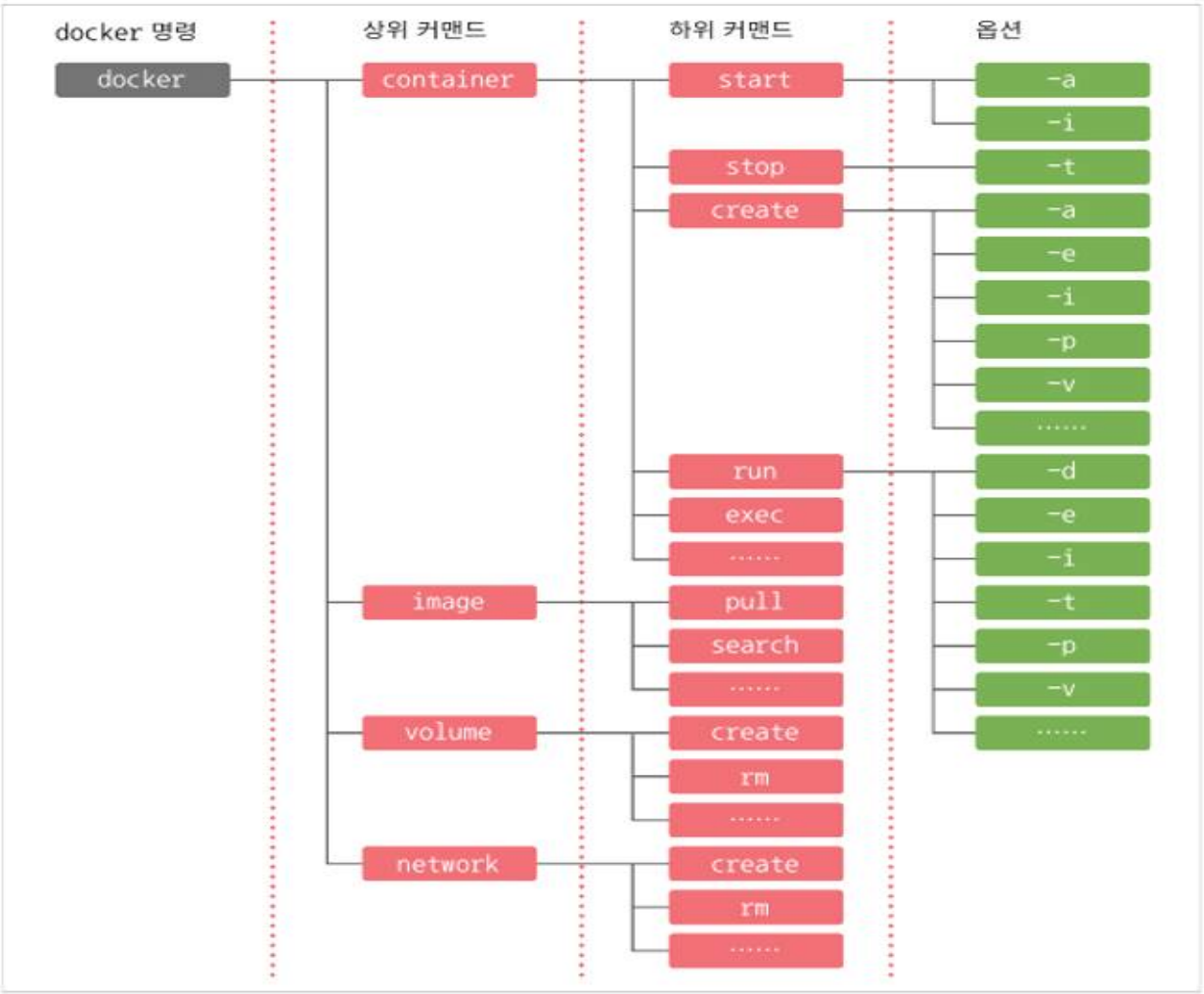
태그를 생략하면 도커 엔진은 latest로 인식함.

도커 컨테이너

- 도커 컨테이너(Docker Container)는 도커 이미지로 생성할 수 있으며, 컨테이너를 생성하면 해당 이미지의 목적에 맞는 파일이 들어 있는, 호스트와 다른 컨테이너로부터 격리된 시스템 자원 및 네트워크를 사용할 수 있는 독립된 공간(프로세스)이 생성된다.

- 컨테이너는 이미지를 읽기 전용으로 사용하되 이미지에서 변경된 사항만 컨테이너 계층에 저장하므로 컨테이너에서 무엇을 하든지 원래 이미지는 영향을 받지 않는다.
또한 생성된 각 컨테이너는 각기 독립된 파일시스템을 제공받으며 호스트와 분리돼 있으므로 특정 컨테이너에서 어떤 어플리케이션을 설치하거나 삭제해도 다른 컨테이너와 호스트는 변화가 없다.
예를 들어 같은 도커 이미지로 A, B 두 개의 컨테이너를 생성한 뒤에 A 컨테이너를 수정해도 B 컨테이너에는 영향을 주지 않는다.


이미지 내려받기
실행
start
생성
create
컨테이너의
생애주기
정지
stop
폐기
rm
이미지 폐기


docker 명령
상위 커맨드
하위 커맨드
옵션
docker
container
start
-a
-i
stop
-t
create
-a
-e
-i
-p
-v
.....
run
-d
-e
exec
-i
.....
-t
image
pull
-p
search
-v
.....
volume
create
rm
.....
network
create
rm
.....