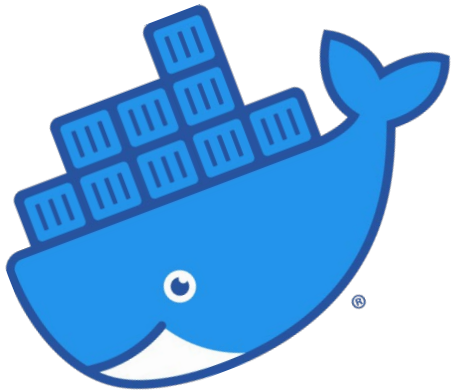
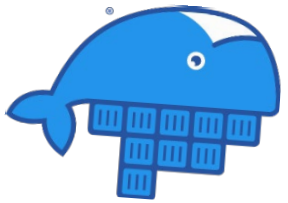
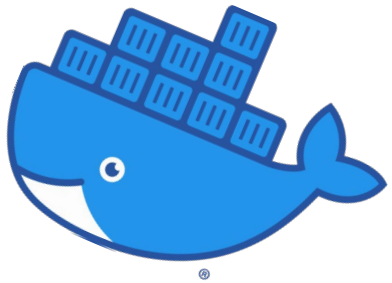




오픈소스 도커 컨테이너(Docker Container) 활용과 문제해결

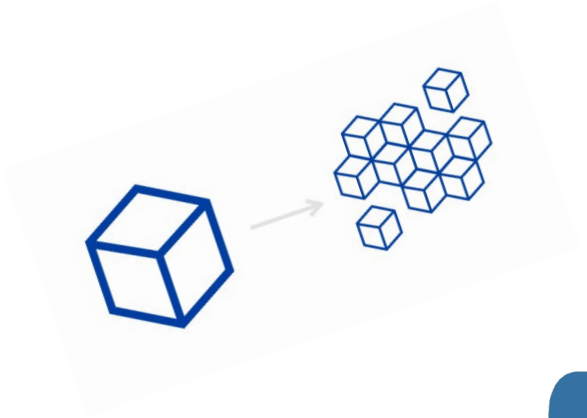


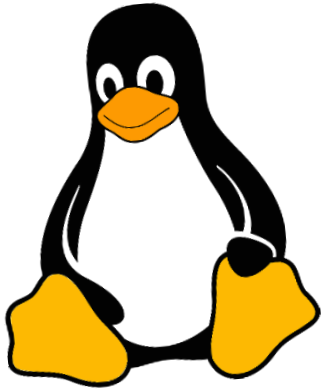
이성미
seongmi.lee@gmail.com

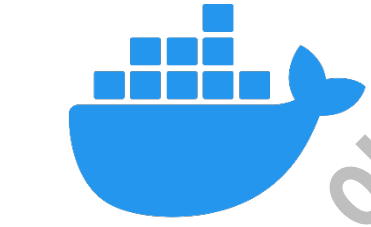
docker

이성미(SEONGMI LEE)

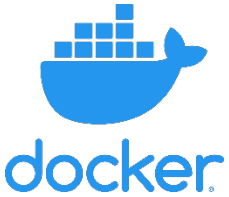
seongmi.lee@gmail.com

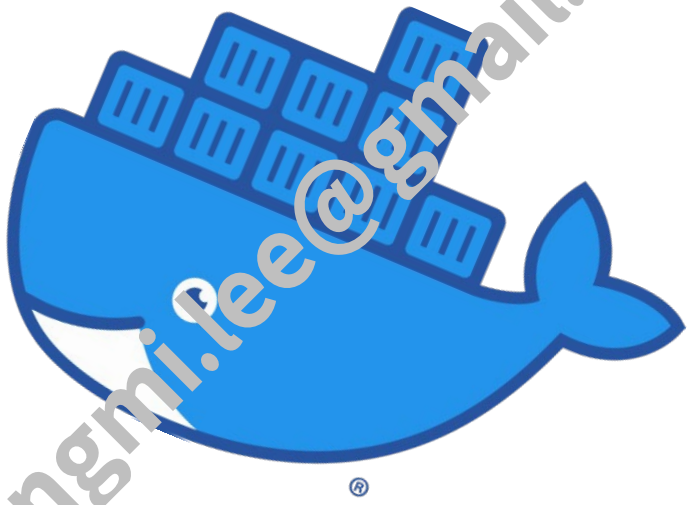




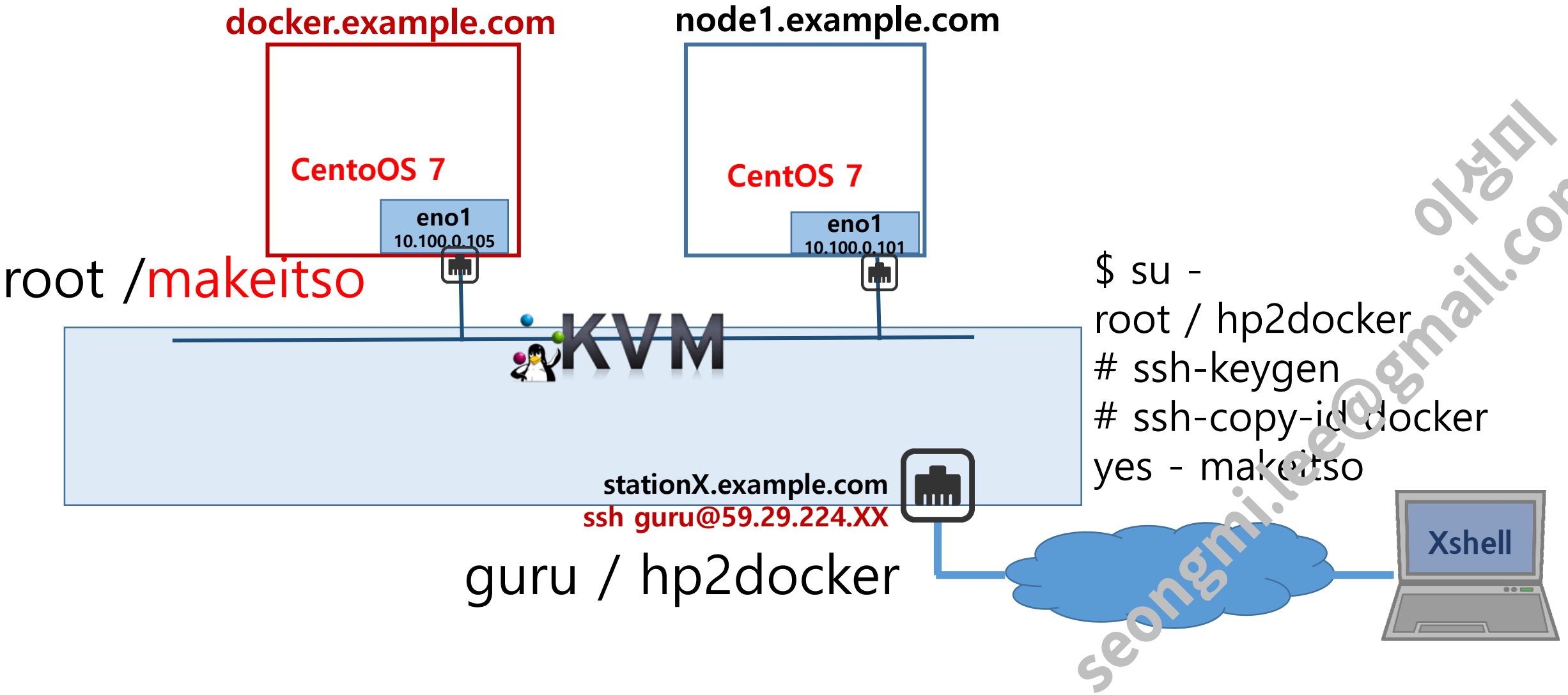


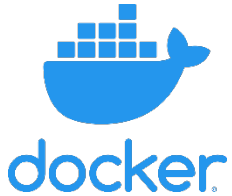
Q: 뭘 알려주나요?





Q 실습환경?





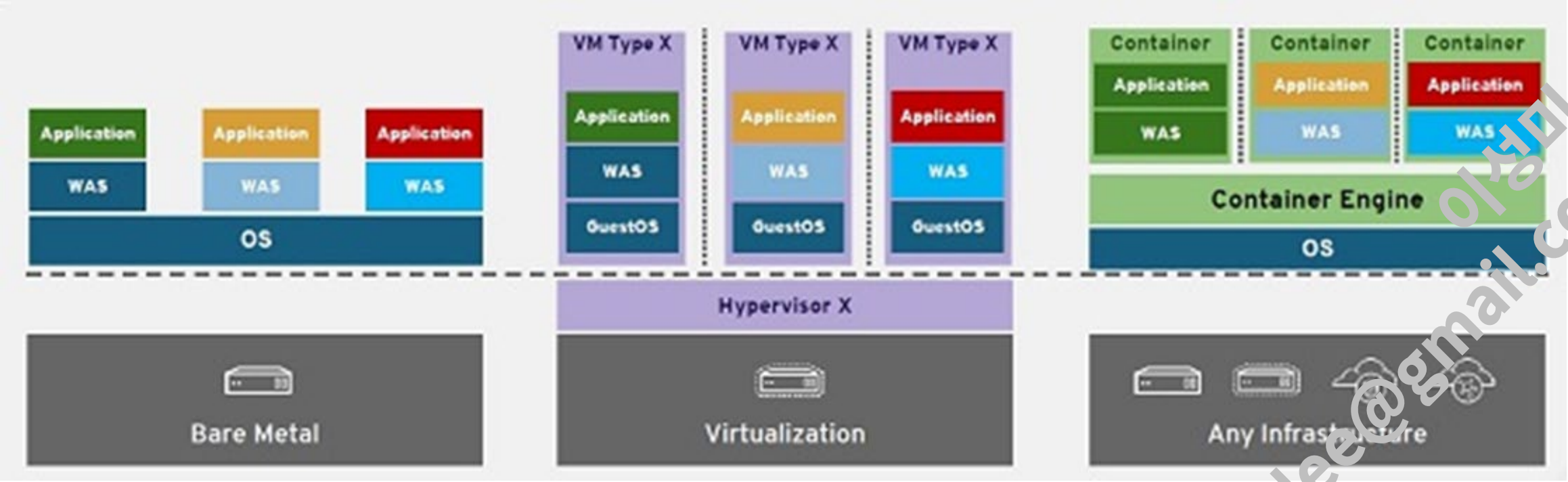
1. CONTAINER TECHNOLOGY

The What and Why of Containers?

- What containers?
  - 소프트웨어 개발, 배포, 실행방식이 달라짐

- Why Containers?
  - resources
  - portability
  - lightweight
  - deploying

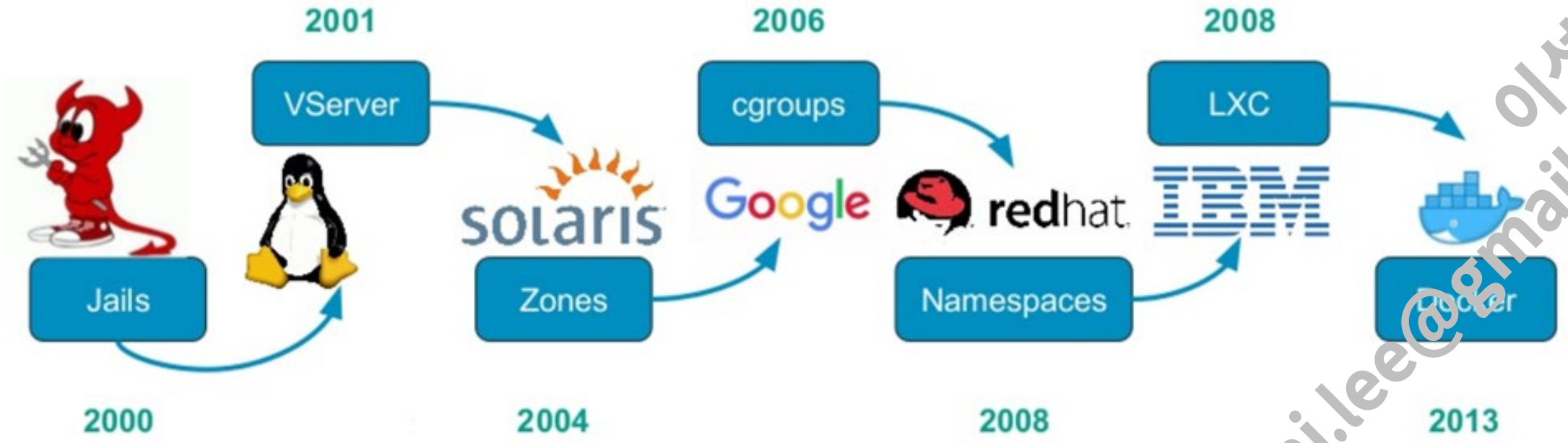
Dedicated machines, Virtual Machines and Containers



https://www.datanet.co.kr/news/articleView.html?idxno=131759

Container Timeline

컨테이너가 뜨는 이유 : 리눅스 커널 레벨에서 컨테이너 기능을 지원



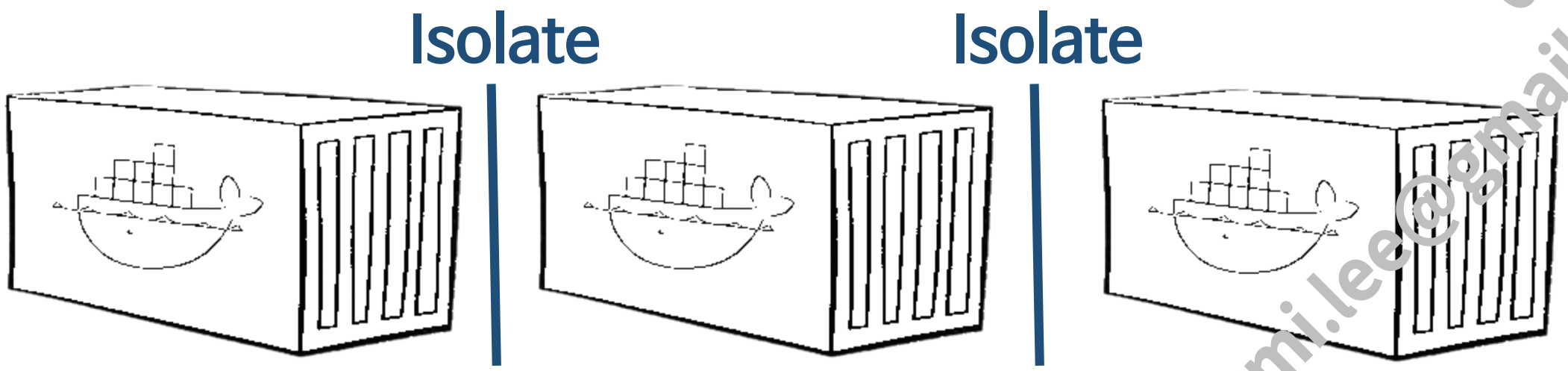
https://pt.slideshare.net/insideHPC/linux-container-technology-101/3

Container Timeline

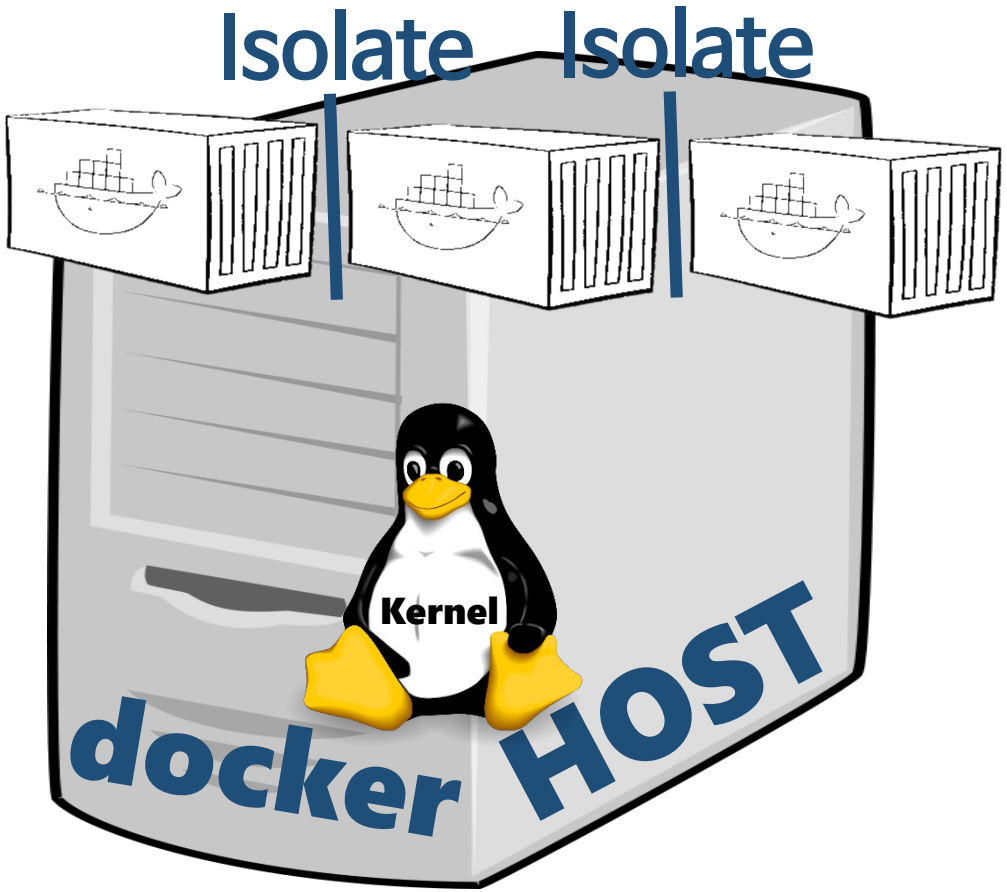
- What containers?
  - 소프트웨어 개발, 배포, 실해방식이 달라짐
- Why Containers?
  - resources
  - portability
  - lightweight
  - deploying

Containers?

- 컨테이너는 하나의 Application 프로세스
- 컨테이너 화 된 프로세스는 커널을 공유



Containers and Linux Kernel

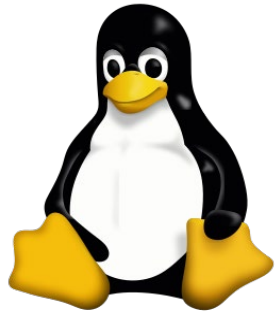


왜 굳이 리눅스에서 돌려요?

chroot

namespace

cgroup



kernel

chroot: Traditional Isolation Techniques

- isolates itself to a specific directory on the filesystem
- use chroot
  - ex: DNS server named, ntpd, and the Postfix email server
- an attacker with root privileges can escape it

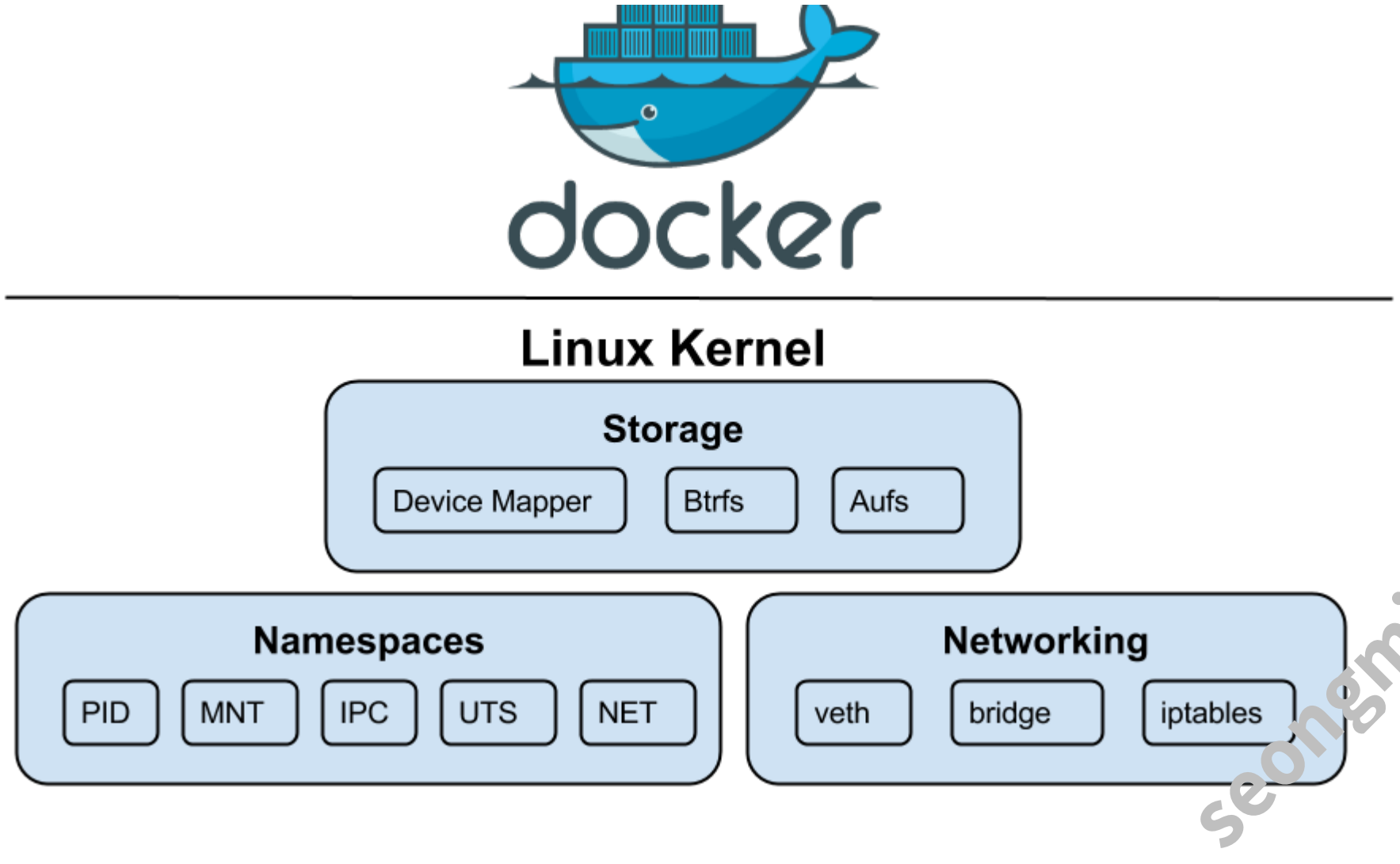
Linux Namespaces

- 특정 리소스의 가시성(visibility)을 제한
- global 리소스의 격리된 인스턴스(isolated instance)
- namespace 종류
  - IPC
  - Network
  - Mount
  - PID
  - UTS
  - User
  - Cgroup(Kernel 4.6 이상)
  - Time(Kernel 5.6)

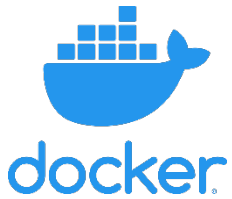
Control Groups

- Resource 측정과 제어
  - CPU 시간, 시스템 메모리, I/O 대역폭 등을 할당하고 제어
  - Systemd는 기본적으로 cgroup을 사용

Linux Kernel & Container

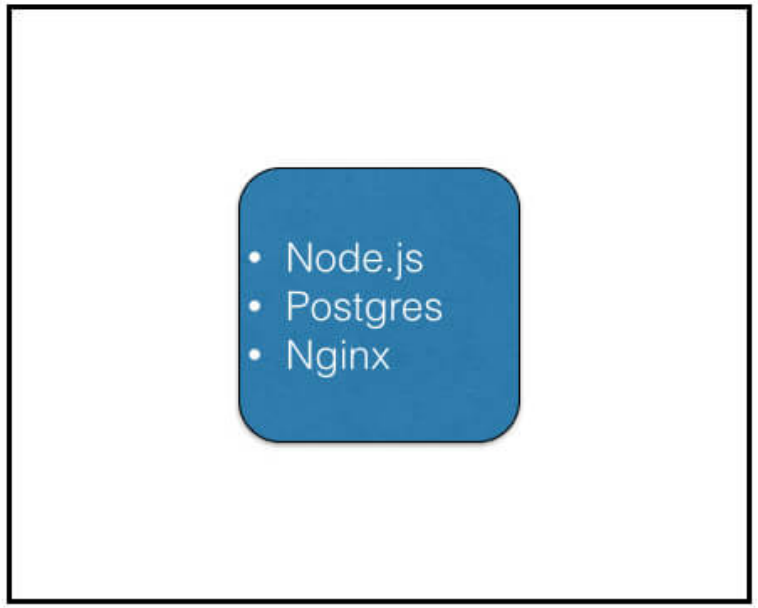


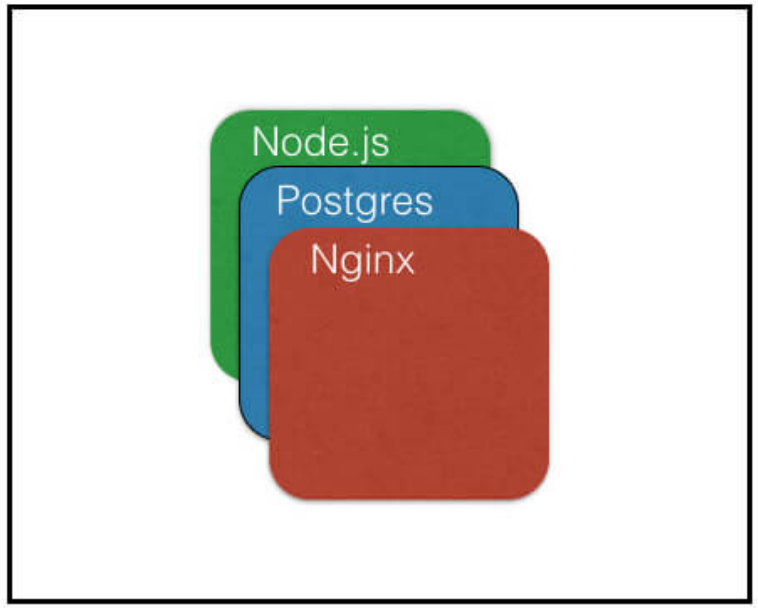
LXC vs Docker Container



- 초기 Docker 버전
  - LXC API를 사용하는 LXC를 기반 + 이동성(portability)과 패키지 기능
- Docker 버전 0.9
  - LXC 의존성 탈피. 커널 API를 새로 작성 → libcontainer → runC
- Docker와 LXC의 차이
  - LXC : init 프로세스가 있어 여러 프로세스를 실행 가능
  - Docker : 컨테이너가 단일 프로세스로 실행

OS Containers vs Application Containers





OS containers

- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones

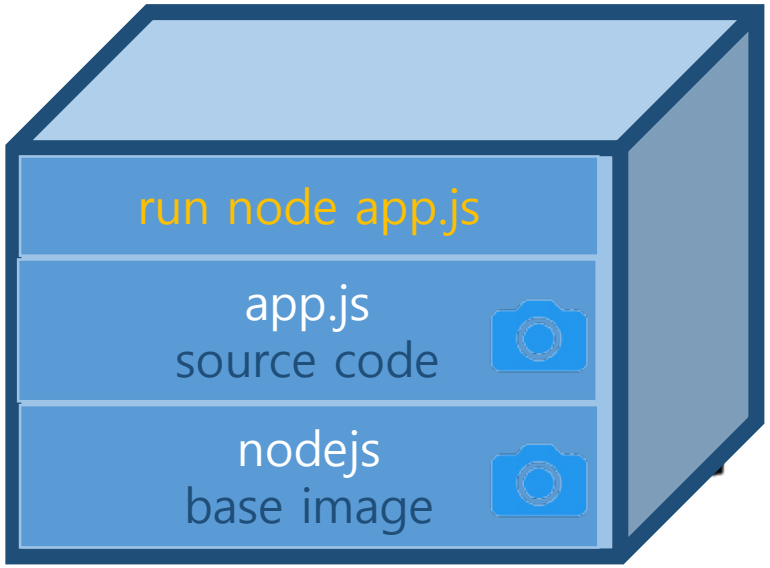
App containers

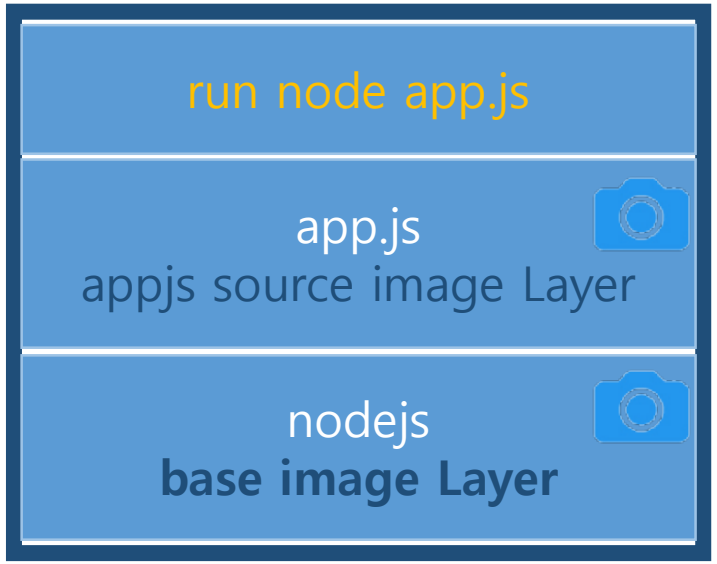
- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

https://piyushchaudhariblog.wordpress.com/

컨테이너? 컨테이너 이미지?

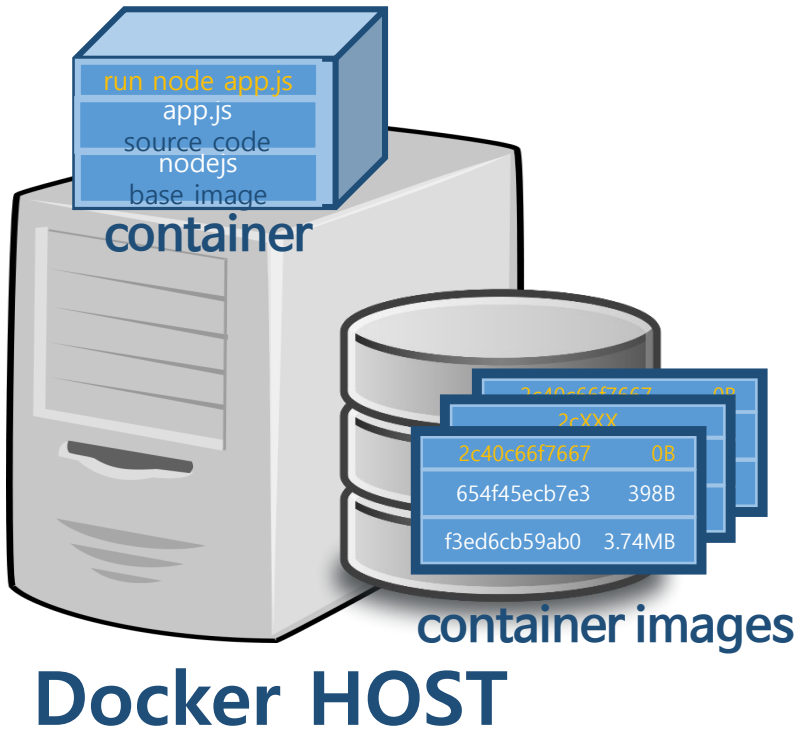
- 이미지
  - Docker 컨테이너 생성하는 instruction을 포함하는 읽기 전용 템플릿
- 컨테이너
  - 이미지의 실행 가능한 instants
  - 컨테이너를 create, start, stop 또는 delete 할 수 있음.



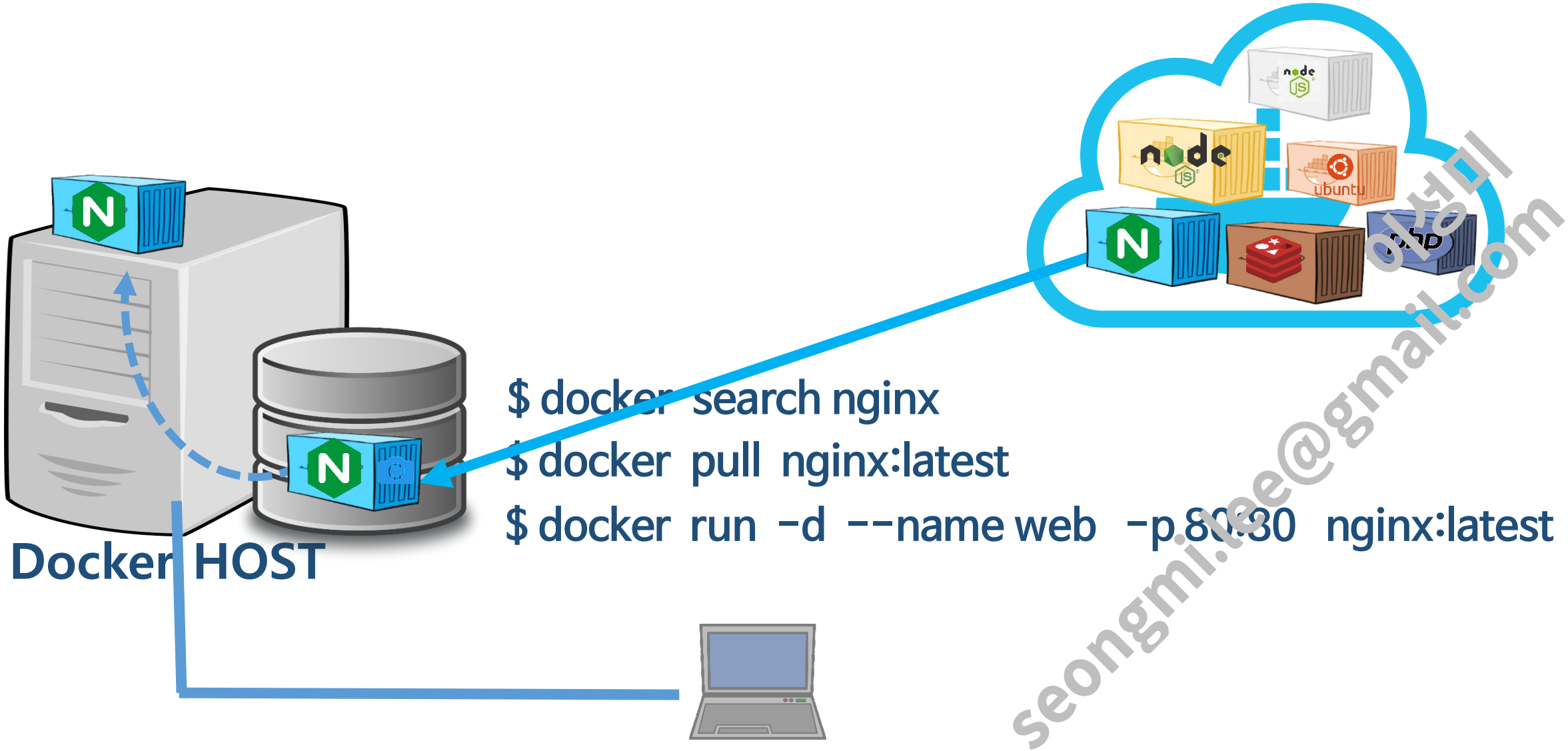


컨테이너? 컨테이너 이미지?

- Container Image
- Container

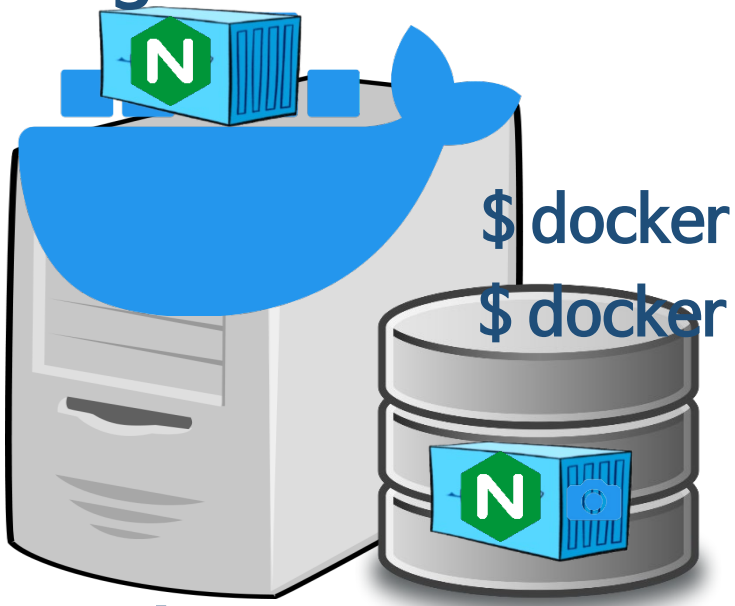


컨테이너 동작방식



Docker Architecture

- Docker Host(Linux Kernel)
- Docker Daemon : systemctl start docker
- Docker Client Command : docker
- Docker Hub
- Container Images
- Container



$ docker search nginx

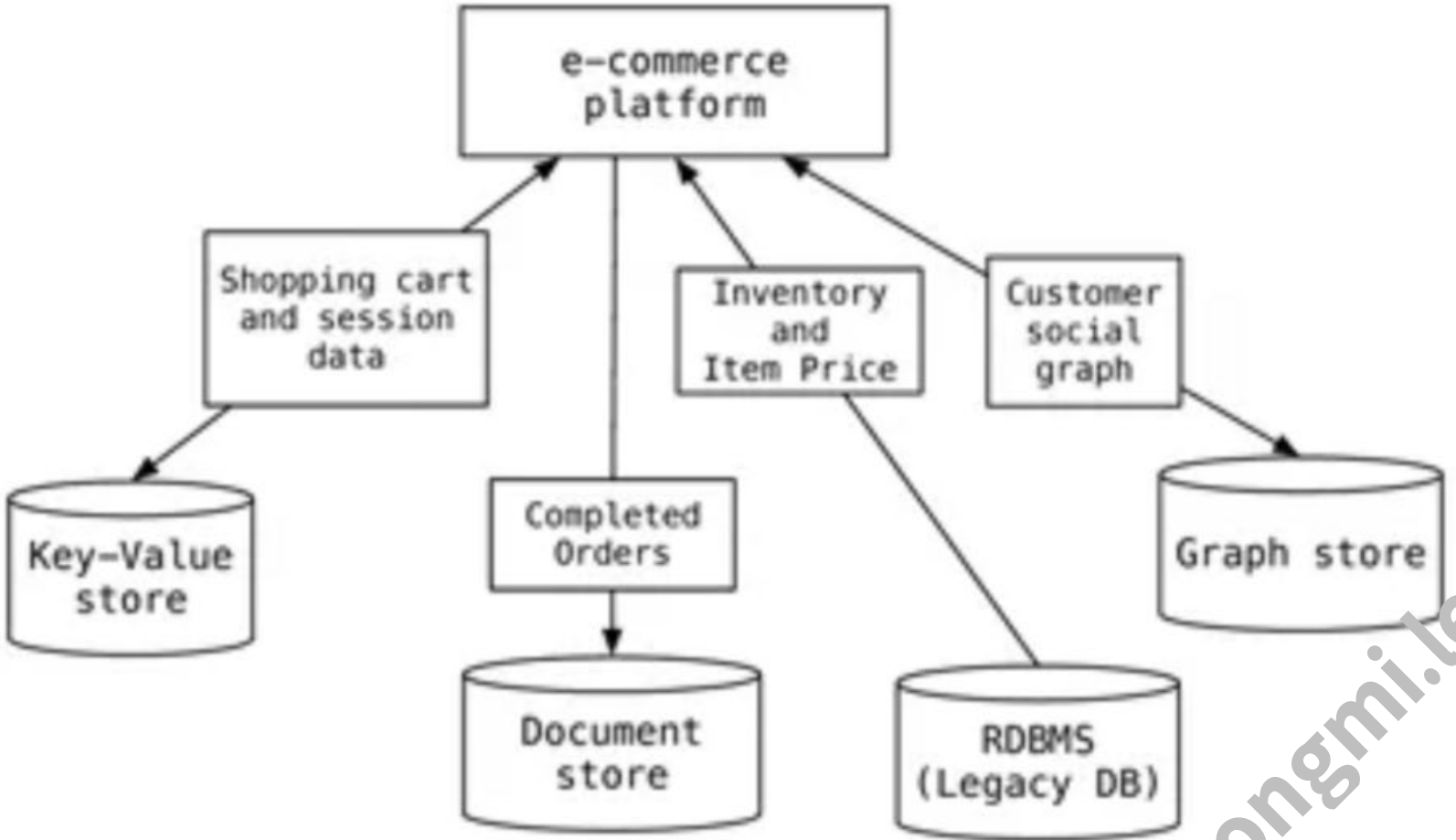
$ docker pull nginx:latest

Docker HOST

Microservice vs Monoliths

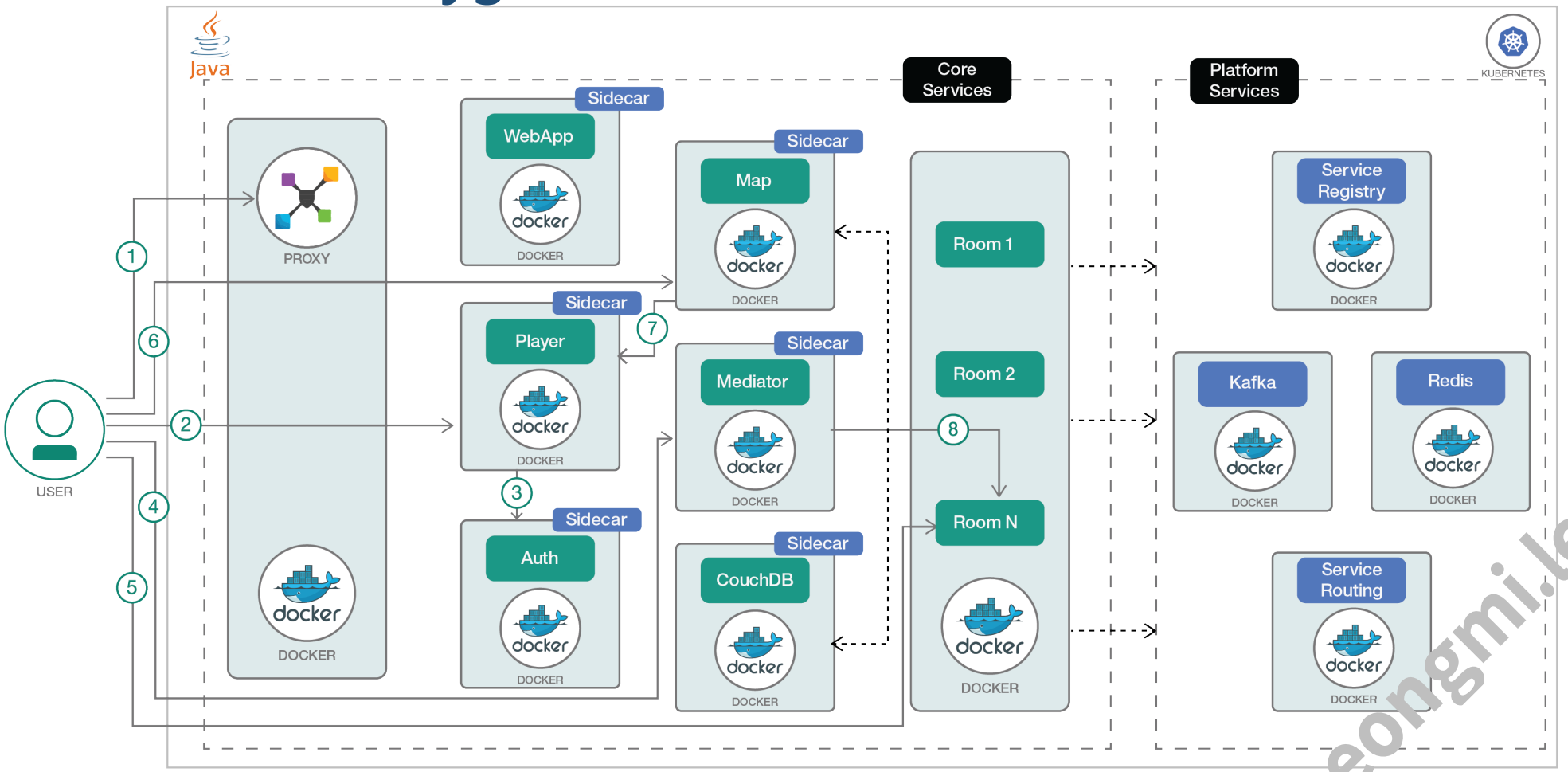
- Microservice
  - 모든 서비스를 개별 요소로 구성
  - 상황에 따라 필요한 리소스만 집중해서 확장: scale out
  - APP container
- Monoliths
  - 소프트웨어 개발과 운영에 대한 전통적인 방법
  - 상황에 따라 전체 시스템을 확장 : scale up
  - OS container and Virtual Machine

무엇을 컨테이너로 만들어야 하나?



무엇을 컨테이너로 만들어야 하나?

- 폴리글랏(Ployglot)



Docker user 추가

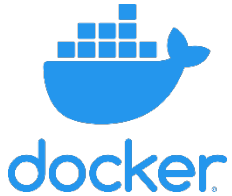
- guru 사용자가 docker 관리자가 되도록 구성
- guru 사용자가 docker group 의 member가 되도로 구성

```
# usermod  -a -G docker  guru
# id guru
uid=1000(guru) gid=1000(guru) groups=1000(guru),981(docker)

# su - guru
$ docker version
Client
...
Server
...
```

실습

1. KVM 구성(Optional)
2. Linux Kernal Update



2. INSTALLING DOCKER

도커 설치 플랫폼

- Installing Docker
  - Standard Linux
  - Microsoft Windows, Mac OS X, or others as a VM
  - Container-specific Linux

Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.

Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.

Docker for Linux

Install Docker on a computer which already has a Linux distribution installed.

https://docs.docker.com/get-docker/

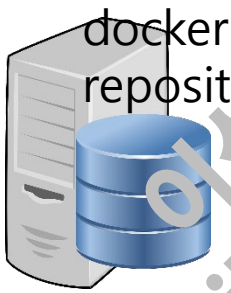
standard Linux에 Docker 설치하기

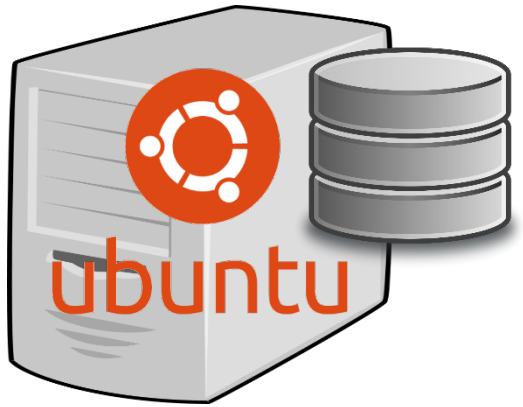
- CentOS와 Ubuntu에 Docker 설치
  https://docs.docker.com/

- 설치방법
  - Repository를 이용해서 설치
  - Download 후 직접설치
  - Script를 이용한 설치

- 설치 후 동작 상태 확인

- 계정 추가





Windows 10(WSL2)에 DockerDesktop 설치하기

- hub.docker.com 계정 등록
- DockerDesktop 설치
  - Hyper-V 가상화 기능활성화
  - WSL2(Windows Subsystem for Linux v.2)의 리눅스 커널 설치
- Docker 동작 상태 확인

Giving non-root access

- docker group의 member로 계정 생성 시 docker client 명령 실행 가능

```
# useradd -a -G docker dockeradmin
# usermod -a -G docker dockeradmin
```

Verifying Installation

- 도커 설치 확인 후 간단한 정보 보기

  $ docker version

  $ docker info

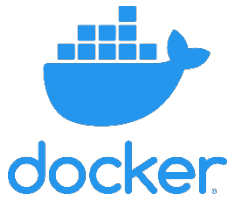
  $ docker run ubuntu:20.04 echo Hello World!

- 컨테이너 isolate 확인
  - PID, user, mount, network, uts, ipc

  $ docker run --name web -d nginx:1.14

  $ docker run --name c1 -it ubuntu:20.04

실습: Docker 설치 및 기본구성

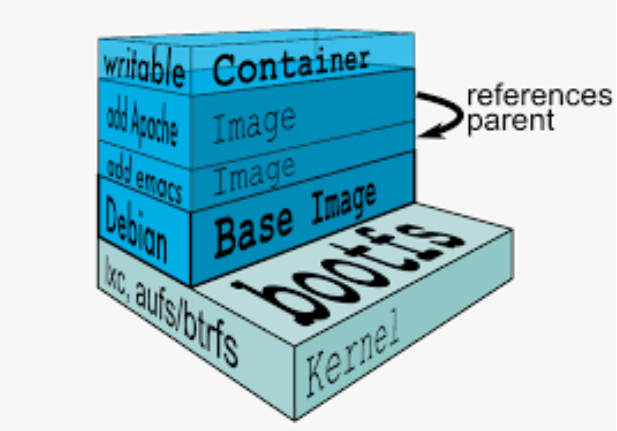


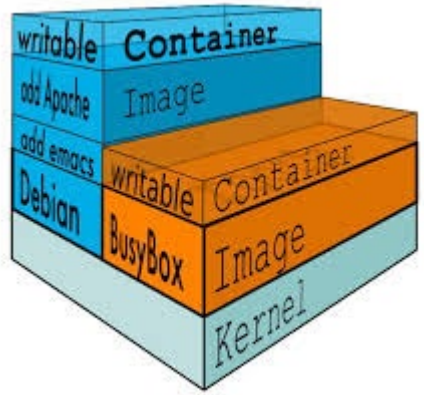
1. 서비스 정보 확인
2. 도커 데몬 제어
3. 사용자 계정에 권한 부여

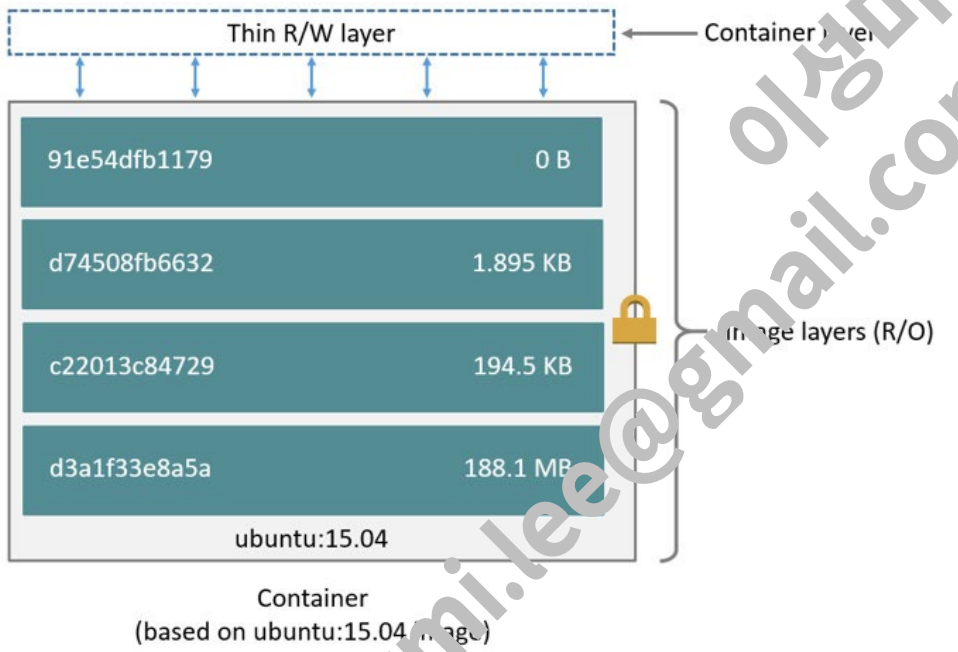
3. RUNNING CONTAINER IMAGES

이미지와 컨테이너의 이해

- 읽기 전용 템플릿으로 컨테이너 인스턴스를 저장한 파일
- overlay라는 레이어 파일 시스템을 사용







https://docs.docker.com/storage/storagedriver/

- Dockerfile로 자신만의 이미지 생성
- 컨테이너 이미지 관리
  - 이미지 다운로드 : docker image pull
  - 이미지 삭제 : docker image rm
  - 이미지 목록보기 : docker image ls

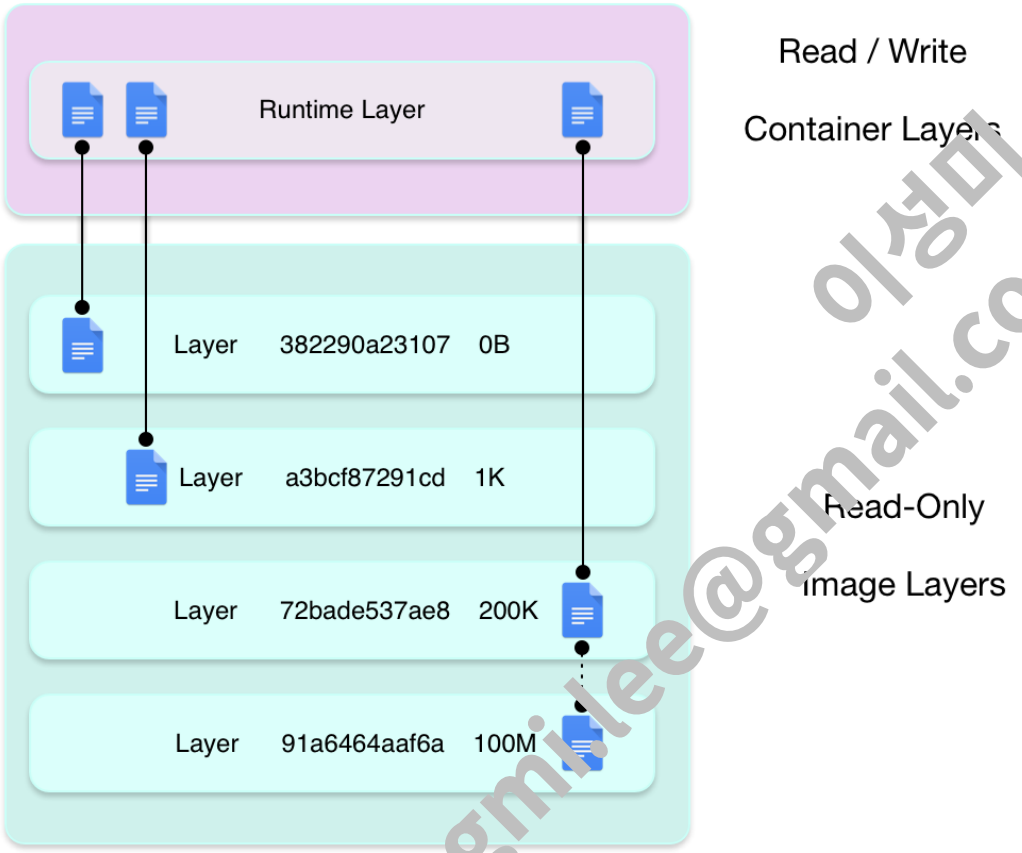
컨테이너 파일 시스템

- UFS (UnionFS)

  $ docker images

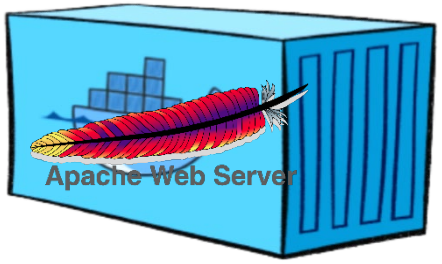
  $ docker image inspect nginx

  # ls /var/lib/docker/overlay2/

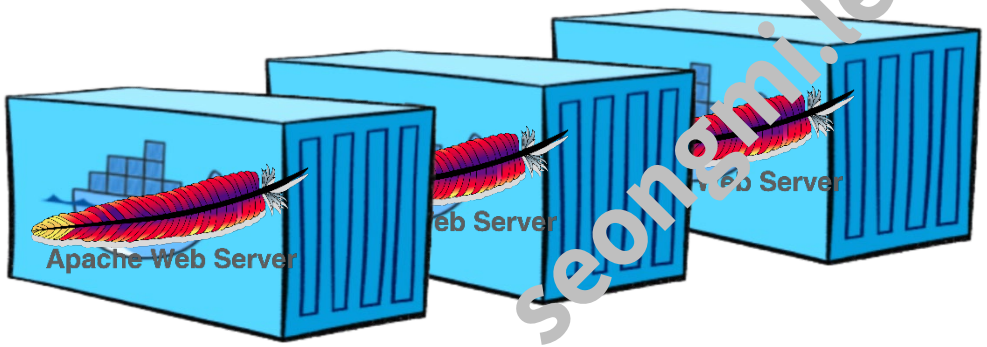


컨테이너로 구성한 서비스 실행

- 컨테이너로 구성한 서비스 실행의 장점
- 구성
  - 서비스 제공에 필요한 모든 요소를 사전에 구현
  - 컨테이너를 실행할 호스트 구성과 상관없이 어떤 호스트에도 서비스 배포 가능



- 분리
  - 각 컨테이너는 자신의 파일시스템과 네트워크 인터페이스를 가지므로 동일 서비스를 여러 개 구성 가능



컨테이너 실행 프로세스

docker container run 〈option〉 container_image 〈command〉

```
$ docker run -d --name web nginx
$ docker run --name c1 ubuntu:18.04   cat /etc/os-release
```

- run command 프로세스
  - 로컬 시스템에서 이미지 검색
  - 레지스트리에서 이미지 다운로드
  - 이미지를 사용하여 컨테이너 인스턴스 실행
  - 컨테이너에서 명령을 실행

컨테이너 실행과 종료

- 컨테이너 생성 : docker container create
- 컨테이너 생성 및 구동 : docker container run
- 컨테이너 구동 : docker container start
- 컨테이너 중지 : docker container stop
- 컨테이너 삭제 : docker container rm
- 컨테이너 목록 확인 : docker container ps

Runtime mode – detached와 foreground

- background mode: detached
  - docker run 명령에 -d 옵션을 사용
  - 컨테이너를 시작시킨 루트 프로세스가 종료되면 컨테이너도 종료

    $ docker run -d --name web nginx
- foreground mode: attach
  - 컨테이너 실행 시 기본 모드

    $ docker run -it ubuntu

컨테이너 관리

- 컨테이너의 접속: docker container attach

  컨테이너 탈출 : Ctrl+p,q
- 컨테이너 프로세스 실행 : docker container exec
- 컨테이너 프로세스 정보보기 : docker container top
- 컨테이너의 표준 출력 결과 보기: docker container logs
- 컨테이너 이름 변경 : docker container rename
- 컨테이너 내에서 파일 복사 : docker container cp
- 컨테이너 내에서 파일 변경이력 확인 : docker container diff

PID와 UTS 네임스페이스

- 컨테이너에 대한 PID 네임스페이스 모드 옵션:
  - --pid="" : 컨테이너에 대한 PID (Process) 네임스페이스 설정
  - --pid=container:id : 다른 컨테이너의 PID 네임스페이스에 연결
  - --pid=host : 호스트의 PID 네임스페이스를 사용

```
$ docker run --pid=host --name c1 -it  ubuntu
$ docker run --pid=container:web --name c2 -it  ubuntu
```

- --uts=host : docker 호스트의 호스트이름을 컨테이너에서 사용

```
$ docker run --uts=host --name c3 -it  ubuntu
```

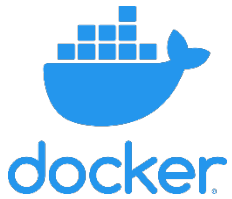
Container LAYER 관리

- 이미지는 overlay라는 레이어 파일 시스템을 사용
- 이미지 생성 프로세스를 레이어로 기록
- 특정 이미지의 레이어 목록 보기: docker image history
- 컨테이너에서 이미지 생성: docker container commit
- 이미지를 단일 레이어로 생성: docker container export
- 단일레이어 파일 import: docker image import

이미지 태그 관리

- docker build나 docker commit 명령으로 새로운 이미지를 생성한 후에,
  이미지 태그 수정이나 추가: docker tag
  - 이미지 버전
  - 이미지 버전 이름
  - latest: 태그를 지정하지 않는 경우 이미지의 기본 태그 값
- 이미지 태그 제거 : docker rmi
  - docker rmi: 이미지에 태그가 여러 개일 경우 태그 삭제

실습: docker의 기본 명령어 연습



1. 간단한 docker command 실습
2. 컨테이너로 이미지 생성하기
3. 이미지에 이름 할당하기
4. 이미지에 태그 할당하기

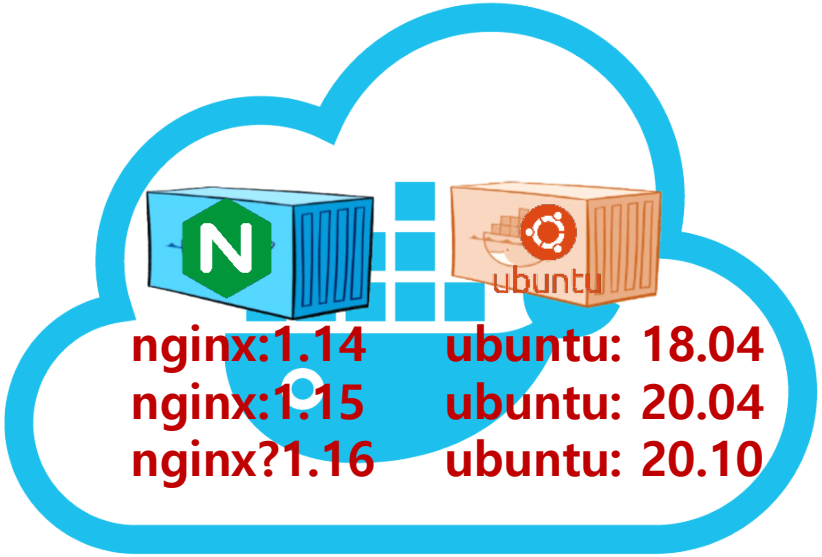
4. CONTAINER REPOSITORY

Docker Registry

- 컨테이너 이미지 저장소
- 저장소 유형
  - Docker Hub 레지스트리: Docker가 공식적으로 제공하는 저장소
  - 프라이빗 레지스트리: 오픈소스 애플리케이션

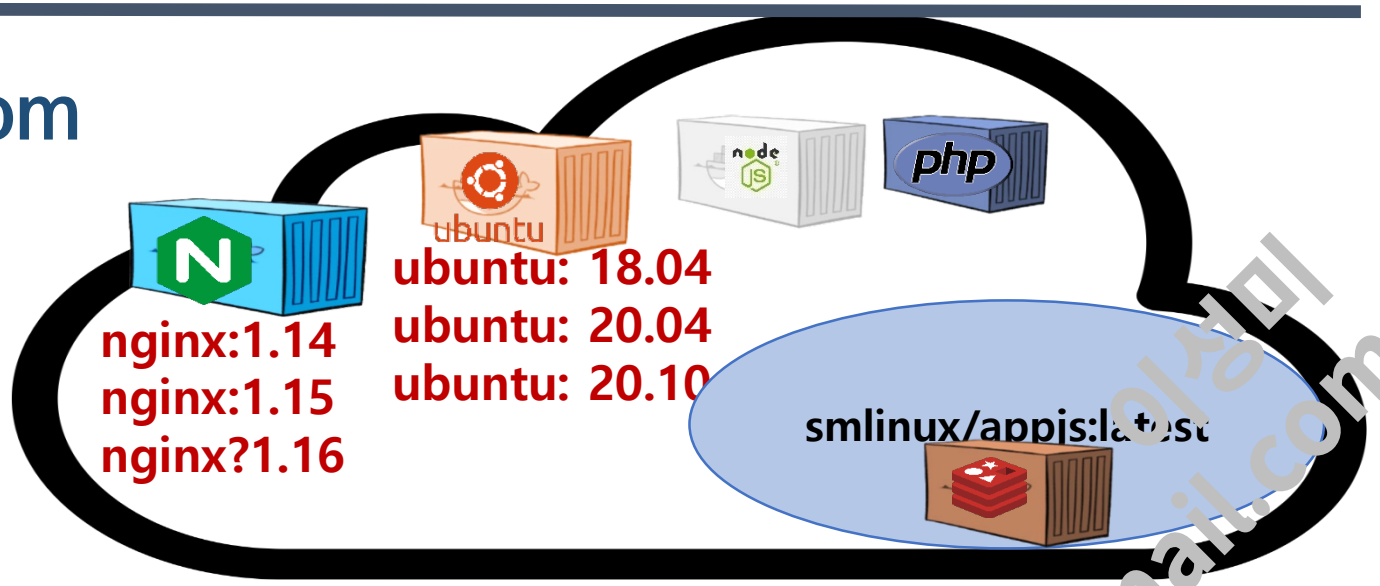
Registry, Repository, Index

- Docker 레지스트리 구성요소
  - Registry: container image를 저장하는 서비스
  - Repository: container image들의 컬렉션
  - Index: 이미지 검색과 태그 및 인증을 담당

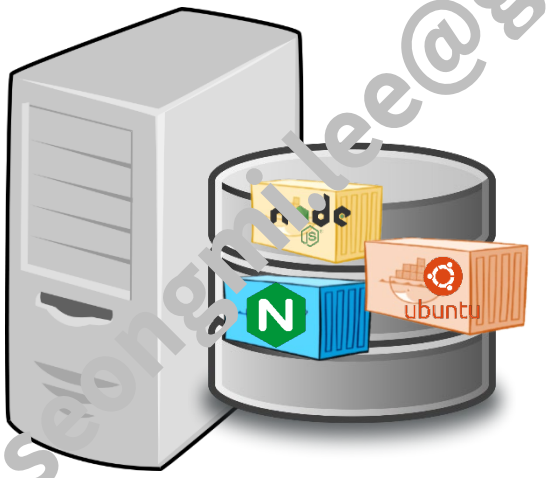


Repository 이름 규칙

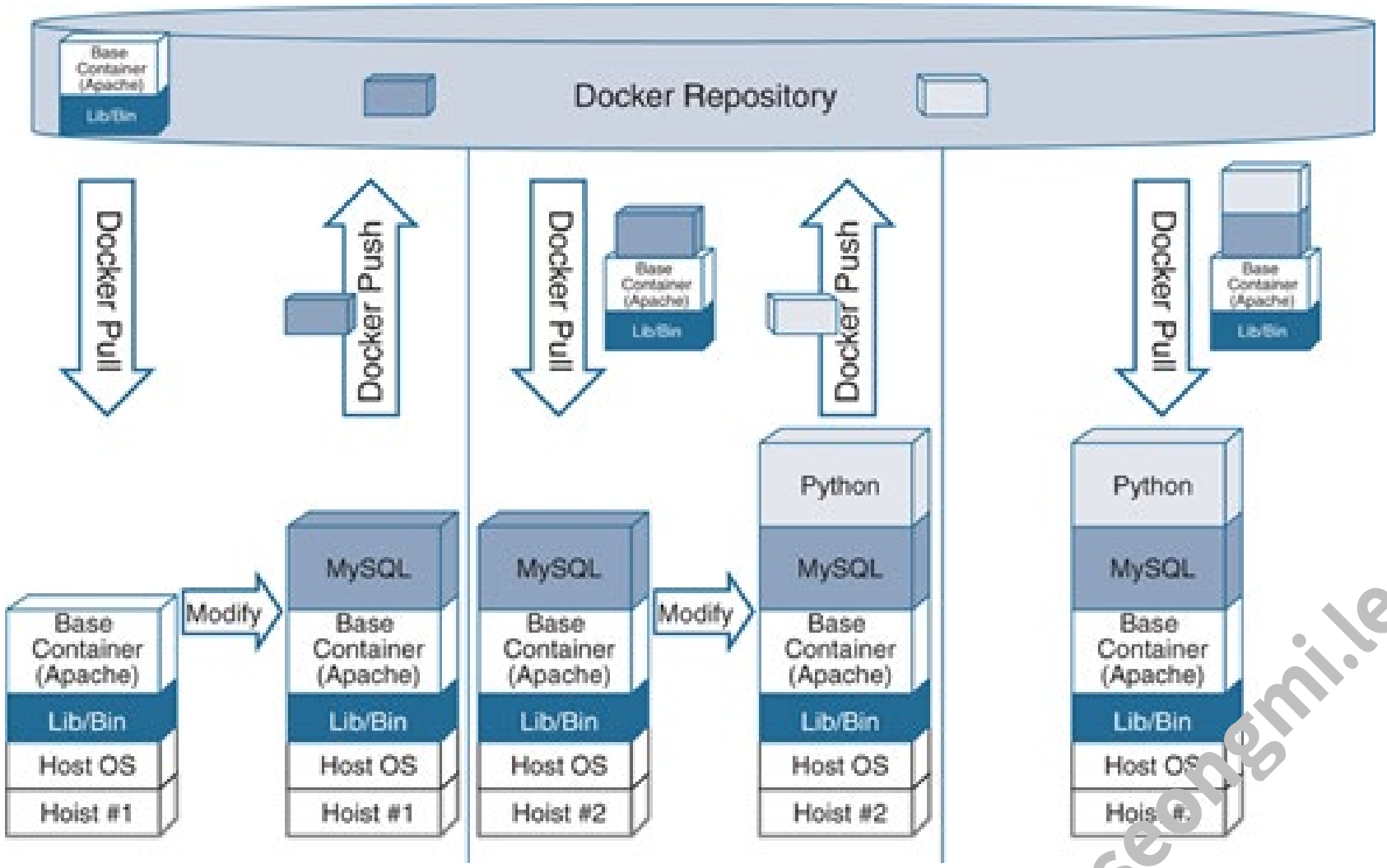
- public registry : hub.docker.com
  - nginx:1.14
  - ubuntu:20.04
  - smlinux/appjs:latest



- private registry : reg.example.com
  - reg.example.com:5000/container_image:tag
  - reg.example.com/webserver:stable.v2



Docker registry와 container image의 관계



이미지 검색

- 컨테이너 이미지 검색 : docker search
  - --filter stars=10 : 별 점을 10개 이상 받은 이미지 검색
  - --no-trunc=true : 이미지 설명을 자르지 말고 출력
  - --automated=true : 자동으로 재구성되는 이미지만 검색

```
$ docker search --filter stars=3 busybox
$ docker search --no-trunc=true mysql
$ docker search --automated=true busybox
$ docker  search --filter  "is-official=true"  --filter "stars=3"  busybox
$ ./dockertags ubuntu
```

Docker Registry

- Docker 이미지를 저장하고 배포하기 위한 오픈 소스 애플리케이션
- Private Registry

registry ☆

Docker Official Images

The Docker Registry 2.0 implementation for storing and distributing Docker images

- 이미지 repository를 기업 내부에서 직접 제공 가능

```
$ docker run -d -p 5000:5000 --restart always --name registry registry:2
$ docker run  - d -p 80:80 --restart=always —name registry ₩
  -e REGISTRY_HTTP_ADDR=0.0.0.0:80  -v /images:/var/lib/registry   registry:2
```

레지스트리에서 이미지 pull & push

- docker pull 명령으로 이미지 다운로드
  - docker run 명령 실행 시 자동 수행
  - Docker Hub 사이트에서 기본 OS 이미지와 미리 구성된 애플리케이션이미지를 제공

    $ docker pull ubuntu:16.04
- docker push 명령으로 이미지 개시
  - hub.docker.com 에 로그인 : docker login

    $ docker login --username=yourUsername --email=yourEmail@address.com
  - 이미지 개시 : docker push

    $ docker push yourUsername/myRepo

실습: Docker Registry 구성

1. Registry 서버 구성
2. Registry 서버 보안 구성
3. 인증서 및 계정기반의 보안 Registry 구성

5. DOCKERFILE

Dockerfile

- 쉽고 간단, 명확한 구문을 갖는 텍스트 파일
- 이미지 생성을 위한 핵심 요소 파일
  - 인스트럭션: 대소문자를 구분하지 않지만 높은 가독성을 위해 대문자 사용
  - 구성 순서가 중요

DOCKERFILE 기본 구성

- 주석: #
- MAINTAINER: 구성 위치 상관없으며 레이어를 생성하지 않음
- FROM : BASE IMAGE
- RUN : COMMAND RUN

ADD와 COPY

ADD	COPY
<원본 경로> <대상 경로> ● 원본 경로: 빌드 작업 디렉터리를 기준으로 상대 경로 ● 원본 경로만 와일드카드 사용 가능 ● 여러 원본 경로 지정은 컴마(,)를 사용 ● 대상 경로: 절대 경로와 상대 경로 사용 ● 대상 경로: 상대 경로 사용 시 WORKDIR 인스트럭션 필요 ● / 는 폴더와 파일을 구분하는 핵심 요소 ● 파일 복사 시 원본 파일의 권한 유지 ● 이미지 생성 시 복사되는 파일을 캐시	
로컬, 네트워크 (URL)에서 파일 복사	로컬 파일 복사만 지원
압축 파일은 대상 경로에서 압축 해제	압축 파일은 압축 해제 없이 복사

CMD와 ENTRYPOINT

- CMD
  - docker run 명령 실행 시 적용할 command를 나열
  - cmd는 하나만 적용가능하고, 여러 개 입력 시 마지막 것으로 적용
  - 컨테이너 동작 시 다른 명령으로 전환 가능
- ENTRYPOINT
  - CMD와 동일한 목적으로 사용되나 docker run 명령에서 전환불가
  - CMD와 함께 사용 시 ENTRYPOINT는 기본 명령어, CMD는 인수를 적용

LABEL

- 버전, 커맨드 등의 정보를 이미지에 심을 때 사용

  LABEL key=value

- value에 공백 포함 시 “”를 사용한다.

```
LABEL description="This is my ₩ multiline description of the software."
LABEL key1="value1" key2="value2" key3="value3"
LABEL key1="value1" ₩
    key2="value2" ₩
    key3="value3“
```

ENV와 ARG

- ENV : 컨테이너 빌드 및 실행 시 적용할 환경변수 정의
- ARG : 컨테이너 빌드 시 사용할 아규먼트 정의

EXPOSE와 VOLUME

- EXPOSE : 호스트와 연결할 포트 번호를 설정
- VOLUME : 호스트 디렉토리를 컨테이너 디렉토리로 마운트

USER, WORKDIR, ONBUILD

- USER : 명령을 실행할 사용자 계정을 설정
- WORKDIR : RUN, CMD, ENTRYPOINT의 명령이 실행될 디렉터리 설정
- ONBUILD : 생성한 이미지를 기반으로 다른 이미지가 생성될 때 명령 실행

STOPSIGNAL, HEALTHCHECK, SHELL

- STOPSIGNAL : docker stop 실행시 작용할 signal을 지정
- HEALTHCHECK : 동작되는 컨테이너의 애플리케이션의 상태를 점검
- SHELL : 컨테이너에 적용할 기본 Shell을 변경

Dockerfile 사용 권장사항

- .dockerignore 파일 사용
  - 이미지 생성 작업 중에 불필요한 파일을 로드 하지 않는다.
- 이미지는 가능한 적은 크기로 구성
- 이미지 레이어 수를 최소화
- 한 컨테이너 당 한 애플리케이션을 구성

DOCKERFILE로 컨테이너 이미지 생성하기

- Dockerfile을 사용한 이미지 생성: docker build
  - -t: 생성될 이미지에 적용할 이름. 여러 번 사용 가능.
  - -f: dockerfile의 파일 이름과 경로. 생략 시 현재 폴더의 dockerfile을 사용

```
docker build -t name:tag  .
```

  - dockerfile이 있는 폴더에서 명령 실행 시 PATH 매개변수로 .을 사용
  - 이미지 생성을 빠르게 하려면 .dockerignore 파일 사용을 권장.

Dockerfile 빌드 시 주의할 점

- 저장 공간 낭비 주의

```
FROM ubuntu:18.04
RUN mkdir /temp-data
RUN dd if=/dev/zero of=/temp-data/filebig count=300 bs=1M
RUN rm -rf /temp-data
CMD ["/bin/bash"]
```

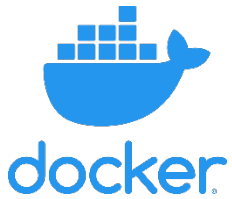
- 볼륨 마운트 주의

```
FROM ubuntu:14.04
VOLUME /MountPointDemo
RUN date > /MountPointDemo/date.txt
RUN cat /MountPointDemo/date.txt
```

Simple example

- webserver 컨테이너 생성
- nodejs 애플리케이션 컨테이너 생성

실습: Dockerfile



1. dockerfile을 사용하여 이미지 생성하기
2. Context와 .dockerignore
3. CMD Directives
4. ENV, ENTRYPOINT, 및 CMD Directives
5. RUN Directive
6. Docker Volume

6. CONTAINER STORAGE

컨테이너 스토리지

- 컨테이너의 임시적인 성격
- 비즈니스 연속성에 대한 필요
  - 컨테이너가 삭제되면 응용프로그램 데이터도 함께 사라짐
  - Docker 컨테이너 이미지는 재사용 가능한 응용 프로그램을 포함
  - 컨테이너 외부에 저장소에 연결하여 데이터를 저장해야 함
- 컨테이너에 대한 스토리지와 볼륨 관리
- 호스트 상의 Docker 스토리지 관리

도커의 스토리지 드라이버

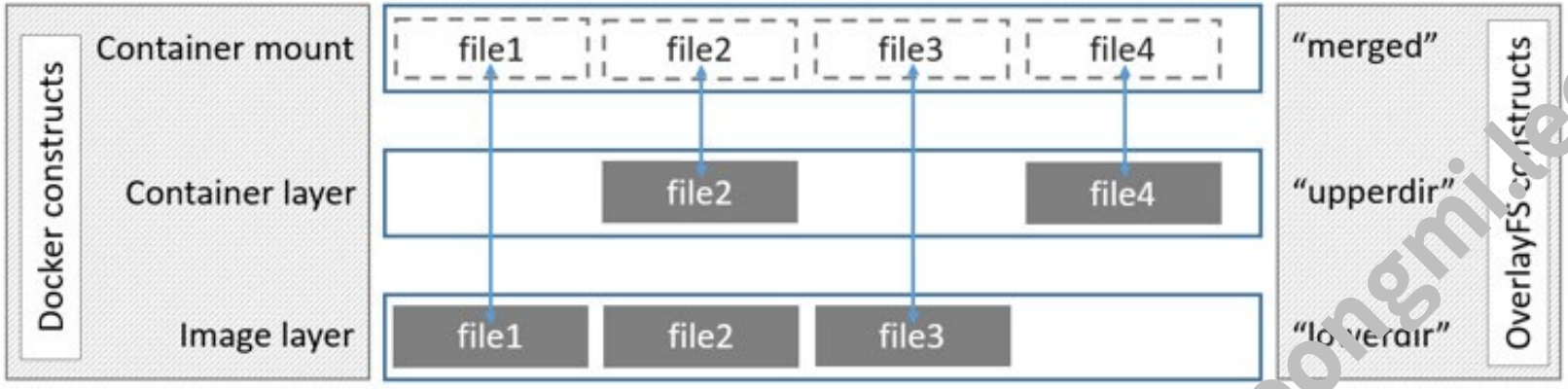
- 스토리지 드라이버
  - OS별 지원되는 스토리지 드라이버
  - overlay2, aufs, devicemapper

Linux	추천 드라이버	지원되는 드라이버
Ubuntu	overlay2, aufs	overlay, devicemapper, zfs, vfs
Debian	overlay2, aufs, devicemapper	overlay, zfs
CentOS	overlay2	overlay, devicemapper, zfs, vfs
Fedora	overlay2	overlay, devicemapper, zfs, vfs

- 스토리지 드라이버 별 backingfilesytem
  - ext4, xfs, btrfs 등

aufs와 overlay

- aufs
  - 최초의 스토리지 드라이버로 커널이 포함되어 있지 않은 파일 시스템
- overlay
  - aufs에 비해 기본적으로 단순한 구조
  - aufs에 비하여 안정성은 조금 낮은 편이지만, 속도가 빠르고 메모리 사용에 효율적



https://docs.docker.com/storage/storagedriver/overlayfs-driver/

호스트의 볼륨 공유

- 호스트의 디렉터리와 컨테이너 간 데이터 공유
  - volume mount
- docker run에 -v 옵션 사용

  -v 〈컨테이너 마운드 경로〉

  -v 〈호스트 경로〉:〈컨테이너 마운드 경로〉

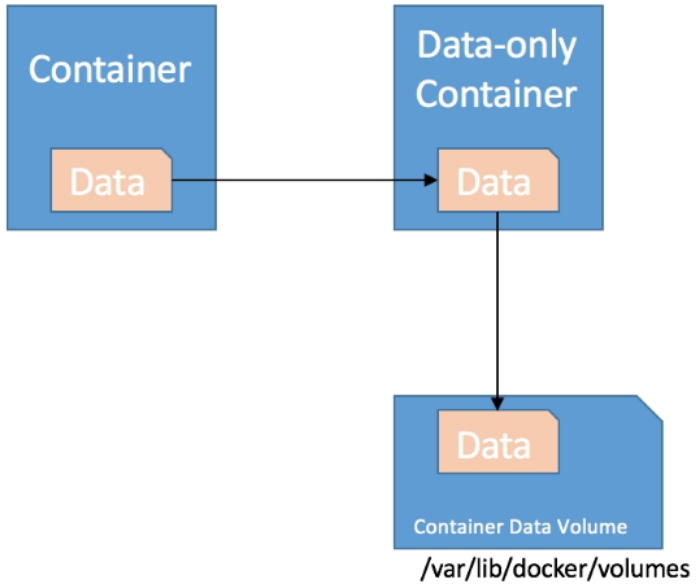
  -v 〈호스트 경로〉:〈컨테이너 마운드 경로〉:〈읽기 쓰기 모드〉

```
$ docker run -it -v /webdata:/usr/share/nginx/html -d -p 80:80 ubuntu:latest
$ docker run -it -v /webdata:/usr/share/nginx/html:ro -d -p 80:80 ubuntu:latest
$ docker run -it -v webdata:/usr/share/nginx/html -d -p 80:80 ubuntu:latest
$ docker run -it -v /usr/share/nginx/html -d -p 80:80 ubuntu:latest
```

데이터 전용 컨테이너

- 컨테이너 중 하나에 이미 볼륨이 마운트 되어 있는 경우 다른 컨테이너를 시작할 때 Docker가 같은 볼륨을 사용하도록 지시

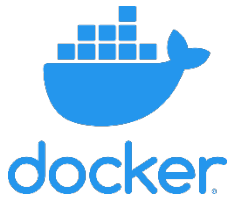
```
$ docker run -it --name dataonly -v /data ubuntu
$ docker run -it --name c1 --volumes-from dataonly busybox
```



docker volume을 사용하여 명시적으로 볼륨 생성

- docker volume create
- docker volume ls
- docker volume rm
- docker volume prune
- docker volume inspect

실습: Docker Volume

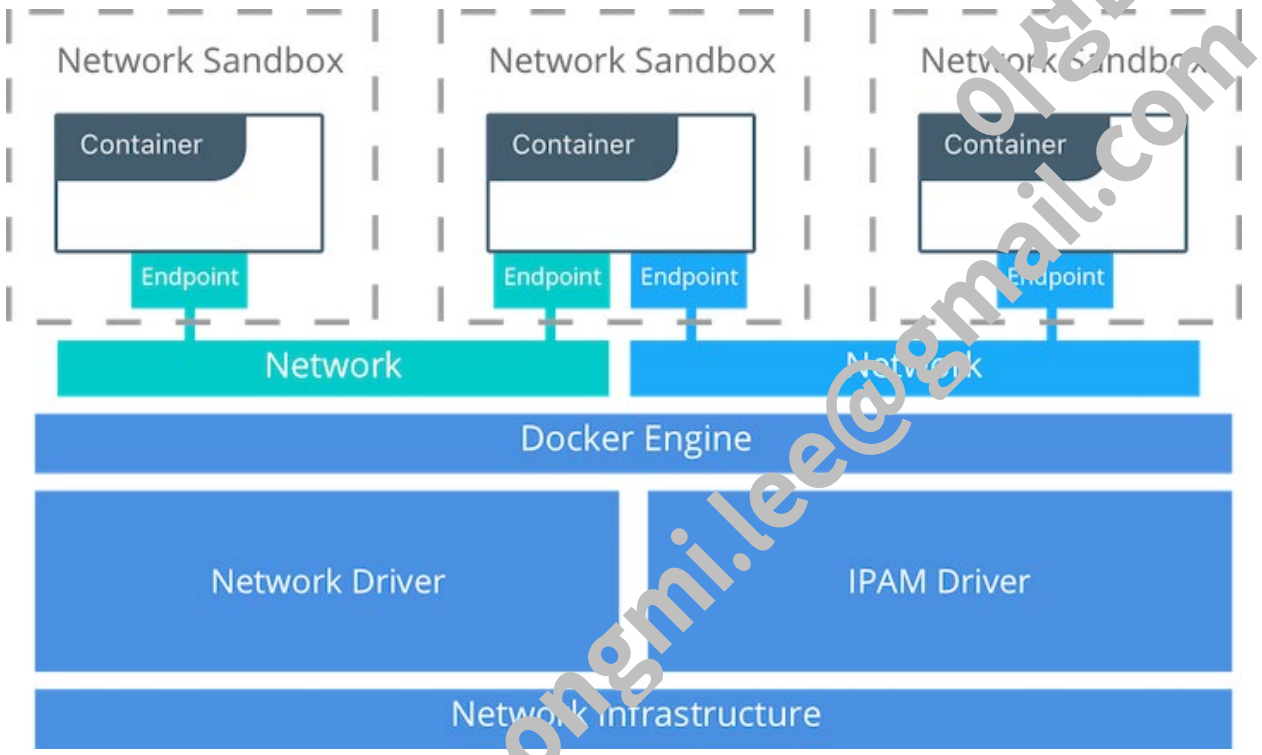


1. Docker Volume 기본
2. 개별 파일들을 볼륨으로 마운트하기
3. 호스트 디렉터리를 볼륨으로 마운트하기
   (SELinux보안 포함)

7. CONTAINER NETWORK

Container Networking Model

- Docker는 컨테이너 네트워크 모델 (CNM) 개념을 사용
  - sandbox: 컨테이너의 격리 환경의 네트워크 구성
  - endpoint: 네트워크 인터페이스
  - network: 네트워크 인터페이스 그룹



https://success.docker.com/article/networking

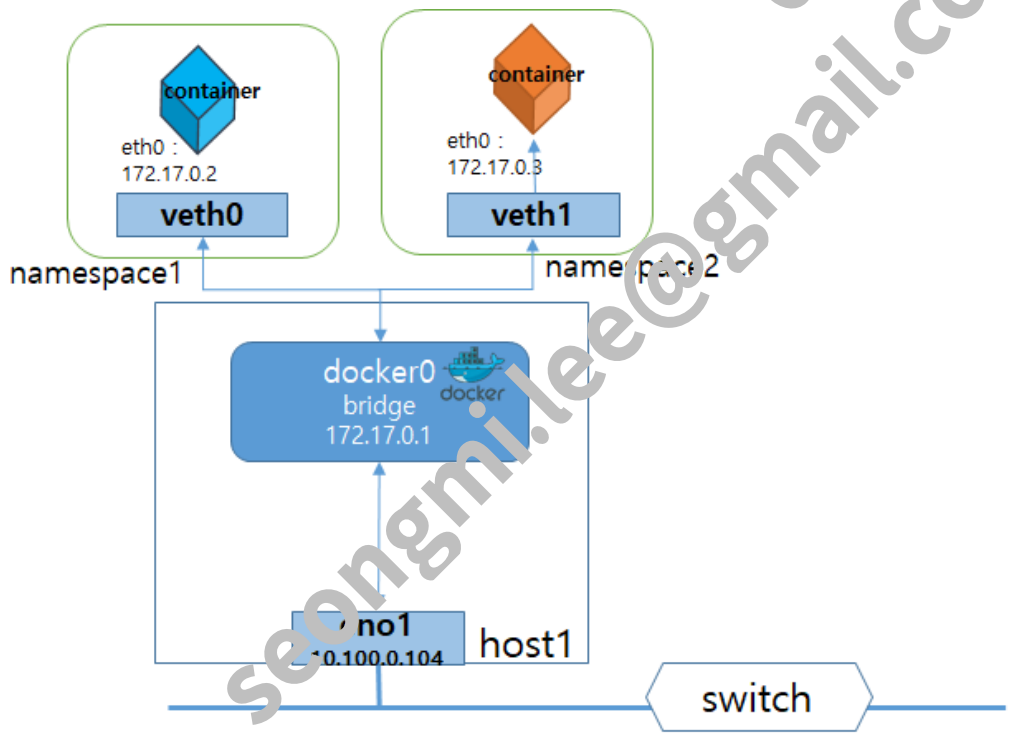
Network Driver

- Docker 설치 시 자동으로 생성되는 네트워크 드라이버
  - bridge: 리눅스의 가상 브릿지
  - host: 컨테이너가 호스트 네트워크를 사용
  - null: 네트워크 구성을 갖지 않음
  - macvlan
  - ipvlan
  - overlay

```
$ docker info
$ docker network ls
```

Bridged network

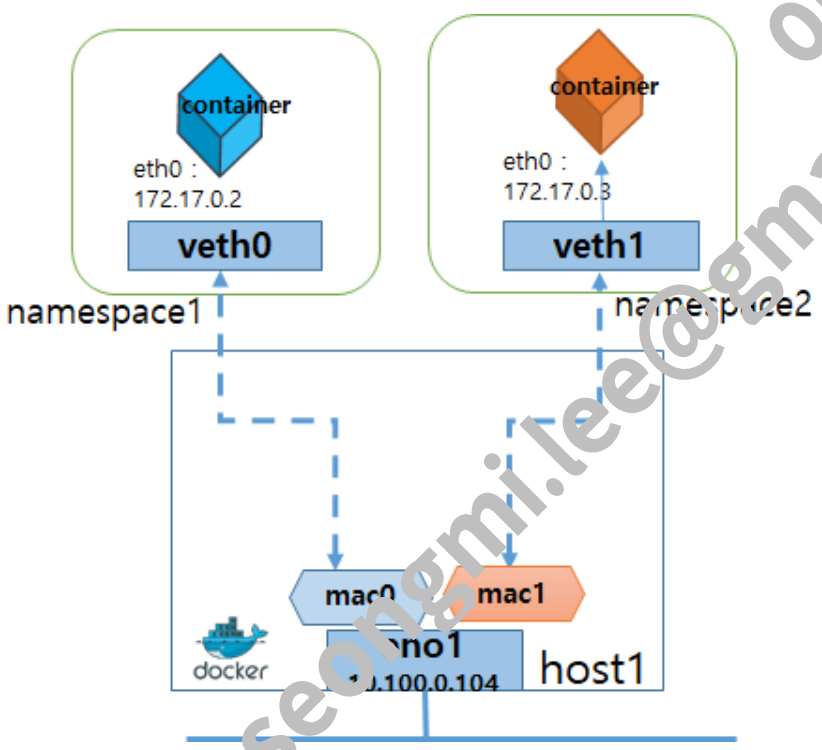
- 브릿지 네트워크는 모든 Docker 호스트에 존재
- docker0
  - Docker 호스트의 default network
  - 컨테이너를 NAT를 사용하여
    호스트 외부 네트워크로 포워딩
  - 172.17.0.0/16을 사용하여
    DHCP로 컨테이너에 IP address 제공
  - 172.17.0.1이 docker0에 할당
  - 컨테이너에 할당된 IP 확인: docker inspect



Macvlan

- parent-interface와 sub-interface
  - parent-interface(eno1)은 컨테이너의 mac0@eth0를 저장 : macvlan bridge
  - 호스트와는 통신이 안되지만, 다른 서브 인터페이스간 통신은 가능

```
$ docker network create -d macvlan ₩
    --subnet=10.0.10.0/24 ₩
    --gateway=10.0.10.1 ₩
    -o parent=eno1 ₩
    my_macvlan
```



User Defined Networking

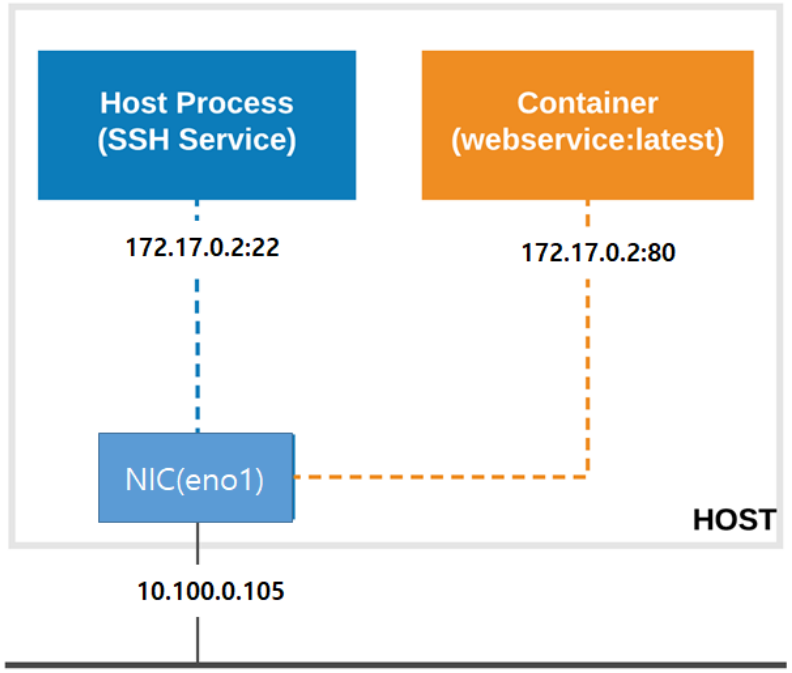
- 네트워크를 사용자가 직접 정의하여 생성 : docker network create
- Docker 네트워크 생성에 사용하는 옵션들

```
# docker network create --driver bridge ₩
--subnet=10.15.20.0/24 ₩
--gateway=10.15.20.1 ₩
--aux-address 1=10.15.20.2 --aux-address 2=10.15.20.3 ₩
--opt com.docker.network.bridge.host_binding_ipv4=10.10.10.101 ₩
--opt com.docker.network.bridge.name=linuxbridge1 ₩
testbridge1
```

컨테이너 간 네트워크 공유

- 다른 컨테이너와 동일 네트워크에 연결 : --net=container:〈컨테이너〉

```
$ docker run -d --name=sshd  --net=bridge  smlinux/sshd
$ docker run -d --name web --net=container:sshd smlinux/webserice
```



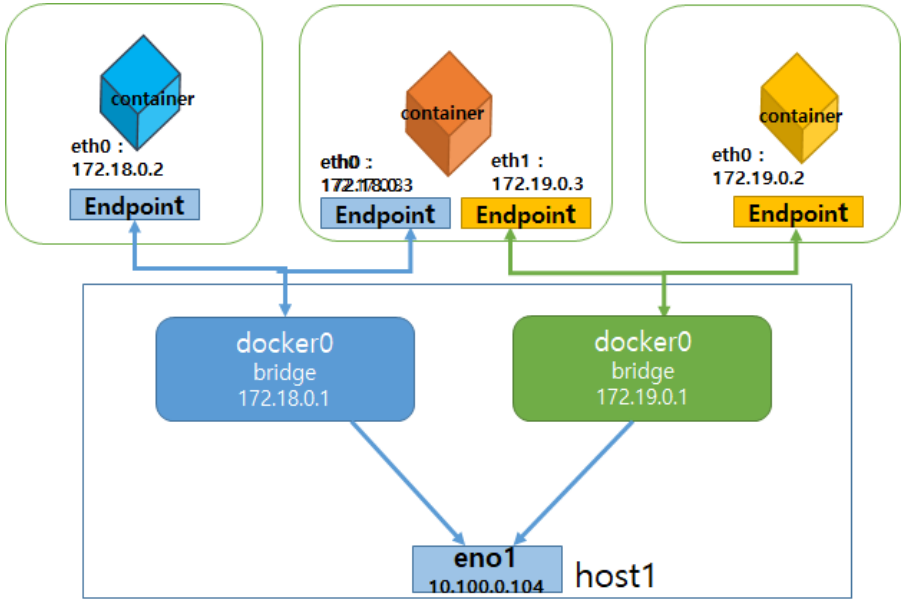
네트워크 연결

- 컨테이너를 통해 네트워크 연결 : docker network connect

  docker network connect ⟨network ID or name⟩ ⟨container Id or name⟩

- 네트워크 연결해제: docker network disconnect

  docker network disconnect ⟨network ID or name⟩ ⟨container Id or name⟩



멀티 호스트 네트워크 생성

- 멀티 Docker 호스트 간 네트워크 연결: overlay network
- 오버레이 네트워크 사용 요구사항
  - Swarm 모드 구성
  - Key-value 스토어 구성

네트워크 플러그인

- 도커 네트워크 플러그인은 대부분 멀티호스트 환경의 PaaS를 위한 VxLAN
- 멀티호스트 플러그인
  - 종류
    - Flannel
    - Weave : weaveworks에서 개발
    - Project Calico 등

포트 매핑과 Expose

- Dockerfile에서 EXPOSE를 사용
- docker run 명령 수행 시 옵션을 사용
  - -p hostPort:containerPort
  - -p containerPort
  - -P
- 열려있는 포트 정보 확인
  - docker ps
  - docker inspect
  - docker port

컨테이너 링크

- 컨테이너와 컨테이너의 네트워크 연결허용 :

  --link containerName:aliasName

- multi-tier 컨테이너 운영
- 호스트이름이 대상 컨테이너의 /etc/hosts 파일에 기록
- 환경 변수가 대상 컨테이너에 전송

실습:Docker Networking

1. Default Networks
2. 특정 네트워크 범위를 설정한 사용자 정의 네트워크
3. Network alias
4. VxLAN weave 네트워크 구성
5. Docker Port와 Link
6. 포트 매핑
7. Link를 사용하여 컨테이너 연결하기

8. Security And Resource Limits

컨테이너 이미지 보안(1)

- 공개된 도커 이미지의 안전성
  - 도커 허브는 계정만 있으면 누구나 repository를 만들어 컨테이너 배포가능
  - quay.io : coreOS에서 운영하는 repository
  - 악의적인 이미지가 포함되어 배포될 수 있음
  - 믿을 수 있는 이미지를 선별하여 사용 : Official Image, Verified Publisher

```
$ docker search  --filter is-official=true  nginx
```

- Docker Bench for Security
  - 도커 호스트와 이미지의 보안 취약점을 발견하는 도움을 주는 컨테이너
  - https://github.com/docker/docker-bench-security

컨테이너 이미지 보안(2)

- 컨테이너를 침입으로 부터 보호
  - root로 로그인 금지
  - 방화벽 설정을 통한 docker host 접근제한
  - VPN 적용 및 내부망 접근 제한
- 컨테이너 애플리케이션 실행용 사용자 추가
- docker socket 파일 공유 제한
  - /var/run/docker.sock 공유제한

호스트 OS 주요 자원 접근 제어

- 주요 시스템 디렉터리 마운트 금지
  - 각 컨테이너에 매핑된 디렉터리의 목록과 권한 확인

    $ docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}'
  - 중요 시스템 디렉토리 목록 : /boot, /dev, /etc, /lib, /proc, /sys, /usr
- 호스트 장치 파일 컨테이너에 직접 노출 금지
  - 컨테이너 내 불필요하게 접근 가능한 호스트 장치의 존재 여부확인

    $ docker inspect --format '{{ .Id }}:Devices={{ .HostConfig.Devices }}'
  - 호스트 장치 파일 컨테이너에 직접 노출 금지

    $ docker run -it --device=/dev/temp_sda:/dev/temp_sda:r centos

참고 : 클라우드 보안가이드(컨테이너 보안) - SKinfosec) 2019

호스트 OS 주요 자원 접근 제어

- 각 컨테이너에 매핑된 디렉터리의 목록과 권한 확인
  - docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}'
- 호스트 장치 파일 컨테이너에 직접 노출 금지
- 주요 시스템 디렉터리 마운트 금지
- 호스트 장치 파일 컨테이너에 직접 노출 금지(Container Runtime)

참고 : 클라우드 보안가이드(컨테이너 보안) - SKinfosec) 2019

인증-권한 제어

- Docker는 기본적으로 사용자 인증을 지원하지 않음
  - Docker Daemon에 접속할 수만 있으면 Docker 클라이언트 모든 명령을 실행 가능
  - docker group 내 신뢰할 수 있는 사용자만 존재하는지 확인

  $ getent group docker
- 필요 시 authorization plugin을 사용한 권한 제어하거나 오케스트레이션을 이용해 외부와 내부 인증을 강화

참고 : 클라우드 보안가이드(컨테이너 보안) - SKinfosec) 2019

네임스페이스 관리

- namespace isolate는 완전한 isolate가 아닌 host내의 논리적 isolate
  - 호스트의 network namespace 공유 금지
  - 호스트의 process namespace 공유 금지
  - 호스트의 IPC namespace 공유 금지
- 컨테이너가 불필요하게 과도한 권한을 가지지 않도록 설정

참고 : 클라우드 보안가이드(컨테이너 보안) - SKinfosec) 2019

컨테이너 네트워크 제어

- Bridge 방식의 default 네트워크 제한
- 불필요한 포트 매핑 금지
- 호스트 네트워크 인터페이스 설정
- 컨테이너 내 ssh 실행 금지
- userland 프록시 사용제한

참고 : 클라우드 보안가이드(컨테이너 보안) - SKinfosec) 2019

Limit Memory

- **memory options**

-m (--memory)	: The maximum amount of memory the container can use.
--memory-reservation	: Guarantee on memory
--kernel memory	: The maximum amount of kernel memory the container can use.
--memory-swap	: The amount of memory this container is allowed to swap to disk.
--memory-swappiness	: Defines how much (and how often) your Linux kernel will copy RAM contents to swap.
--oom-kill-disable	: Disable out-of-memory killer.
--oom-score-adj int	: Tune host's OOM preferences (-1000 to 1000)

Limit CPU

- cpu options

--cpu-shares : 컨테이너가 사용하는 CPU 비중을 1024 값을 기반으로 설정

--cpuset-cpu : 컨테이너가 사용할 수 있는 CPU나 코어를 제한

--cpu-period : CPU CFS 스케줄러 기간을 지정

--cpu-quota : 컨테이너에 CPU CFS 할당량을 부여.

--cpus : 컨테이너가 사용할 수 있는 CPU 리소스

```
$ docker run --cpu-shares 2048 --name c1 -d smlinux/stress:latest
$ docker run -it --cpuset-cpus 1  smlinux/stress:latest stress --cpu 1
$ docker run -it —cpus=".5"  smlinux/stress:latest stress --cpu 1
$ docker run --cpu-period=100000 --cpu-quota=100000 --name c1 -d smlinux/stress stress --cpu 1
$ docker run --cpu-period=100000 --cpu-quota=25000 --name c2 -d smlinux/stress stress --cpu 1
$ docker stats c1 c2
```

Limit Block IO

- block_IO options

--blkio-weight

--blkio-weight-device

--device-read-bps

--device-write-bps

--device-read-iops

--device-write-iops

```
$ docker run -it --rm --blkio-weight 100  ubuntu /bin/bash
$ docker run -it --rm --device-write-bps /dev/vda:1mb ubuntu /bin/bash
$ docker run -it --rm --device-write-bps /dev/vda:10mb ubuntu /bin/bash
$ docker run -it --rm --device-write-iops /dev/vda:10 ubuntu:latest /bin/bash
$ docker run -it --rm --device-write-iops /dev/vda:100 ubuntu:latest /bin/bash
```

Limit Restarts

- restart={no|always|on-failure|unless-stoped}

  no :기본 값으로 컨테이너는 명시적으로 실행할 때만 시작

  always :종료 코드로 종료했는지 상관없이 Docker 서비스가 시작하면 항상 컨테이너를 재시작

  on-failure : 0이 아닌 코드로 컨테이너가 종료할 경우에만 컨테이너를 다시 시작함.
  컨테이너를 재시작하려는 시도를 횟수로 지정: on-failure:5

  unless-stoped: Docker 데몬 시작 시, 이전에 종료된 상태의 컨테이너는 다시 시작하지 않음

  $ docker run -d --restart=on-failure:10 my-image

  $ docker run --restart=always mongo

Limit Filesystems

- file에 쓰기 방지
  - --read-only flag

    $ docker run --read-only debian touch file
  - volumes by adding :ro

    $ docker run -v $(pwd):/pwd:ro debian touch /pwd/file

Limit Capabilities

- defines sets of privileges
- Docker containers run with a subset of capabilities
- give extended privileges to a container
  - --privileged argument
- --cap-add and --cap-drop arguments
  - --cap-add : 컨테이너에 capabilities를 부여하는 플래그
  - --cap-drop : 컨테이너에 capabilities를 제한하는 플래그

Apply Resource Limits (ulimits)

- resource limits that can be applied to processes
- cpu : soft,hard limist(초단위)
  - soft(SIGTERM):hard(SIGKILL)

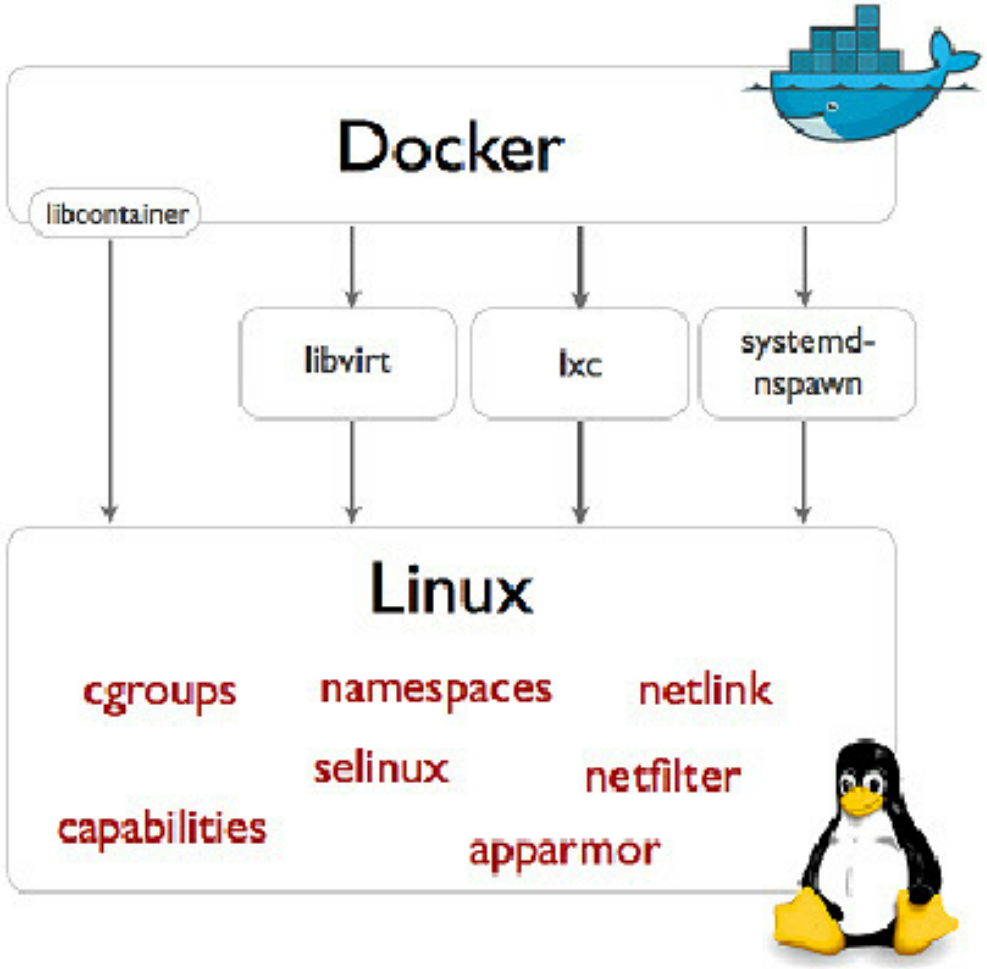
  $ docker run --ulimit cpu=12:14 debian
- nofile : 오픈할 수 있는 파일디스크립터 수

  $ docker run --ulimit nofile=25 debian
- nproc

  $ docker run --ulimit nproc=10 -it debian

Linux Security Modules

- SELinux



iptables: A More Secure Approach

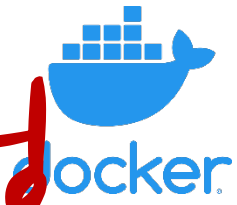
- Default policy
  - allow all traffic, then block untrusted traffic"That which is not explicitly forbidden is allowed" easier to administer, but far less safe
  - deny all traffic, then allow trusted traffic"That which is not explicitly allowed is prohibited" more work initially, but safer
- Block inbound while allowing locally intiated connections

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -P INPUT DROP
```

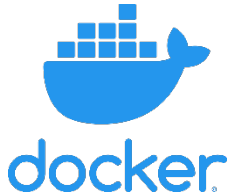
Docker Monitoring

- Using the tools built into Docker
  - docker stats
  - docker top
  - docker exec
  - docker logs
  - docker events
- Using standard operating system commands
- Advanced Container Resource Analysis
  - cAdvisor

실습: 컨테이너 보안 및 리소스 제한 설정



1. 컨테이너를 리소스 제한 및 모니터링
2. 부팅 시 시작하도록 컨테이너 구성하기
3. capabilities 제한
4. iptables와 컨테이너
5. 외부에서 컨테이너에 접속하기
6. DNAT rule 없이 port forwarding : docker-proxy
7. 컨테이너간의 통신 제한
8. 컨테이너간의 통신 부분허용 : --link
9. 컨테이너 로그 보기



9. DOCKER COMPOSE

Docker Compose

- tool for defining and running multi-container Docker applications
- Install Compose on Linux systems

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

YAML(YAML Ain't Markup Language)

- 사람이 쉽게 읽을 수 있는 데이터 직렬화 양식
- 기본 문법
  - 구조화된 데이터를 표현하기 위한 데이터 포맷
  - Python처럼 들여쓰기로 데이터 계층을 표기
  - 들여쓰기를 할 때에는 Tab이 아닌 Space Bar를 사용
  - 가독성이 좋아 설정 파일에 적합한 형식
  - 배열: '-'를 붙이며 뒤에는 반드시 Space Bar를 입력해야 함
  - 공식 사이트 : http://yaml.org/

Docker Compose 사용하기

docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]

- docker-compose 명령의 주요 옵션들

-f	: docker-compose.yml 파일 경로와 파일 이름
-p	: 프로젝트 이름으로 기본 값은 명령을 수행하는 폴더 이름
--verbose	: 이미지에 대한 구성 내용을 출력
-v	: Docker Compose 클라이언트 버전 출력

Docker Compose를 위한 YAML

- Compose 파일은 YAML 파일로 서비스, 네트워크, 볼륨 등 애플리케이션에 대한 모든 것을 정의
- Compose 파일의 기본경로와 이름
  - 현재 디렉터리의 docker-compose.yml
  - 파일 확장자로 .yml 또는 .yaml 모두 사용가능
- Dockerfile에서 정의한 옵션은 docker-compose.yml 파일에서 다시 정의할 필요 없음.

Docker Compose를 위한 YAML version

- version1 : do not declare a version
- version2
  - must indicate the version number at the root of the document
  - must be declared under the services key
- version3
  - cross-compatible between Compose and the Docker swarm
  - remove : volume_driver, volumes_from, cpu_shares, cpu_quota, cpuset,...
  - add: deploy

- version1 : do not declare a version
- version2
  - must indicate the version number at the root of the document
  - must be declared under the services key
- version3
  - cross-compatible between Compose and the Docker swarm
  - remove : volume_driver, volumes_from, cpu_shares, cpu_quota, cpuset,...
  - add: deploy