Docker Container

JADECROSS 정환열 이사

coordinatorj@jadecross.com

2022

IT Medical Center
JADECROSS

IT Medical Center, JADECROSS

목 차

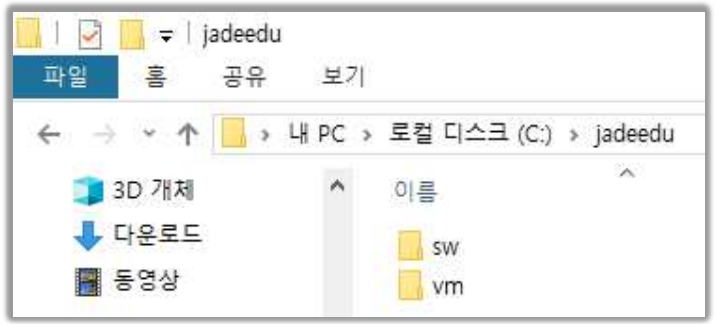
Appendix

실습환경 설정

실습 환경설정 (1/6) - 실습파일 다운로드 및 압축 해제

- 실습파일 다운로드
  - https://www.dropbox.com/s/znp9247yjl11d7w/jadeedu_docker_20210206.zip?dl=0

- jadeedu_docker.zip파일을 C:₩ 하위에 압축 해제

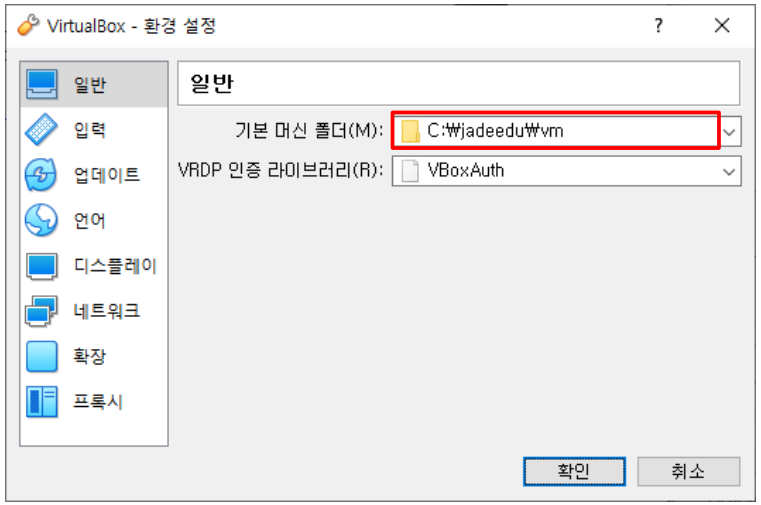


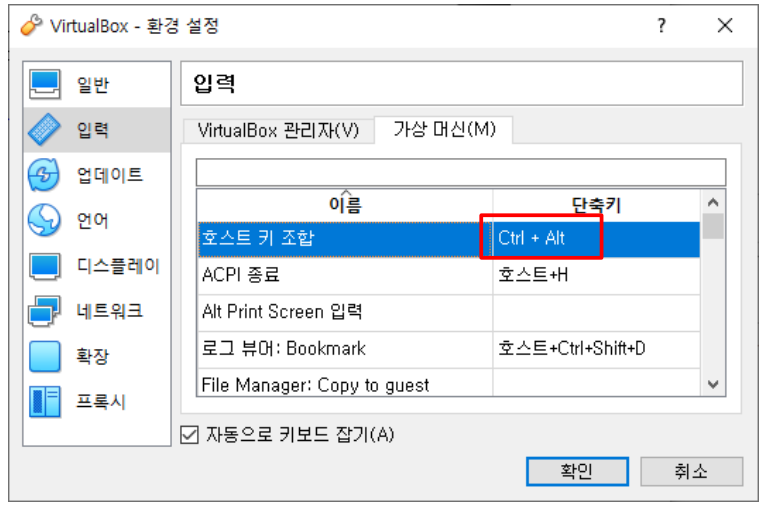
실습 환경설정 (2/6) - VirtualBox 설치

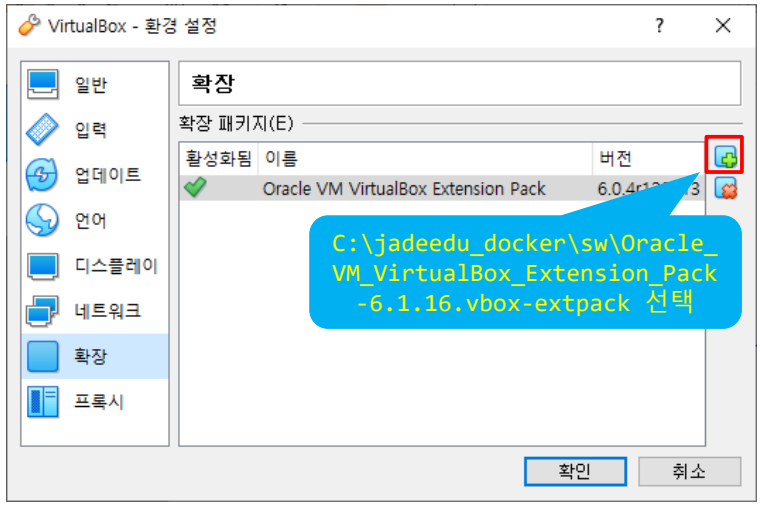
- **VirtualBox 6 설치**
  - `C:\jadeedu_docker\sw\VirtualBox-6.1.16-140961-Win.exe` 더블클릭

- **VirtualBox 환경 설정**
  - 파일 → 환경설정

일반	입력	확장
기본 머신 폴더	가상 머신 └▶ 호스트 키 조합	확장 패키지 추가
`C:\jadeedu\vm`	`Ctrl + Alt`	



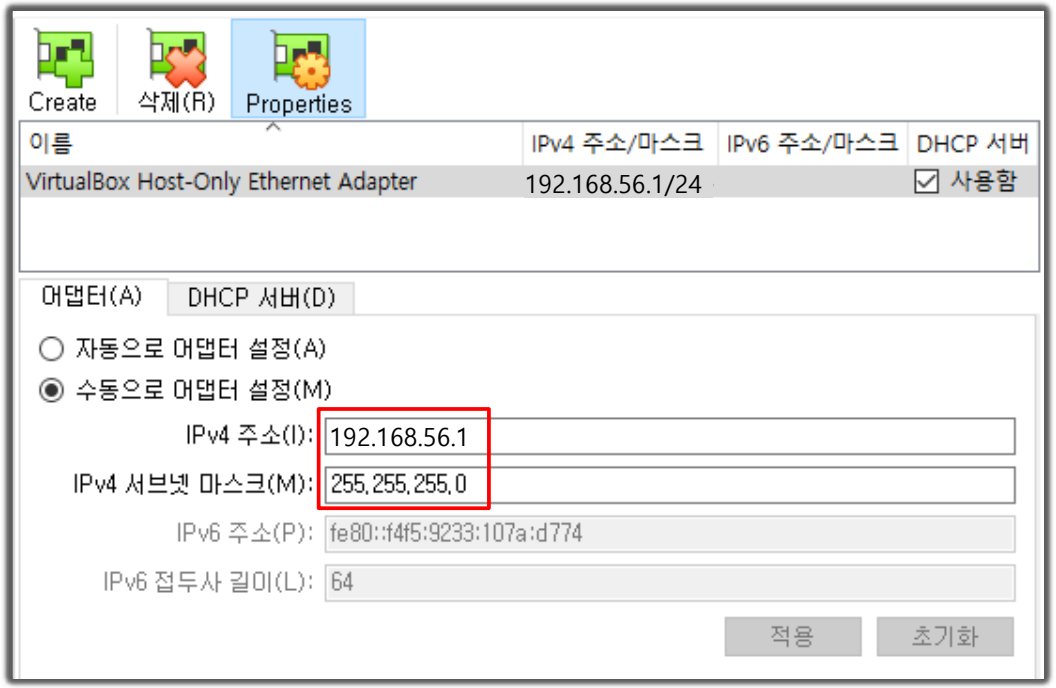


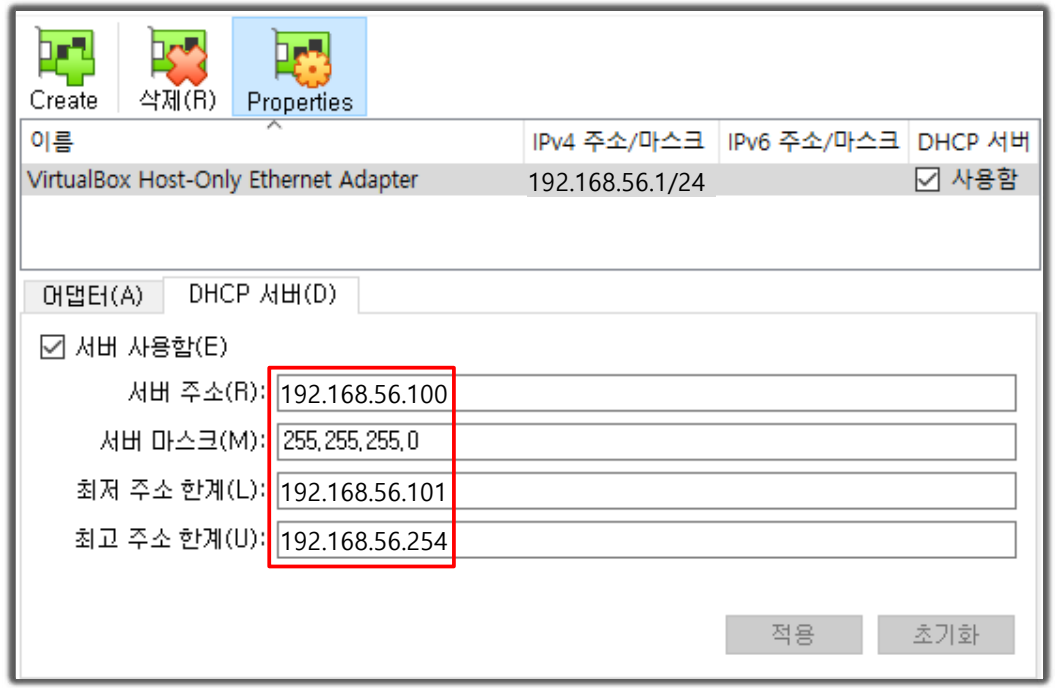


실습 환경설정 (3/6) - VirtualBox 환경설정

- **네트워크 설정**
  - "Tools → 네트워크" 선택

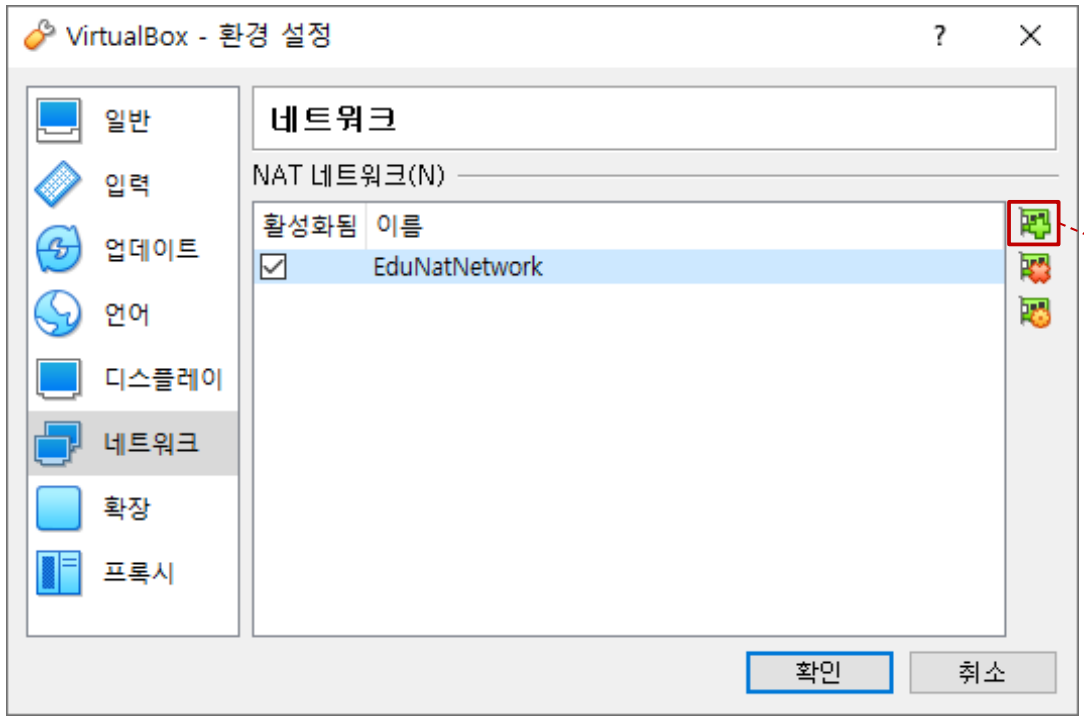
- **IP 대역 설정**
  - 192.168.56.0/24 대역으로 네트워크 설정

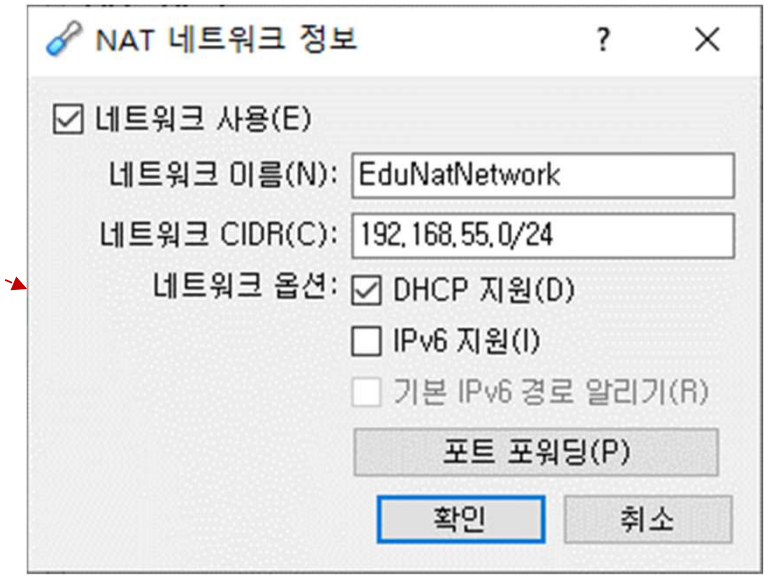




실습 환경설정 (4/6) - VirtualBox 환경설정

- **"NAT 네트워크" 추가**
  - 파일 → 환경 설정 → 네트워크" → 
- **NAT 네트워크 정보**
  - 네트워크 이름 : EduNatNetwork
  - 네트워크 CIDR : 192.168.55.0/24

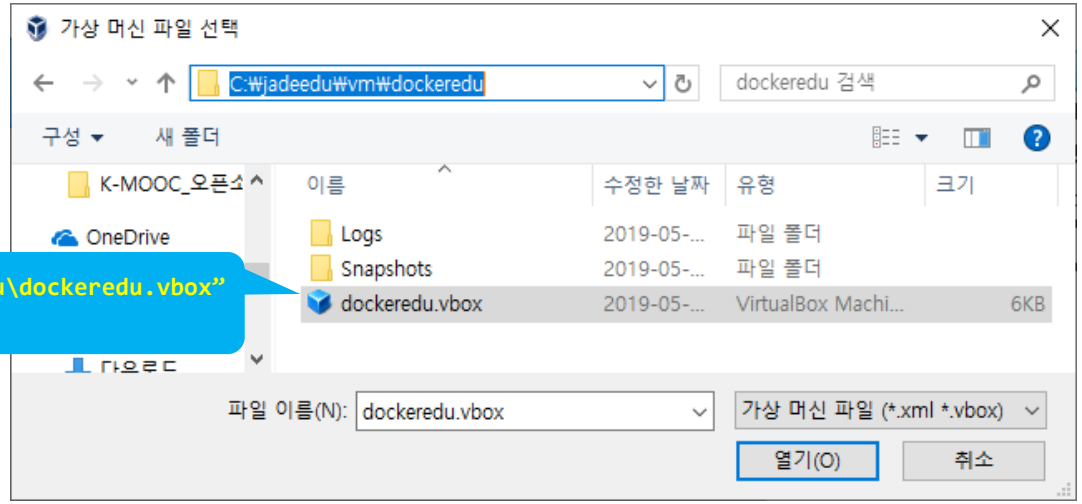




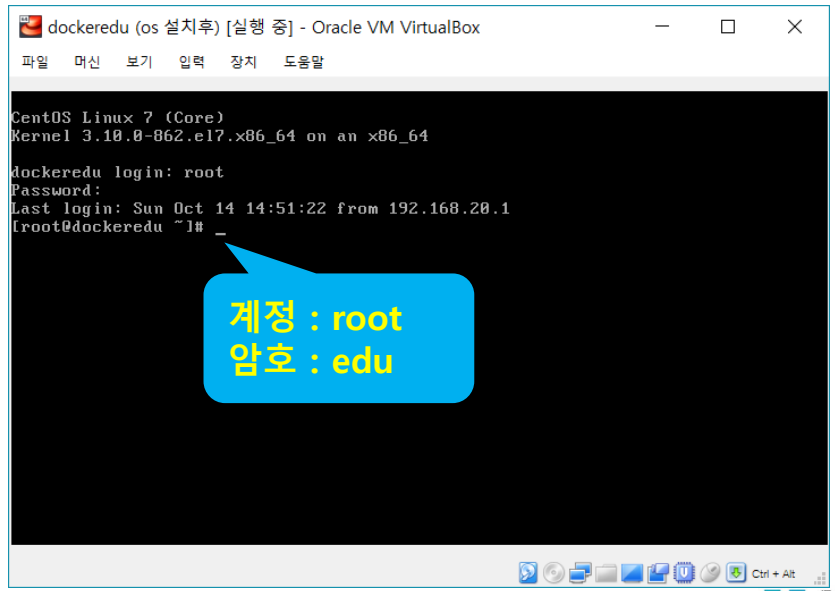
실습 환경설정 (5/6) - 실습 VM 가져오기 및 실행

- 

클릭하여 VM 가져오기

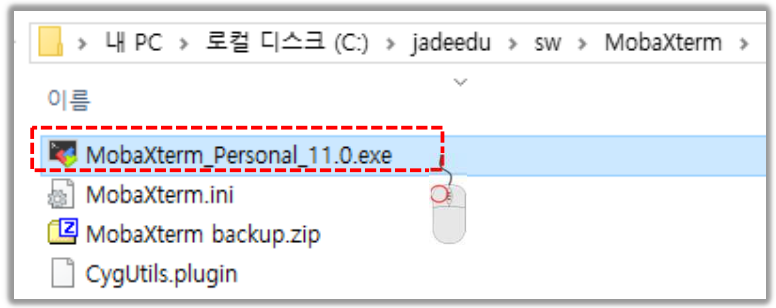


- dockeredu 가상머신 실행

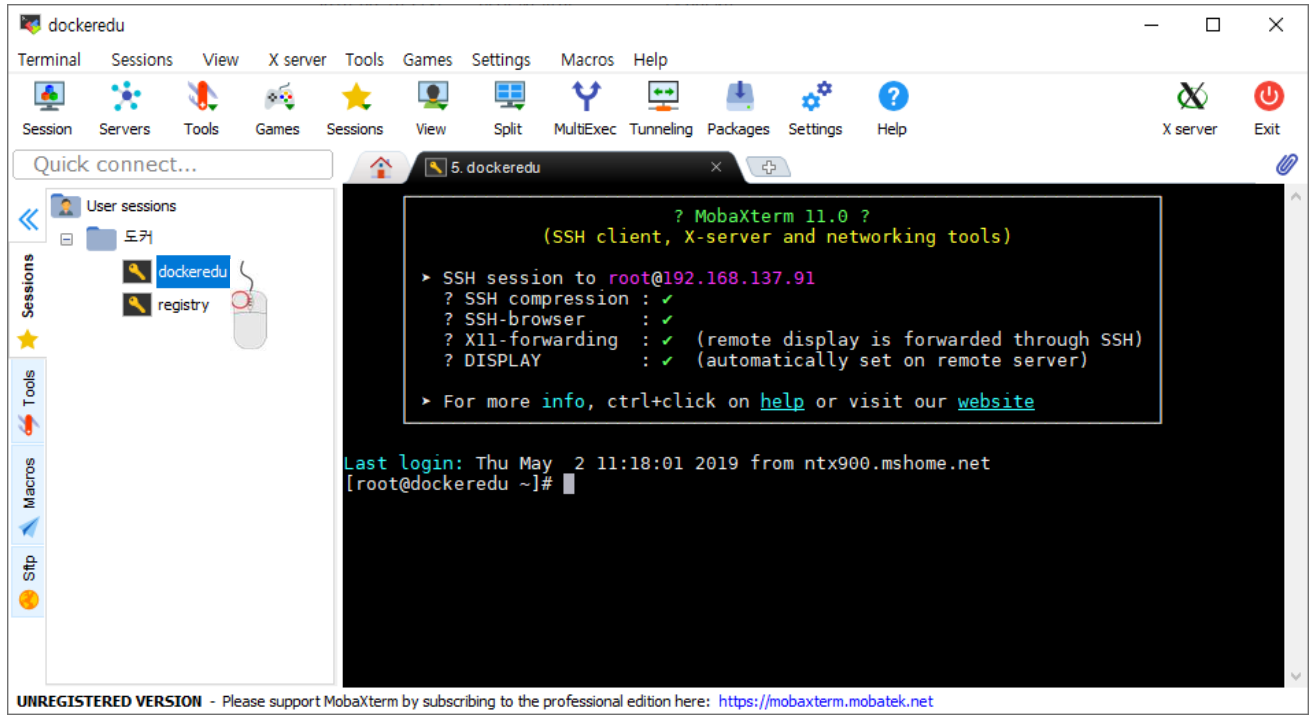


실습 환경설정 (6/6) - 원격 접속

- **MobaXterm 실행**



- **dockeredu VM 원격 접속**

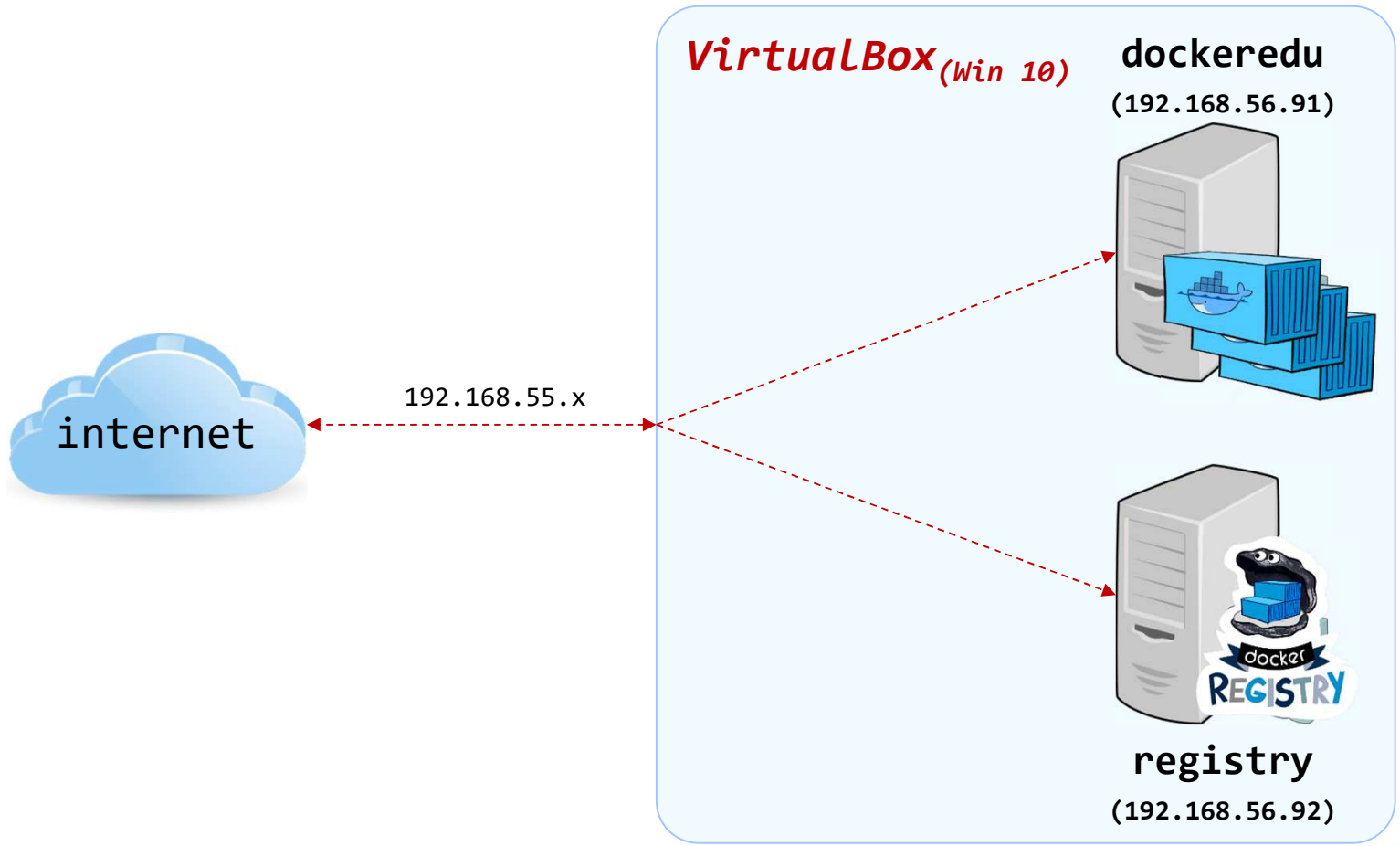


실습 환경

- **실습 환경 최소 조건**
  - Windows 10, 8G RAM, VirtualBox(하이퍼바이저) 실행 가능해야 함
  - SSD 디스크 사용할수록 실습이 용이함

- **최소 VM 메모리 설정**

	dockeredu	registry
vCPU	2	2
Memory (G)	4	1



실습 환경 - Play with Kubernetes

- https://labs.play-with-k8s.com
  - 도커에서 제공하는 사이트로 4시간 사용가능, github 또는 docker 계정으로 로그인

- **노드 생성 및 실습환경 구성**
  - "ADD NEW INSTANCE" 클릭

① 도커 20.10 버전이 이미 설치되어 있고, 정상 동작하는지 확인

```
[node1 ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.1
 API version:       1.41
 Go version:        go1.13.15
~~~
Server: Docker Engine - Community
 Engine:
  Version:          20.10.1
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
~~~
```

② 아래 명령을 실행하여 실습에 필요한 파일 다운로드 및 환경구성

```
curl -O http://220.95.250.70:7778/dockeredu/docker_lab_playground.tar.gz
tar xfz docker_lab_playground.tar.gz
export JAVA_HOME=/root/jdk8
export M2_HOME=/root/maven3
export PATH=$JAVA_HOME/bin:$M2_HOME/bin:$PATH
```

- 복사 : Ctrl + insert
- 붙여넣기 : Shift + insert

도커 체험

docker 설치

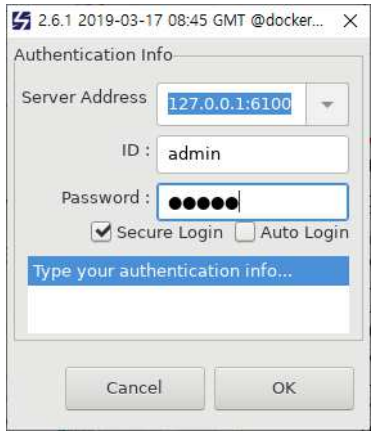
- **사전 작업 : 방화벽 해제**
  - systemctl stop firewalld
  - systemctl disable firewalld
  - systemctl status firewalld
- **docker 설치**
  - curl -sSL https://get.docker.com | sh
- **docker 데몬 시작**
  - systemctl enable docker
  - systemctl start docker
  - systemctl status docker
- **docker 실행 확인**
  - docker version

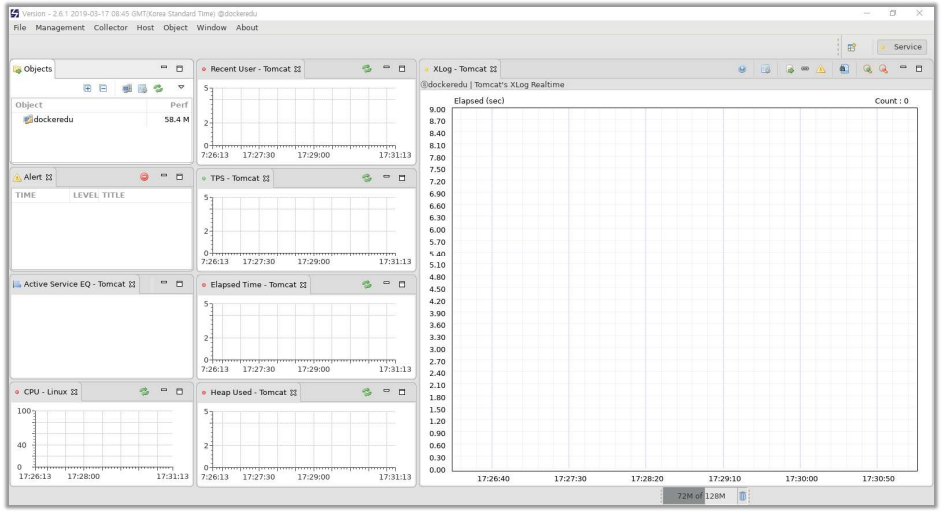
```
[root@dockeredu ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.0
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        7287ab3
 Built:             Tue Dec  8 18:57:35 2020
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.0
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       eeddea2
  Built:            Tue Dec  8 18:56:55 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.3
  GitCommit:        269548fa27e0089a8b8278fc4fc781d7f65a939b
 runc:
  Version:          1.0.0-rc92
  GitCommit:        ff819c7e9184c13b7c2607fe6c30ae19403a7aff
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Dockernization된 방명록 서비스 실행 (1/2)

① 스카우터 서버 시작

```
[root@dockeredu ~]# scouter.server.boot
nohup: redirecting stderr to stdout
  ____                  _
 / ___|  ___ ___  _   _| |_ ___ _ __
 \___ \ / __/   \| | | | __/ _ \ '__|
  ___) | (_| (+) | |_| | ||  __/ |
 |____/ \___\___/ \__,_|\__\___|_|
Open Source S/W Performance Monitoring
Scouter version 2.6.1
```



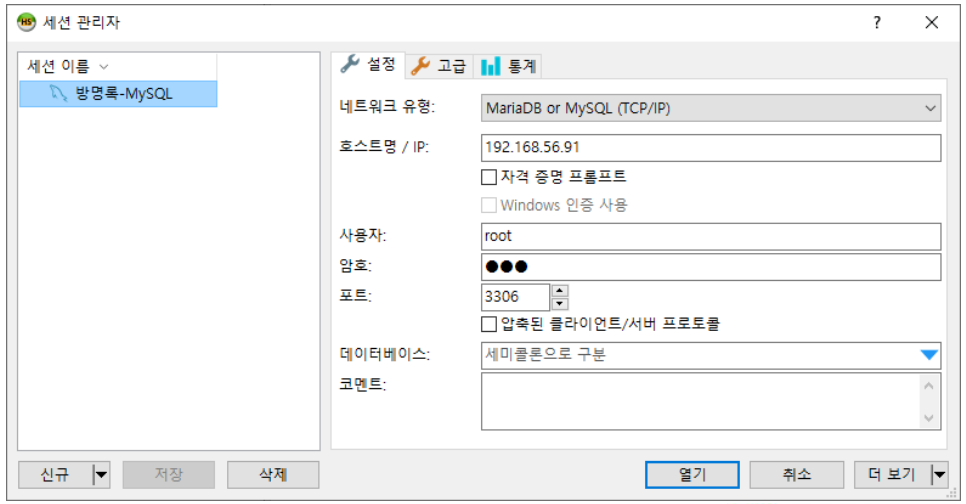


② 스카우터 클라이언트 시작

```
[root@dockeredu server]# scouter.client
[1] 11929
```

③ Docker Hub에서 yu3papa/mysql_hangul:1.0 이미지로 guestbookdb(MySQL) 컨테이너 시작

```
[root@dockeredu ~]# docker container run -d --name=guestbookdb -e MYSQL_ROOT_PASSWORD=edu -e MYSQL_DATABASE=guestbook -p 3306:3306
yu3papa/mysql_hangul:1.0
Unable to find image 'yu3papa/mysql_hangul:1.0' locally
1.0: Pulling from yu3papa/mysql_hangul
f8efbffe7b95: Pull complete
a3ed95caeb02: Pull complete
b2c02ce03be6: Pull complete
472b062f8300: Pull complete
43ce1c4a9027: Pull complete
0bdbb33da391: Pull complete
667e3bbf4c2d: Pull complete
17094828b65a: Pull complete
345523e055b2: Pull complete
6fa32ff2e443: Pull complete
e38c173fa1d4: Pull complete
Digest: sha256:8c8189041a255d9bab0f36f365c47ff65d0787744b96aee8834e40fec31b68da
Status: Downloaded newer image for yu3papa/mysql_hangul:1.0
cc6a8679f0347944d87e80f5784398b2730bc0d0ed00b4490a0ed3cf41de0cf1
```

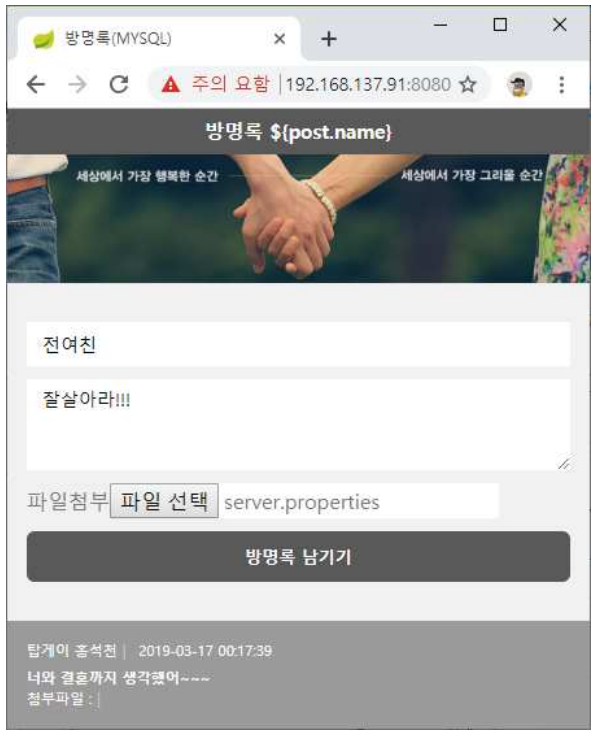


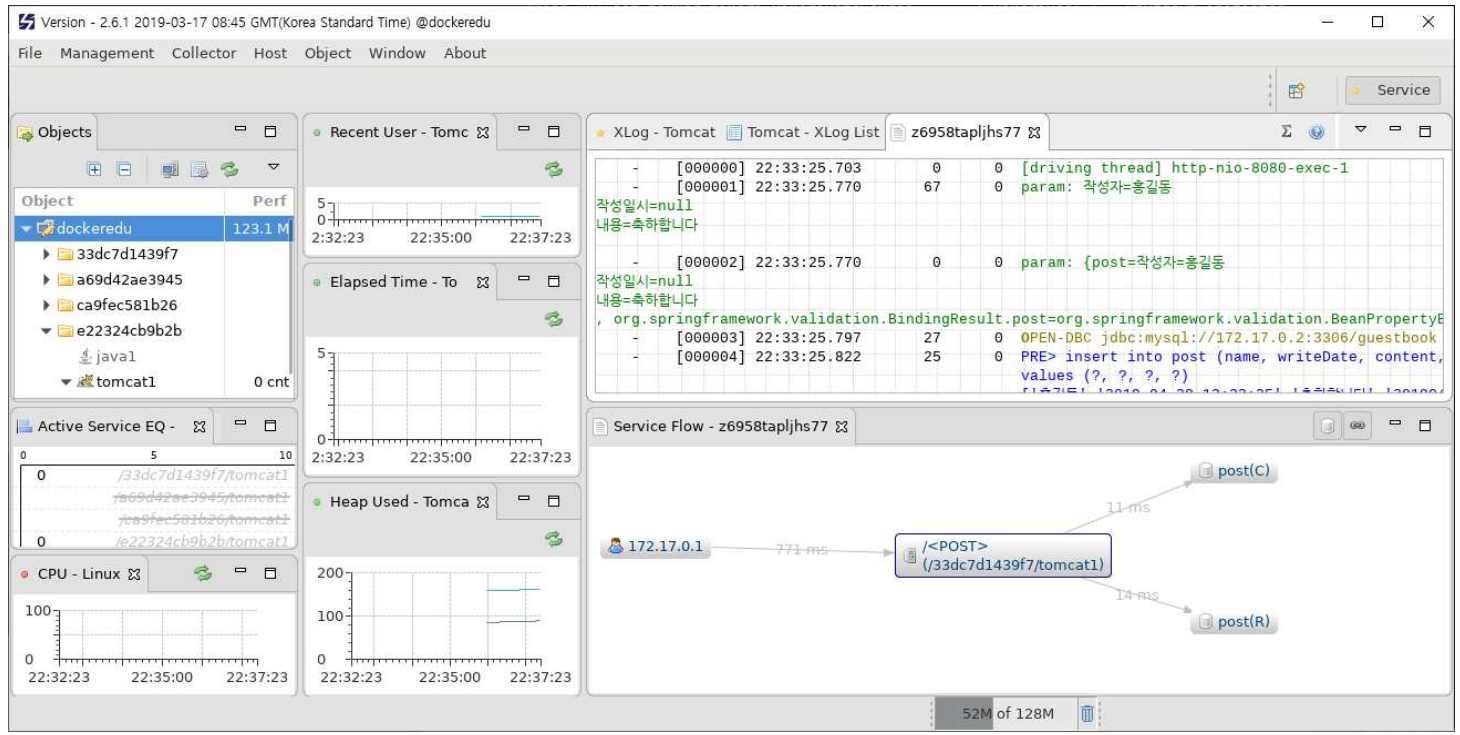
Dockernization된 방명록 서비스 실행 (2/2)

④ Docker Hub에 있는 yu3papa/guestbook:1.0 이미지로부터 guestbookdb 컨테이너를 링크하는 guestbookapp 컨테이너 시작

```
[root@dockeredu ~]# docker container run -d --name=guestbookapp -p 8080:8080 --link guestbookdb:mysql yu3papa/guestbook:1.0
Unable to find image 'yu3papa/guestbook:1.0' locally
1.0: Pulling from yu3papa/guestbook
~~~
8c1ad07205c8: Pull complete
Digest: sha256:6973c0443c4b69ad32dfe08a7f8149cf841640c6468e45e40fd7858e5eeb3c3f
Status: Downloaded newer image for yu3papa/guestbook:1.0
7d173bc1861f33d59e9528fe5851ac08662d185cc5cc6523d6938a9ff171e962
```

⑤ http://192.168.56.91:8080 주소로 접속하여 방명록 작성하고 스카우터를 이용하여 트랜잭션 프로파일 확인





docker-compose를 이용한 나만의 Cloud Storage 만들기 (1/2)

- **ownCloud**
  - https://github.com/owncloud-docker/server

① ownCloud 폴더 생성하고 이동

```
[root@dockeredu ~]# mkdir ~/ownCloud; cd $_
[root@dockeredu ownCloud]#
```

② docker-compose용 환경변수 만들기

```
[root@dockeredu ownCloud]# cat << EOF >| .env
OWNCLOUD_VERSION=10.0
OWNCLOUD_DOMAIN=localhost
ADMIN_USERNAME=admin
ADMIN_PASSWORD=admin
HTTP_PORT=80
EOF
```

③ docker-compose.yml 다운로드

```
[root@dockeredu ownCloud]# wget http://jadecross.iptime.org:7778/dockeredu/owncloud/docker-compose.yml
또는 sw\LAB_Docker\ownCloud\docker-compose.yml 파일 이동
~~~
Length: 1649 (1.6K) [text/plain]
Saving to: ‘docker-compose.yml’
100%[=========================================================================================================>] 1,649       --.-K/s   in 0s
2019-05-01 15:11:20 (118 MB/s) - ‘docker-compose.yml’ saved [1649/1649]
```

④ docker-compose up -d 로 실행

```
[root@dockeredu ownCloud]# docker-compose up -d
Creating network "owncloud_default" with the default driver
Creating owncloud_db_1    ... done
Creating owncloud_redis_1 ... done
Creating owncloud_owncloud_1 ... done
```

docker-compose 설치

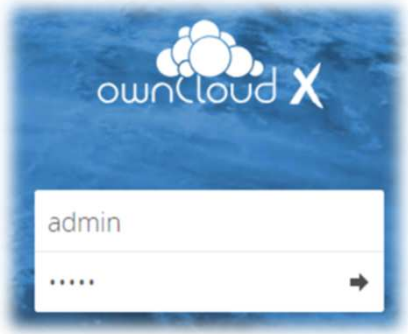
- **binary 다운로드**
  - curl -L https://github.com/docker/compose/releases/download/1.24.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

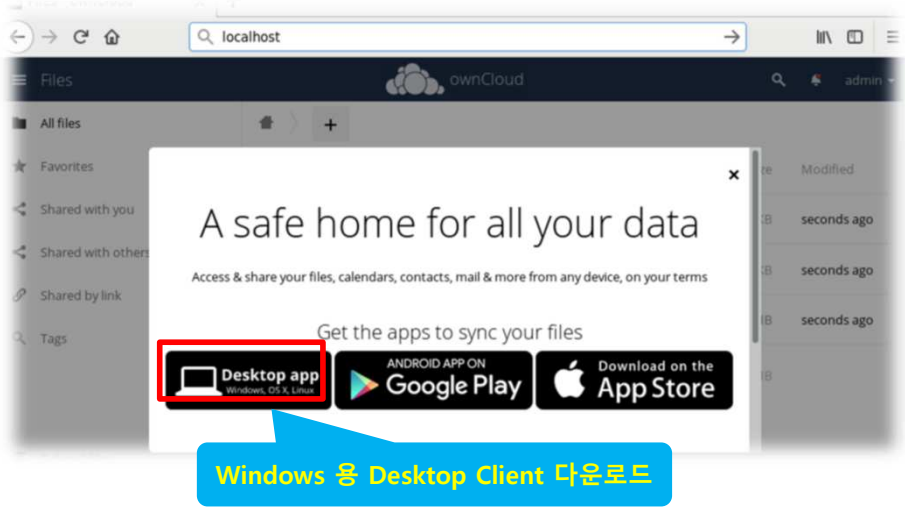
- **실행 퍼미션 추가**
  - chmod +x /usr/local/bin/docker-compose

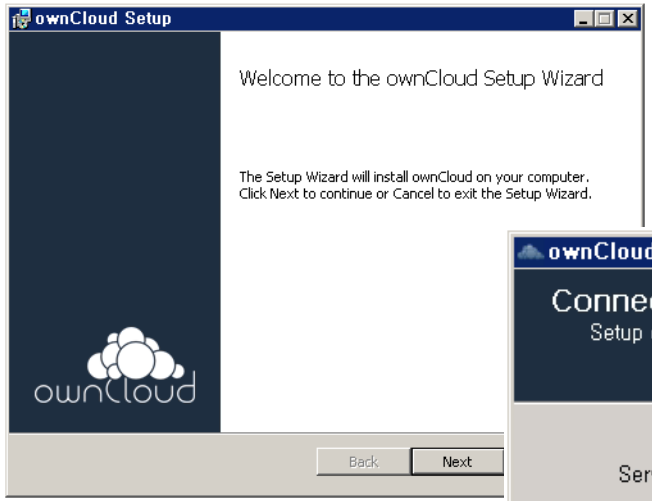
- **설치 확인**
  - docker-compose --version
    - docker-compose version 1.24.0, build 0aa59064

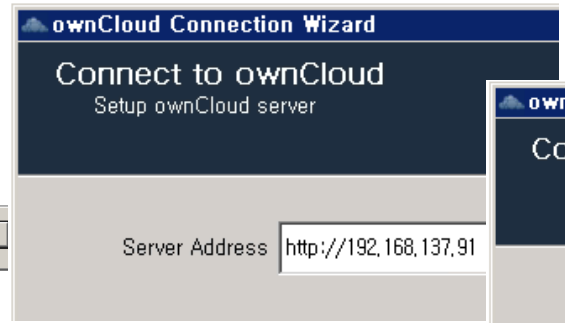
docker-compose를 이용한 나만의 Cloud Storage 만들기 (2/2)

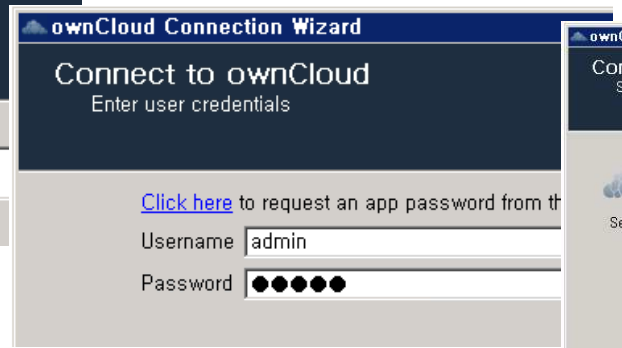
- **ownCloud 서버 접속**
  - http://192.168.56.91
  - 계정
    - admin / admin
- **Desktop Client 설치**

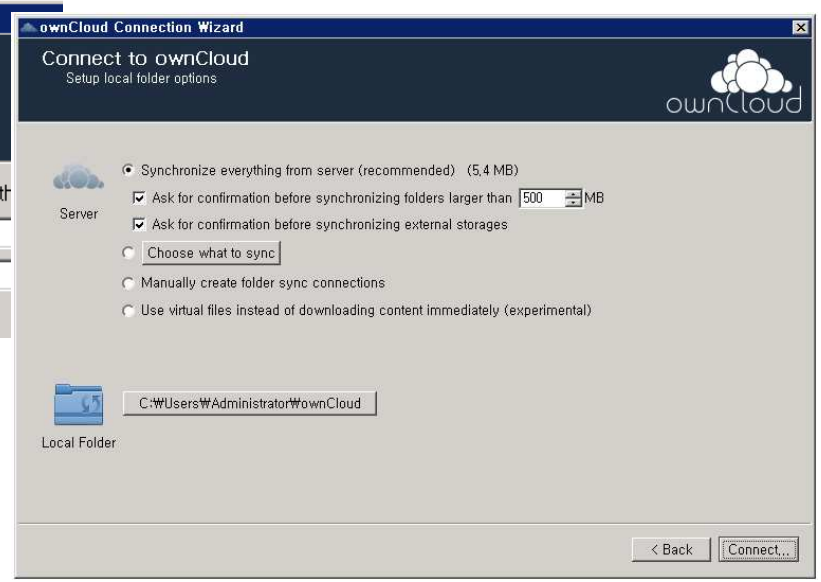












도커 개요

보잘것 없어 보이는 철재 상자 – 컨테이너

철도 업체나 트럭 업체는 항구에 내려놓은 화물이 어떻게 배에 실리는 지에는 관심을 두지 않았습니다.

회사마다, 분야마다 다른 형태의 상자와 도구를 쓰다보니 효율성이 떨어져 상자를 쓰지 않느니만 못했습니다.

항구 노동자들 역시 정치적 영향력이 큰 노조를 조직해서 하역 작업의 기계화를 막았습니다.

참고 : http://www.opennaru.com/open-source/containers-metaphor-for-what-docker-is/

육지와 바다는 수송 시스템이 전혀 다른 형태로 되어있습니다.

예전에는 화물을 트럭으로 부두까지 실어온 후 선박이 도착하면 다른 화물들과 함께 배의 측면으로 옮겨 부두노동자가 포장하여 적재하였습니다.

그 과정에서 화물의 손상, 분실, 시간, 비용이 크게 발생하였으며, 운송 수단으로서 신뢰도 낮았습니다.

19세기 후반부터 미국과 유럽의 수많은 운송업체들은 박스 형태의 운송을 시도하였습니다.

물건을 나를 때 상자에 넣어서 운반하는 것이 편리하다는 것은 누구나 알고 있는 사실이지만, 문제는 표준화였습니다.

배를 소유한 해운업체는 구태여 자기 돈을 들여 컨테이너를 이동시킬 크레인을 항구에 설치하려 하지 않았습니다.

1950년대 컨테이너의 등장

Malcolm P. McLean

트럭 운전사였던 말콤 맥린 (Malcolm P. McLean)은 트럭이나 선박에 화물을 적재할 때 걸리는 오랜 대기 시간을 해결하기 위해 "박스" 만 분리하여 배에 싣는 아이디어를 생각하게 됩니다.

이것은 교통 수단이 무엇이든지 박스를 오픈 하지 않고 트럭, 선박, 철도에서 즉시 화물을 교환할 수 있는 혼합수송 방법입니다.

그는 해운업계에 들어오기 전 트럭 운전과 트럭 운송업체를 운영해 봤기 때문에 표준화를 통해 화물을 효율적으로 운송하는 방법이 매우 어려운 일이라는 것을 잘 알고 있었습니다.

하지만 그는 해운사와 육송 운송업체들이 각자 자신들의 기득권을 지키고 서로의 영역을 침범하지 못하도록 방어하는 데 열을 올리는 동안 선박과 트럭이 함께 쓸 수 있는 공통 규격의 컨테이너 개발에 힘을 기울였습니다.

또한 집요하게 노조와 정부 관계자들을 설득했습니다.

맥린은 바다와 육지를 포괄한다는 뜻의 시랜드(Sea-Land)사를 설립하고 1956년 3월에 최초로 컨테이너를 이용하여 해양 화물 운송 시작하게 되었습니다.

그는 컨테이너를 이용한 화물 운송의 혁명을 이끌면서, 2007년에는 포브스(Forbes)에서 선정한 '20세기 후반 세계를 바꾼 인물 15인'에 선정됩니다.

1950년대에 트럭운송업을 하던 Malcom McLean 과 그의 헤드 엔지니어인 Keith Tantlinger가 컨테이너를 표준화하여 세계를 변화시켰습니다.

그 당시에 배를 수작업으로 선적하는 비용은 톤당 5.86 달러였으며, 컨테이너를 사용하면 톤당 16 센트 밖에 안되며 비용은 36 배 절감됩니다.

또한 컨테이너 화는 또한 선박의 적재 및 하역 작업에 소요되는 시간을 크게 줄였습니다.

참고 : http://www.opennaru.com/open-source/containers-metaphor-for-what-docker-is/

오픈소스 개념으로 컨테이너 기술을 공유한 Keith Tantlinger

Keith Tantlinger

McLean 의 헤드 엔지니어였던 Tantlinger는 화물을 컨테이너에 적재하고, 트럭으로 운반하고, 크레인으로 운반할 수 있는 코너 캐스팅과 트위스트록 (Twistlock) 매커니즘과 같은 컨테이너로 복합 운송을 할 수 있도록 하는 기반 기술을 개발하였습니다.

더욱 중요한 것은 **Tantlinger가 McLean 에게 라이센스 없이 컨테이너 기술을 사용할 수 있게 특허공개를 설득**한 것입니다.

오늘날 오픈소스을 통한 혁신 사례에서 보듯이 컨테이너 기반 기술을 공개하여 모든 항구와 모든 화물선에서 컨테이너를 이용하면 화물의 운송의 혁신이 일어날 것을 확신하였습니다.

Tantlinger는 소프트웨어 용어로 오픈소스를 컨테이너 기술로 확산할 수 있는 아이디어에 활용한 것입니다.

Twistlock

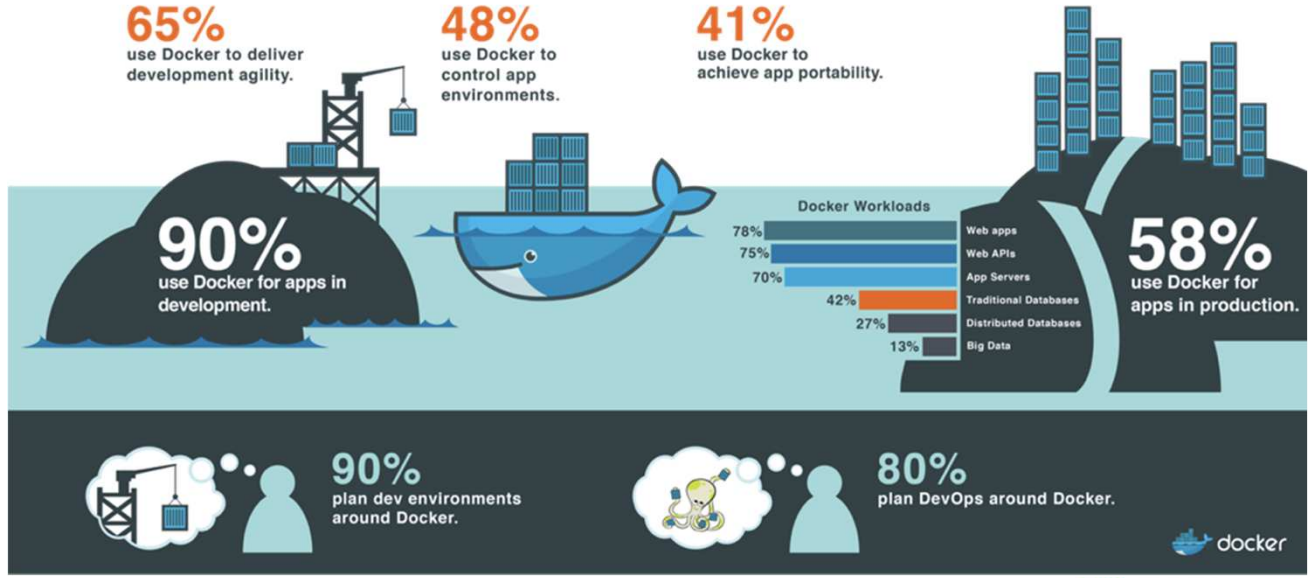
참고 : http://www.opennaru.com/open-source/containers-metaphor-for-what-docker-is/

화물운송 분야의 컨테이너 -vs- 도커 컨테이너 공통점은?

- **Docker가 구축하는 애플리케이션에 대한 표준화된 컨테이너는 화물 운송 분야의 컨테이너는 대한 은유입니다.**
  - 컨테이너가 Tantlinger 에 의해 기술적인 표준화가 되기 전부터 오랫동안 구축된 개념입니다. HP, 오라클, IBM 과 같은 대형 벤더들은 수년간 켄테이너 기술을 사용해 왔으며, 특히 구글은 내부 프로젝트에서 매우 유사한 구현 방식을 사용하였습니다.
    도커는 오픈소스와 커뮤니티를 중심으로 기술의 표준화와 발전을 이끌고 있습니다.
  - 화물 컨테이너의 내부 화물은 운송에 중요하진 않습니다. 세계의 모든 선박과 트럭 그리고 크레인은 컨테이너 규격에 적합해야 합니다. 마찬가지로 도커 컨테이너도 어떤 애플리케이션(관련 파일, 프레임워크, 의존성 등)이 내부에 있는지 중요하지 않습니다.
    컨테이너는 모든 리눅스 배포판에서 실행되며, AMAZON AWS, Micrsoft Azue, Google Cloud Platform, Rackplace 등 모든 퍼블릭 클라우드 환경에서 운영됩니다.
  - 해외로 이사를 간다고 가정을 하면 사실상 컨테이너에 이사짐을 넣은 후 트럭으로 이동하여, 크레인으로 배에 옮겨져 다른 나라로 운송합니다. 마찬가지로 컨테이너를 이용하면 개발자가 로컬 시스템에서 애플리케이션을 빌드하고 테스트 할 수 있으며, 애플리케이션을 서버에 Push할 수 있습니다. 개발자는 컨테이너로 배포하게 되면 개발환경이나 운영환경이나 동일하게 동작할 것이라는 것을 알수 있습니다.

  - 컨테이너는 컨테이너의 라이프사이클을 손쉽게 관리할 수 있게 할 뿐만 아니라 "Run Anywhere" 비전처럼 클라우드 서비스 어디에서도 운영될 수 있습니다.
    컨테이너는 가상화와 비교했을 때 클라우드 마이그레이션 과정을 비교하기 어려울 정도로 단순화되고 효율적입니다

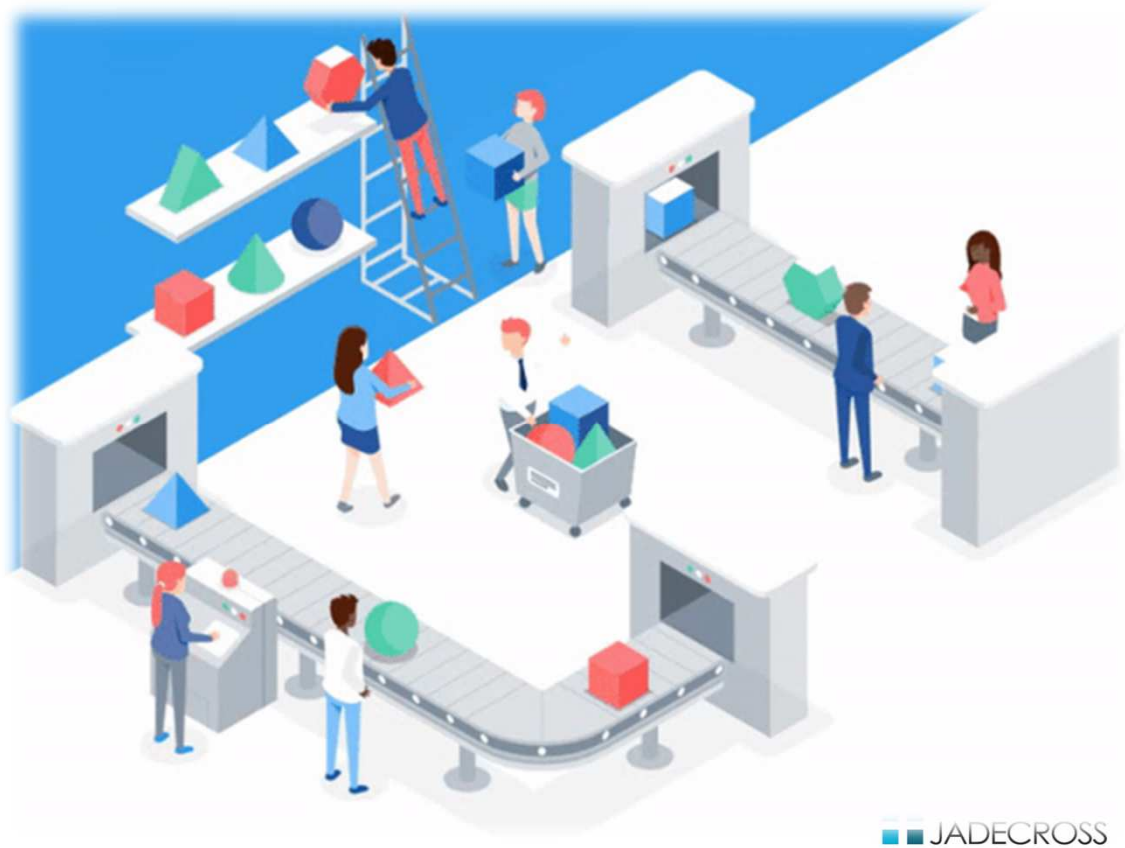
  - IT 기술에서의 컨테이너화는 개발과 테스트 단계에서 사용되는 시스템 환경을 그대로 물리서버, 가상화, 클라우드와 같이 상이한 환경에서 배포하고 운영할 수 있도록 합니다. 이는 리눅스와 같은 개방형 기술을 기반으로 하고 있어 기존의 가상화 기술이 가지고 있던 벤더 종속성 없이 Linux 의 기본 기능만으로 손쉽게 컨테이너 환경을 구축할 수 있습니다.



참고 : http://www.opennaru.com/open-source/containers-metaphor-for-what-docker-is/

Enterprise Container Platform for High Velocity Innovation

Securely build, share and run any application, anywhere

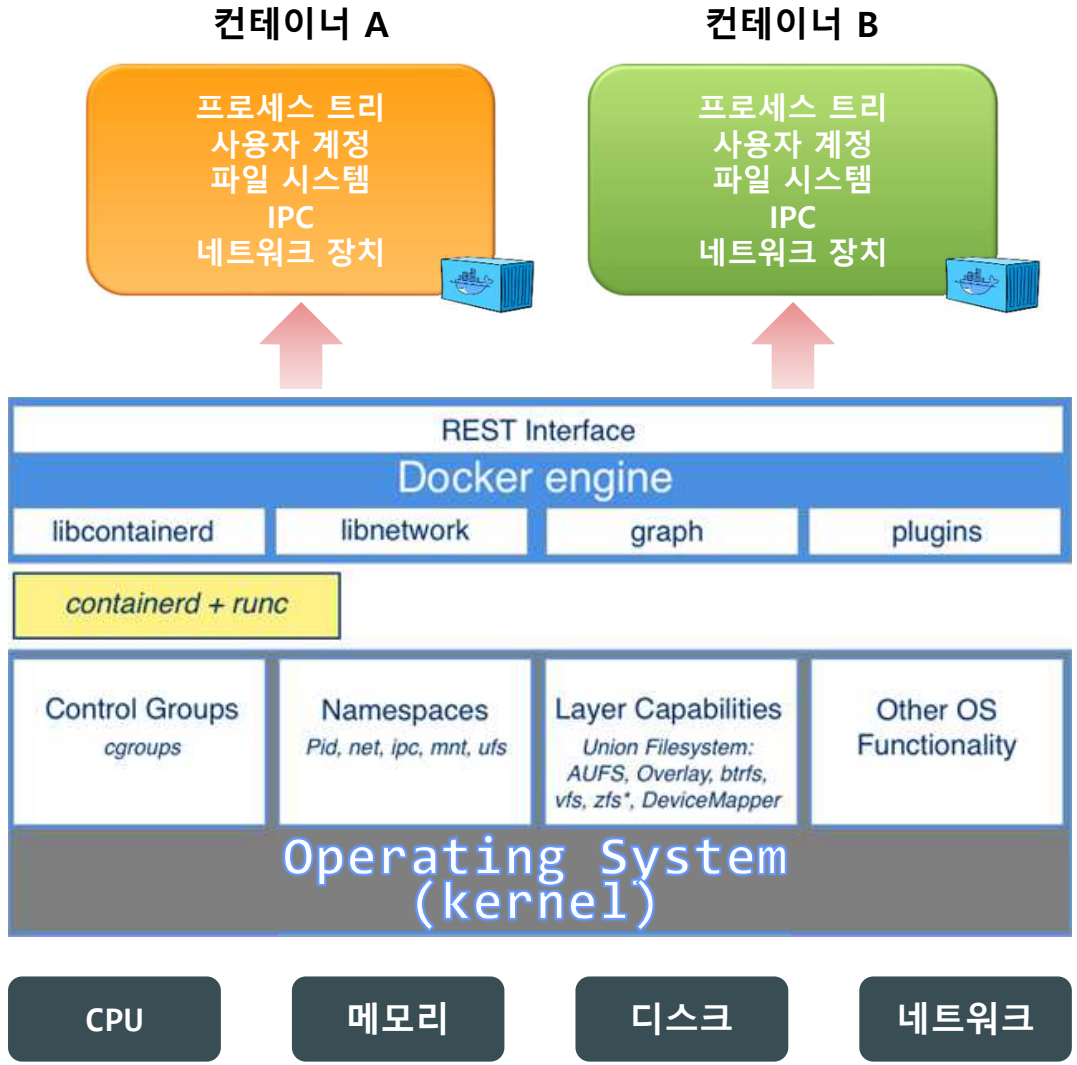


컨테이너 기반기술

컨테이너의 특징

- **컨테이너의 특징**
  - 호스트와 애플리케이션을 분리 : OS 레벨 가상화
  - 적은 시스템 리소스와 디스크 공간 소비
  - 리소스 사용에 유연성과 효율성 제공
  - 가상머신과는 달리 개별적인 커널을 갖지 않음

- **컨테이너 기반 기술**
  - cgroup
    - 리소스에 대한 제한과 우선 순위를 제어
  - namespace
    - 애플리케이션에 대한 **격리(isolation) 환경**을 제공



파일시스템 격리- chroot (1/2)

- **chroot**
  - 프로세스에게 격리된 루트 디렉토리 제공
  - 1979년에 UNIX 에 포함됨
  - httpd(아파치 웹서버), vsftpd(FTP 서버)등 이미 많은 패키지에서 내부적으로 사용되고 있음

```
CHROOT(1)                          User Commands

NAME
       chroot - run command or interactive shell with special root directory

SYNOPSIS
       chroot [OPTION] NEWROOT [COMMAND [ARG]...]
       chroot OPTION

DESCRIPTION
       Run COMMAND with root directory set to NEWROOT.

       --userspec=USER:GROUP
              specify user and group (ID or name) to use
```

파일시스템 격리- chroot (2/2)

- **newroot 디렉토리를 생성하고 루트 디렉토리로 설정하여 파일 시스템 격리**

① newroot 하위에 bin lib64 디렉토리 생성

```
[root@dockeredu ~]# mkdir -p ~/newroot/bin newroot/lib64
```

② bash, ls, date 명령어가 사용하는 라이브러리 확인

```
[root@dockeredu ~]# ldd /bin/bash
        linux-vdso.so.1 =>  (0x00007ffec73e1000)
        libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007f326b8ce000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007f326b6ca000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f326b2fd000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f326baf8000)
[root@dockeredu ~]# ldd /bin/ls
        linux-vdso.so.1 =>  (0x00007ffd1d851000)
        libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f9ea3a35000)
        libcap.so.2 => /lib64/libcap.so.2 (0x00007f9ea3830000)
        libacl.so.1 => /lib64/libacl.so.1 (0x00007f9ea3627000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f9ea325a000)
        libpcre.so.1 => /lib64/libpcre.so.1 (0x00007f9ea2ff8000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007f9ea2df4000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9ea3c5c000)
        libattr.so.1 => /lib64/libattr.so.1 (0x00007f9ea2bef000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f9ea29d3000)
[root@dockeredu ~]# ldd /bin/date
        linux-vdso.so.1 =>  (0x00007ffe83b7a000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f1cdf787000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f1cdfb54000)
```

③ bash, ls, date 명령어가 사용하는 라이브러리 복사

```
[root@dockeredu ~]# cp /lib64/libtinfo.so.5 newroot/lib64
[root@dockeredu ~]# cp /lib64/libdl.so.2 newroot/lib64
[root@dockeredu ~]# cp /lib64/libc.so.6 newroot/lib64
[root@dockeredu ~]# cp /lib64/ld-linux-x86-64.so.2 newroot/lib64
[root@dockeredu ~]# cp /lib64/libselinux.so.1 newroot/lib64
[root@dockeredu ~]# cp /lib64/libcap.so.2 newroot/lib64
[root@dockeredu ~]# cp /lib64/libacl.so.1 newroot/lib64
[root@dockeredu ~]# cp /lib64/libpcre.so.1 newroot/lib64
[root@dockeredu ~]# cp /lib64/libattr.so.1 newroot/lib64
[root@dockeredu ~]# cp /lib64/libpthread.so.0 newroot/lib64
```

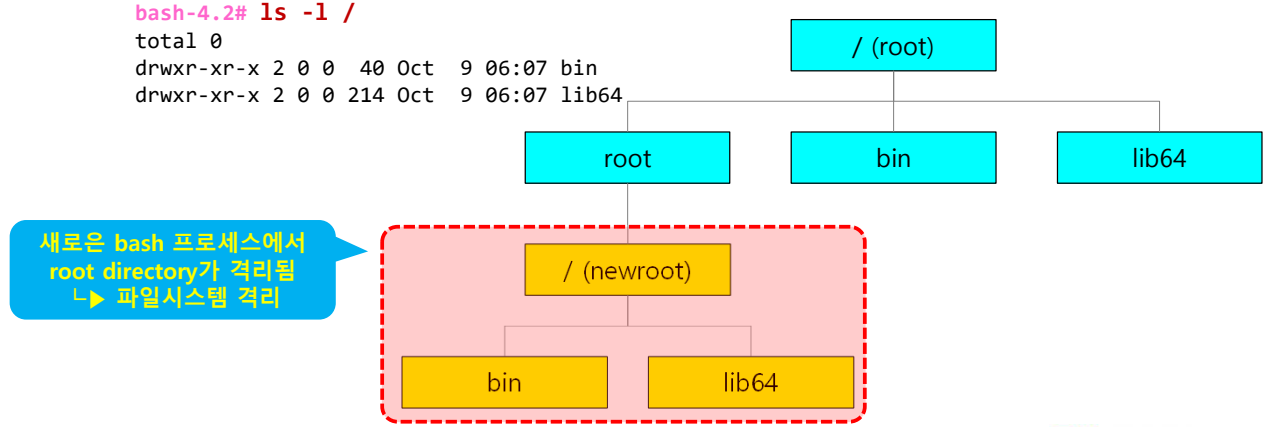
④ bash, ls, date 명령어 복사

```
[root@dockeredu ~]# cp /bin/bash newroot/bin
[root@dockeredu ~]# cp /bin/ls newroot/bin
[root@dockeredu ~]# cp /bin/date newroot/bin

[root@dockeredu ~]# tree newroot (또는 nautilus 실행)
```

```
[root@dockeredu ~]# tree newroot
newroot
├── bin
│   ├── bash
│   ├── date
│   └── ls
└── lib64
    ├── ld-linux-x86-64.so.2
    ├── libacl.so.1
    ├── libattr.so.1
    ├── libcap.so.2
    ├── libc.so.6
    ├── libdl.so.2
    ├── libpcre.so.1
    ├── libpthread.so.0
    ├── libselinux.so.1
    └── libtinfo.so.5
```

⑤ newroot를 root directory로 설정

```
[root@dockeredu ~]# chroot newroot
bash-4.2# pwd
/
bash-4.2# /bin/ls -l /
total 0
drwxr-xr-x 2 0 0  40 Oct  9 06:07 bin
drwxr-xr-x 2 0 0 214 Oct  9 06:07 lib64
bash-4.2# /bin/date
Tue Oct  9 06:12:07 UTC 2018

bash-4.2# PATH=/bin:$PATH
bash-4.2# date
Tue Oct  9 06:14:04 UTC 2018
bash-4.2# ls -l /
total 0
drwxr-xr-x 2 0 0  40 Oct  9 06:07 bin
drwxr-xr-x 2 0 0 214 Oct  9 06:07 lib64
```



namespaces (1/5)

- **리눅스 네임스페이스**
  - 동일 시스템 내부에서, 각각 별개의 독립된 공간을 사용하는 것처럼 격리된 환경을 제공하는 리눅스 커널의 기능
  - H/W를 가상화하는 hypervisor와 달리 커널 레벨에서 process의 실행 환경 자체를 격리하는 기술
  - 리눅스 네임스페이스를 구현할 때는 `unshare()`와 `setns()` 시스템 콜을 이용해 만들고 IPC, Network, Mount, PID, UTS, User의 6개의 상수 플래그를 `clone()`, `unshare()`, `setns()` 시스템 콜에 전달해 수행
- **리눅스 커널에서 지원하는 네임스페이스 종류**

namespace	description
IPC	System V IPC resources : shared memory, message queue, semaphore
Network	A physical or virtual network interfaces, full networking stacks and firerwall rules
Mount	Filesystem mount points
PID	Process IDs so that each container can have its own init(aka PID 1)
UTS	Hostname and NIS domain name
User	User and group IDs allow for a separate root user in each container

- **네임스페이스를 지원하는 시스템 콜**

시스템 콜	설명
clone()	새로운 프로세스를 만들고 이를 새로 지정된 네임스페이스에 연결
unshare()	현재 프로세스를 새로 지정된 네임스페이스에 연결
setns()	이미 존재하는 네임스페이스에 프로세스를 연결

namespaces (2/5)

- **unshare 명령**

```
UNSHARE(1)                                                    User Commands
                                    UNSHARE(1)

NAME
       unshare - run program with some namespaces unshared from parent

SYNOPSIS
       unshare [options] program [arguments]

DESCRIPTION
       Unshares the indicated namespaces from the parent process and then executes the specified program.  The na
mespaces to be unshared are indicated via options.
       Unshareable namespaces are:

       mount namespace
              Mounting and unmounting filesystems will not affect the rest of the system (CLONE_NEWNS flag), except for f
ilesystems which are explicitly marked as shared
              (with mount --make-shared; see /proc/self/mountinfo or findmnt -o+PROPAGATION for the shared flags).

              unshare automatically sets propagation to private in the new mount namespace to make sure that the new name
space is really unshared. This feature is possible
              to disable by option --propagation unchanged.  Note that private is the kernel default.

       UTS namespace
              Setting hostname or domainname will not affect the rest of the system.  (CLONE_NEWUTS flag)

       IPC namespace
              The process will have an independent namespace for System V message queues, semaphore sets and shared memor
y segments.  (CLONE_NEWIPC flag)

       network namespace
              The process will have independent IPv4 and IPv6 stacks, IP routing tables, firewall rules, the /proc/net an
d /sys/class/net directory trees, sockets, etc.
              (CLONE_NEWNET flag)

       pid namespace
              Children will have a distinct set of PID to process mappings from their parent.  (CLONE_NEWPID flag)

       user namespace
              The process will have a distinct set of UIDs, GIDs and capabilities.  (CLONE_NEWUSER flag)
```

namespaces (3/5)

▪ unshare -m 명령을 사용하여 mount namespace를 격리

① /tmp/mount_ns 디렉토리를 생성하고 unshare 명령을 사용하여 mount namespace를 분리한다.

```
[root@dockeredu ~]# mkdir /tmp/mount_ns
[root@dockeredu ~]# unshare -m /bin/bash
```

② mount namespace가 분리된 현재 process의 mount정보를 확인

```
[root@dockeredu ~]# ls -l /proc/$$/ns/mnt
lrwxrwxrwx 1 root 0 Oct  9 18:35 /proc/12286/ns/mnt -> mnt:[4026532116]
```

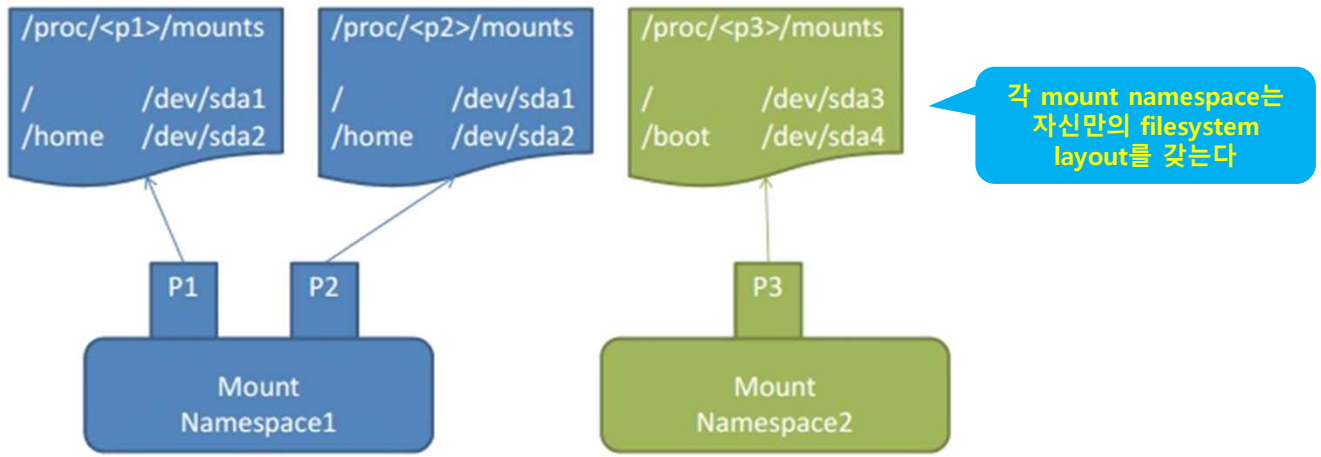
③ tmpfs 파일시스템을 해당 디렉토리에 mount 한 뒤 파일시스템 정보를 확인

```
[root@dockeredu ~]# mount -t tmpfs tmpfs /tmp/mount_ns

[root@dockeredu ~]# df -h | grep mount_ns
tmpfs                2.0G     0  2.0G   0% /tmp/mount_ns
```

④ 새로운 세션을 열어 분리한 mount filesystem이 보이지 않는 것을 확인한다

⑤ 현재 process의 mount정보를 확인

```
[root@dockeredu ~]# ls -l /proc/$$/ns/mnt
lrwxrwxrwx 1 root 0 Oct  9 18:45 /proc/12341/ns/mnt -> mnt:[4026531840]


[root@dockeredu ~]# df -h | grep mount_ns

[root@dockeredu ~]# mount | grep mount_ns
```



namespaces (4/5)

- **UTS namespace** (Unix Timesharing namespace)
  - process에게 호스트 이름과 도메인 이름을 네임스페이스 별로 격리
  - 호스트 이름에 종속적인 애플리케이션에서 중요한 의미를 갖는다.

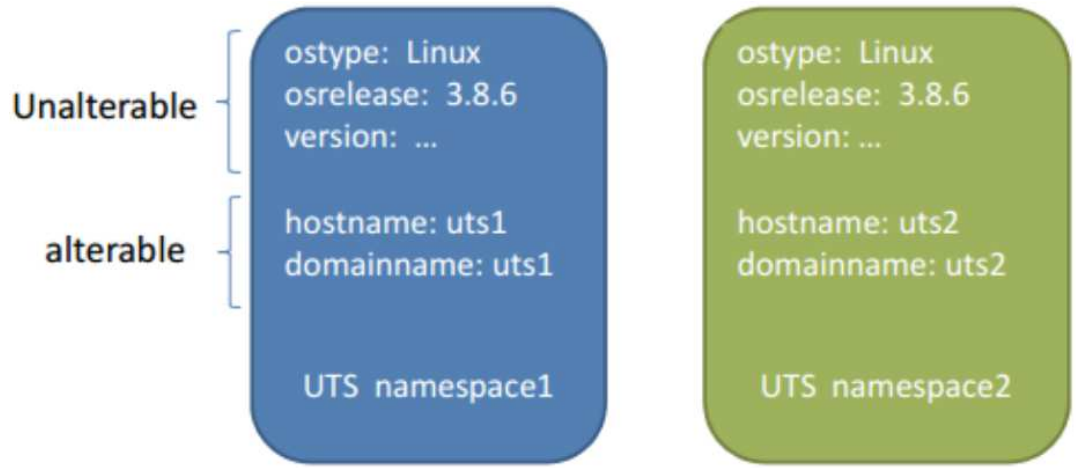
- **UTS 네임스페이스를 격리하고 hostanme을 변경해 본다.**

① 최초 접속한 세션에서의 호스트네임을 확인

```
[root@dockeredu ~]# hostname
dockeredu
```

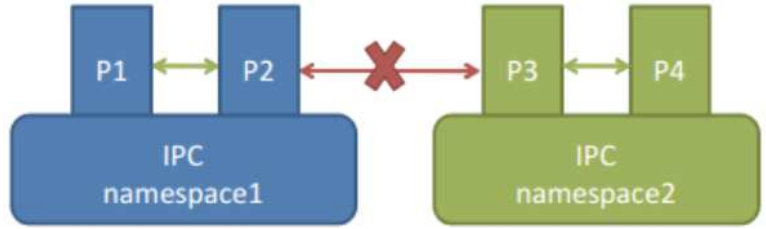
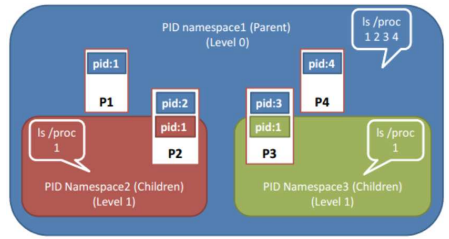
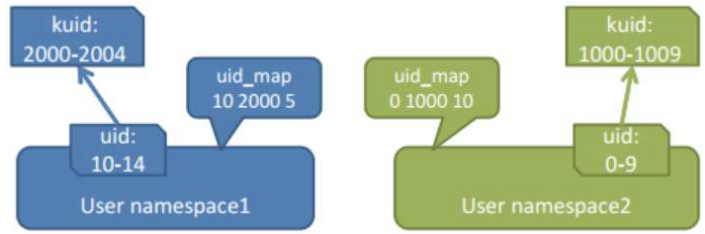
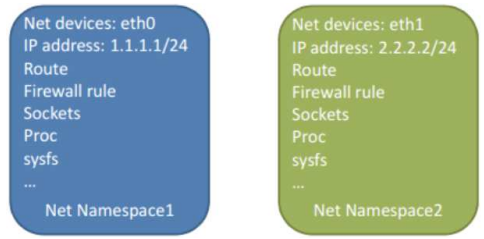
② unshare -u 명령어를 이용하여 UTS namespace를 격리한다. 격리 후 새로운 hostname으로 setting하고 호스트네임이 변경된 것을 확인

```
[root@dockeredu ~]# unshare -u /bin/bash
[root@dockeredu ~]# hostname con01
[root@dockeredu ~]# hostname
con01
```



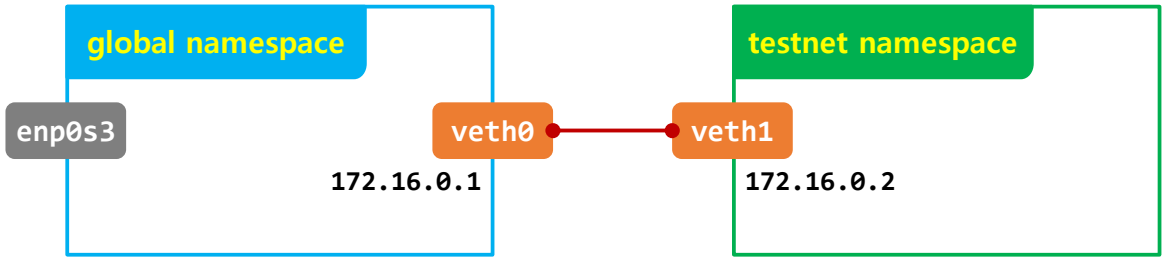
namespaces (5/5)

- 기타 네임스페이스

네임스페이스	설명	
IPC	프로세스간 데이터 교환 및 프로세스와 스레드간의 작업을 동기화하는 기능을 제공하는 namespace이다. semaphore, file locking, mutex 와 같은 자원에 대한 접근제어(primitives)를 제공하며, 컨테이너에서 실제 프로세스를 분리하기 위해서도 필요하다.	
PID	프로세스 ID를 분할하여 관리하기 위해 필요한 namespace이다. init 프로세스만이 가질 수 있는 PID 1번을 독립적으로 추가할당하며, 동일한 OS에서 init 프로세스 뿐만 아니라 여러 프로세스가 PID 충돌없이 실행 가능하게 해준다.	
User	프로세스가 namespace 내부와 기본 namespace 간에 각기 다른 사용자 및 그룹 ID를 가질 수 있도록 지원한다. kernel v3.8에서 소개되어 아직 많은 시스템에서 지원하지는 않으며, 외부에서는 권한이 없는 ID를 갖고 있는 프로세스가, 자신이 생성한 컨테이너 내부에서는 루트 권한으로 실행 가능하다.	
Network	네트워크 장치, 주소, 경로 및 방화벽 규칙 같은 네트워킹 자원을 격리한다. 네트워크 스택의 논리적 복사본을 효과적으로 생성하여 여러 네임스페이스가 동일 포트로 다수의 서비스 제공 가능하며, iproute2 package 등을 이용하여 구현 가능하다.	

네트워크 네임스페이스 (1/4)

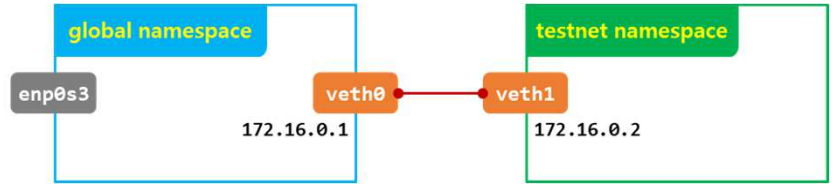
- **LAB**
  - testnet 네트워크 네임스페이스를 생성하고 linux bridge기술을 이용하여 2개의 네트워크 네임스페이스간에 통신이 가능하도록 설정



① ip 명령을 이용하여 L2, L3 계층의 네트워크 인터페이스를 확인

```
[root@dockeredu ~]# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.91/24 brd 192.168.56.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe1b:adc4/64 scope link
       valid_lft forever preferred_lft forever

[root@dockeredu ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
```

네트워크 네임스페이스 (2/4)



② 신규 네트워크 네임스페이스 생성 : testnet 네임스페이스 생성

```
[root@dockeredu ~]# ip netns add testnet
[root@dockeredu ~]#
```

③ testnet 네임스페이스의 네트워크 정보보기

```
[root@dockeredu ~]# ip netns exec testnet ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
[root@dockeredu ~]#
```

④ testnet 네임스페이스의 lo 네트워크 활성화

```
[root@dockeredu ~]# ip netns exec testnet ip link set lo up
[root@dockeredu ~]# ip netns exec testnet ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

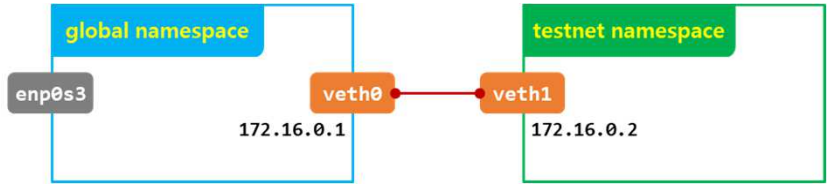
⑤ linux bridge 생성 : veth 인터페이스 설정

```
[root@dockeredu ~]# ip link add veth0 type veth peer name veth1
[root@dockeredu ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
3: veth1@veth0: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether aa:4f:97:ae:9f:63 brd ff:ff:ff:ff:ff:ff
4: veth0@veth1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 76:87:ad:36:c2:aa brd ff:ff:ff:ff:ff:ff
```

⑥ veth0 와 veth1 이 pair 로 맺어진것을 ethtool 명령을 이용하여 확인

```
[root@dockeredu ~]# ethtool -S veth1
NIC statistics:
     peer_ifindex: 4
```

네트워크 네임스페이스 (3/4)



⑦ veth1 을 testnet 네임스페이스에 할당

```
[root@dockeredu ~]# ip link set veth1 netns testnet
```

⑧ veth1이 global 네트워크 네임스페이스에서 사라지고 testnet 네임스페이스로 격리됨

```
[root@dockeredu ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
4: veth0@if3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 76:87:ad:36:c2:aa brd ff:ff:ff:ff:ff:ff link-netnsid 0
[root@dockeredu ~]# ip netns exec testnet ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: veth1@if4: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether aa:4f:97:ae:9f:63 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

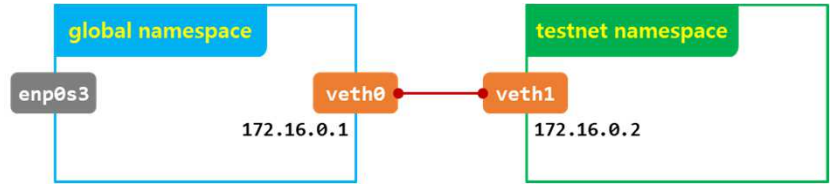
⑨ veth0 에 IP 설정

```
[root@dockeredu ~]# ip link set veth0 up
[root@dockeredu ~]# ip address add 172.16.0.1/24 dev veth0
[root@dockeredu ~]# ip addr show dev veth0
4: veth0@if3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group default qlen 1000
    link/ether 76:87:ad:36:c2:aa brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.0.1/24 scope global veth0
       valid_lft forever preferred_lft forever
```

⑩ veth1 에 IP 설정

```
[root@dockeredu ~]# ip netns exec testnet ip link set veth1 up
[root@dockeredu ~]# ip netns exec testnet ip address add 172.16.0.2/24 dev veth1
[root@dockeredu ~]# ip netns exec testnet ip address show dev veth1
3: veth1@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether aa:4f:97:ae:9f:63 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.0.2/24 scope global veth1
       valid_lft forever preferred_lft forever
```

네트워크 네임스페이스 (4/4)



⑪ veth0에서 veth1으로 통신이 되는지 확인

```
[root@dockeredu ~]# ping -c 2 172.16.0.2
PING 172.16.0.2 (172.16.0.2) 56(84) bytes of data.
64 bytes from 172.16.0.2: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 172.16.0.2: icmp_seq=2 ttl=64 time=0.119 ms

--- 172.16.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.063/0.091/0.119/0.028 ms
```

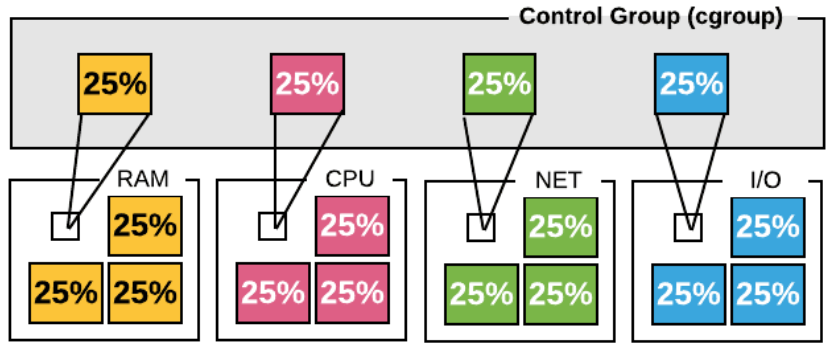
⑫ veth1에서 veth0으로 통신이 되는지 확인

```
[root@dockeredu ~]# ip netns exec testnet ping -c 2 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 172.16.0.1: icmp_seq=2 ttl=64 time=0.145 ms

--- 172.16.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.060/0.102/0.145/0.043 ms
```

Control Groups

- **Linux Kernel에서 제공하는 기능**
  - 프로세스 그룹에 대한 리소스 제한(Limit), 격리(Isolation), 모니터링 (CPU, Memory, Storage, Network I/O)
  - 2006년 9월 부터 'Process Container'로 개발시작
  - 2007년 단어의 모호함 때문에 cgroups 로 이름변경
  - 2008년 1월 Linux Kernel 2.6.24에 적용
- **cgroup 서브시스템**
  - /sys/fs/cgroup 에 초기화된 cgroup 서브시스템이 존재
  - cgroup 서브시스템은 가상파일시스템(VFS)의 디렉토리로 표시되는 계층구조로 구성됨
  - process tree처럼 cgroup도 부모의 일부속성을 상속 받는다.
  - 여러 cgroup 계층 구조가 시스템에 동시에 존재 할 수 있음.
  - 각 계층은 단일 또는 자원 그룹을 나타냄
  - 그룹에 할당 된 태스크에는 해당 자원에 대한 제한이 적용됨
- **Resource metering and limiting**
  - memory
  - CPU
  - block I/O
  - network[1)]
- **Device node (/dev/*) access control**

1) With cooperation from iptables, kernel network facilities



서브시스템	설명
cpuset	개별 CPU 및 메모리 노드를 cgroup 작업에 할당
cpuacct	그룹별 CPU 자원 사용에 대한 통계/보고서 생성
cpu	CPU에 cgroup 작업 액세스를 제공하기 위한 스케줄러를 사용
memory	cgroup의 작업에 사용되는 메모리 제한 설정
devices	cgroup의 작업 단위로 장치에 대한 액세스를 허용하거나 거부
freezer	cgroup의 작업을 일시 중지하거나 다시 시작
net_cls	특정 cgroup 작업에서 발생하는 패킷을 식별하기 위한 태그를 지정
net_prio	cgroup 작업에서 생성되는 네트워크 트래픽의 우선순위 선정
blkio	블럭 장치와의 입출력 접근 제한을 설정
perf_event	perf 도구를 사용하여 cgroup을 모니터링 할 수 있도록 설정
hugetlb	HugeTLB 에 대한 제한 설정

CPU Controller

Completely Fair Scheduler

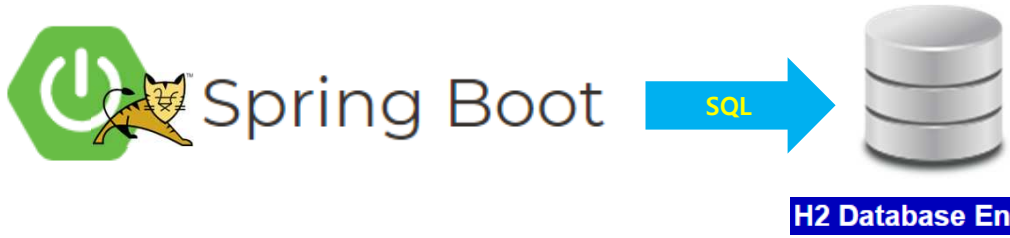
- The CPU controller is responsible for grouping tasks together that will be viewed by the scheduler as a single unit. The **CFS** *(Completely Fair Scheduler)* scheduler will first divide CPU time equally between all entities in the same level, and then proceed by doing the same in the next level. Basic use cases for that are described in the main cgroup documentation file, cgroups.txt.
- Users of this functionality should be aware that deep hierarchies will of course impose scheduler overhead, since the scheduler will have to take extra steps and look up additional data structures to make its final decision. Through the CPU controller, the scheduler is also able to cap the CPU utilization of a particular group. This is particularly useful in environments in which CPU is paid for by the hour, and one values predictability over performance.

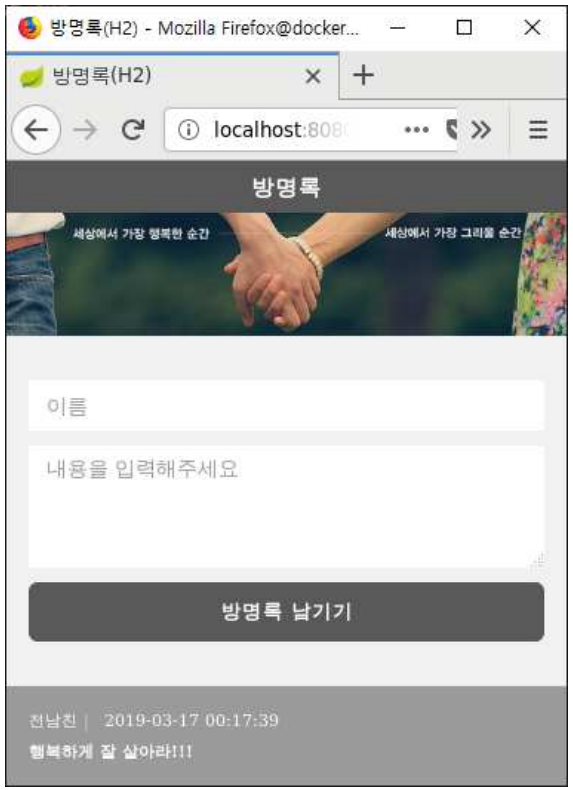
Files

파일	설명
cpu.shares	The weight of each group living in the same hierarchy, that translates into the amount of CPU it is expected to get. Upon cgroup creation, each group gets assigned a default of 1024. The percentage of CPU assigned to the cgroup is the value of shares divided by the sum of all shares in all cgroups in the same level.
cpu.cfs_period_us	The duration in microseconds of each scheduler period, for bandwidth decisions. This defaults to 100000us or 100ms. Larger periods will improve throughput at the expense of latency, since the scheduler will be able to sustain a cpu-bound workload for longer. The opposite of true for smaller periods. Note that this only affects non-RT tasks that are scheduled by the CFS scheduler.
cpu.cfs_quota_us	The maximum time in microseconds during each cfs_period_us in for the current group will be allowed to run. For instance, **if it is set to half of cpu_period_us, the cgroup will only be able to peak run for 50 % of the time.** One should note that this represents aggregate time over all CPUs in the system. Therefore, in order to allow full usage of two CPUs, for instance, one should set this value to twice the value of cfs_period_us.

CPU 사용률 제한 데모 (1/4)

- **어플리케이션 구조**
  - Spring Boot + H2 (Memory DB)






```java
@RequestMapping(method = RequestMethod.GET)
public String index(Model model) {
    // 01. 방명록 조회
    model.addAttribute("posts", postService.getAll());
    // 02. 의미없이 CPU 사용량 증가
    this.useCPU(1000);
    return "index";
}

private double useCPU(int loopCount) {
    double result = 0;
    for (int i = 1; i < loopCount; i++) {
        result = i * Math.random() / loopCount;
        for (int j = 1; j < loopCount; j++) {
            result = i * j * Math.random() / loopCount;
            for (int k = 1; k < loopCount; k++) {
                // CPU 만 소모
            }
        }
    }
    return result;
}
```

방명록 웹 어플리케이션의index 페이지는
의미 없이 CPU를 사용하는 코드가 구현되어 있음

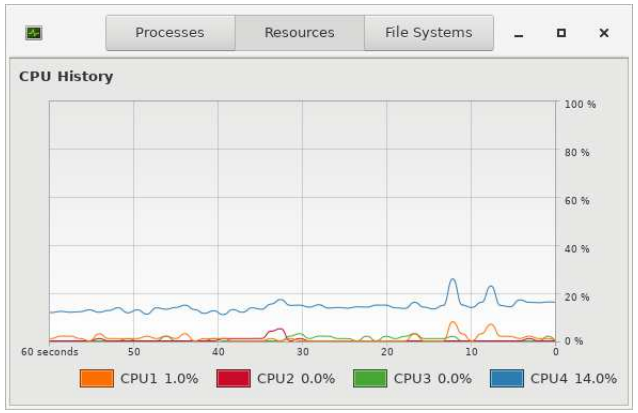
CPU 사용률 제한 데모 (2/4)

✓ cgroup 제어를 위한 패키지 설치

```
[root@dockeredu ~]# yum install -y libcgroup-tools
...
Installed:
  libcgroup-tools.x86_64 0:0.41-21.el7
```

✓ CPU 사용률을 모니터링할수 있도록 리눅스용 작업관리자 실행

```
[root@dockeredu ~]# gnome-system-monitor
```



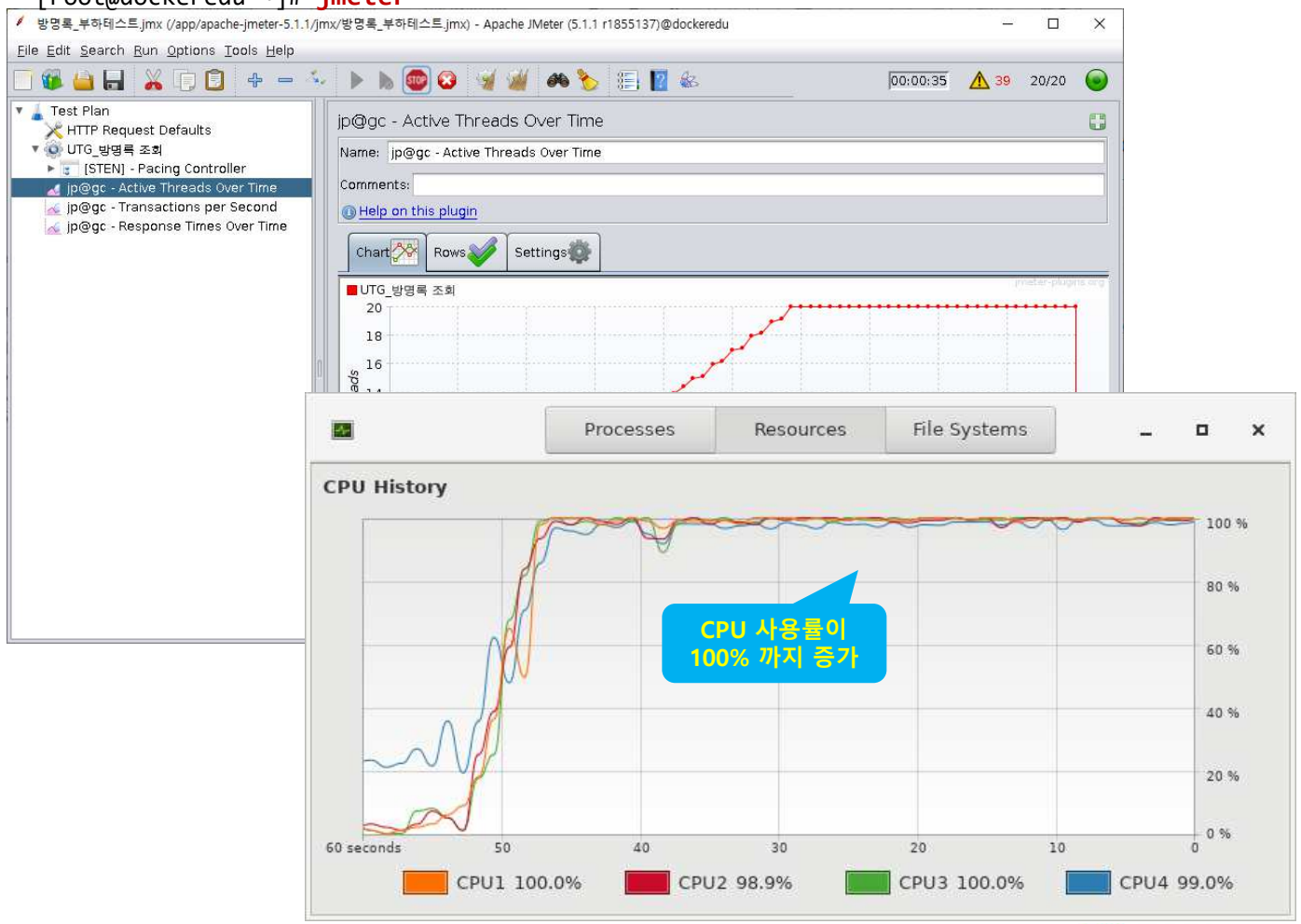
✓ 방명록 Spring Boot Java Application 실행

```
[root@dockeredu ~]# cd ~/guestbook/
[root@dockeredu guestbook]# java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)
...
```

✓ jmeter 실행

- File → Open : /app/jmeter/jmx/방명록_부하테스트.jmx
- index 페이지에 부하를 발생시키고 CPU 사용률이 100%까지 사용되는것을 모니터링

```
[root@dockeredu ~]# jmeter
```



CPU 사용률 제한 데모 (3/4)

✓ cgcreate 명령을 이용하여 cpu 제어를 위한 Control Group 생성하고 /sys/fs/cgroup/cpu/limit_cpu_50 폴더안의 파일 목록을 확인

```
[root@dockeredu ~]# cgcreate -g cpu:limit_cpu_50

[root@dockeredu ~]# ls /sys/fs/cgroup/cpu/limit_cpu_50
cgroup.clone_children  cgroup.procs  cpuacct.usage         cpu.cfs_period_us  cpu.rt_period_us   cpu.shares  notify_on_release
cgroup.event_control   cpuacct.stat  cpuacct.usage_percpu  cpu.cfs_quota_us   cpu.rt_runtime_us  cpu.stat    tasks
```

✓ 방명록 Java Application의 PID를 확인하고 limit_cpu_50 컨트롤그룹의 제어를 받도록 추가

```
[root@dockeredu ~]# ps -ef | grep java | grep guestbook
root      3654  3495 31 13:17 pts/0    00:05:16 java -jar guestbook_H2.jar

[root@dockeredu ~]# cgclassify -g cpu:limit_cpu_50 3654
[root@dockeredu ~]# cat /sys/fs/cgroup/cpu/limit_cpu_50/tasks
3654
3655
3656
3657
3658
3659
3660
3661
...
```

Linux에서는 Thread도 Process로 취급하기때문에 tasks 파일에 Java Process(Parent Process) ID를 추가하면 Thread까지 제어를 받지 못하므로 cgclassify 명령을 이용하여여 함

Java Thread

✓ cpu.cfs_period_us 파일의 내용을 확인

```
[root@dockeredu ~]# cat /sys/fs/cgroup/cpu/limit_cpu_50/cpu.cfs_period_us
100000
```

✓ OS에 설치된 CPU 코어수를 확인

```
[root@dockeredu ~]# cat /proc/cpuinfo | grep processor | wc -l
4
```

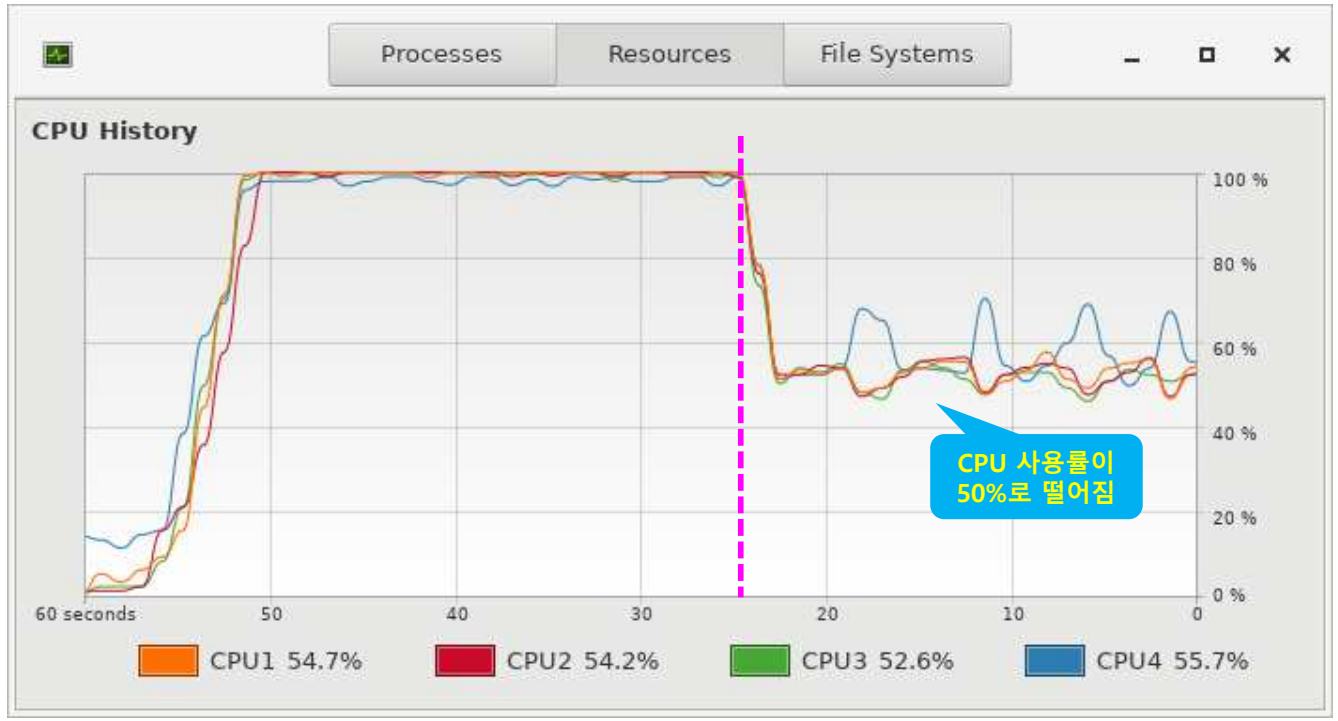
CPU 사용률 제한 데모 (4/4)

✓ cpu.cfs_quota_us 값을 조절하여 CPU 사용률을 50%만 사용하도록 설정

```
[root@dockeredu ~]# cgset -r cpu.cfs_quota_us=200000 limit_cpu_50
```

100000 * 4 * 0.5

- 0.5 → *CPU 사용률*
- 4 → *CPU 코어수*
- 100000 → *cpu.cfs_period_us 값*



도커 설치

사전 설정

▪ 방화벽 해제

- Docker는 HOST 머신에서 컨테이너로 패킷 필터링 및 포워딩을 통해 통신하기 때문에 HOST 머신의 방화벽을 Disable할 것을 권고

① 방화벽 서비스 종료

```
[root@docker ~]# systemctl stop firewalld
```

② 방화벽 상시가동 중지

```
[root@docker ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

③ 방화벽 서비스 상태 보기

```
[root@docker ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)

Apr 16 17:04:12 docker systemd[1]: Starting firewalld - dynamic firewall daemon...
Apr 16 17:04:13 docker systemd[1]: Started firewalld - dynamic firewall daemon.
Apr 16 19:27:41 docker systemd[1]: Stopping firewalld - dynamic firewall daemon...
Apr 16 19:27:51 docker systemd[1]: Stopped firewalld - dynamic firewall daemon.
```

▪ selinux 해제

- Linux의 어플리케이션 방화벽 기능을 하는 selinux 기능 해제 권고
- /etc/selinux/config 파일 편집
  - SELINUX=disabled

https://docs.docker.com/engine/install/centos/

docker 설치 - script를 이용한 최신버전 설치

- **docker.com에서 제공하는 쉘 스크립트를 이용하여 최신 버전을 간편하게 설치하는 방법**
  - 운영환경에서는 권장하지 않고, 안정화된 특정버전 설치 권장함
    - ① script를 이용한 최신버전 설치

```
[root@docker ~]# curl -sSL https://get.docker.com | sh
# Executing docker install script, commit: 36b78b2
+ sh -c 'yum install -y -q yum-utils'
Package yum-utils-1.1.31-46.el7_5.noarch already installed and latest version
+ sh -c 'yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo'
Loaded plugins: fastestmirror
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
+ '[' edge '!=' stable ']'
+ sh -c 'yum-config-manager --enable docker-ce-edge'
Loaded plugins: fastestmirror
+ sh -c 'yum makecache'
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.kaist.ac.kr
~~~
docker-ce-edge                                                                                | 2.9 kB  00:00:00
docker-ce-stable                                                                              | 2.9 kB  00:00:00
~~~
Metadata Cache Created
+ sh -c 'yum install -y -q docker-ce'
Delta RPMs reduced 346 k of updates to 203 k (41% saved)
~~~
setsebool:  SELinux is disabled.
If you would like to use Docker as a non-root user, you should now consider
adding your user to the "docker" group with something like:
  sudo usermod -aG docker your-user
Remember that you will have to log out and back in for this to take effect!

WARNING: Adding a user to the "docker" group will grant the ability to run  containers which can be used to obtain root privileges on the
docker host. Refer to https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface for more information.
```

docker 설치 - 패키지 매니저를 이용한 설치 (설치버전 선택 가능)

- **docker운영에 필요한 필수 패키지 설치**

  # yum install -y yum-utils device-mapper-persistent-data lvm2

- **dokcer repository 추가**

  # yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

- **docker 설치가능 버전 확인**

  # yum list docker-ce --showduplicates

```
Available Packages
docker-ce.x86_64            17.03.0.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.03.1.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.03.2.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.06.0.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.06.1.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.06.2.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.09.0.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.09.1.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.12.0.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            17.12.1.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            18.03.0.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            18.03.1.ce-1.el7.centos            docker-ce-stable
docker-ce.x86_64            18.06.0.ce-3.el7                   docker-ce-stable
docker-ce.x86_64            18.06.1.ce-3.el7                   docker-ce-stable
```

- **docker 최신버전 설치**

  # yum install -y docker-ce

- **(optional) 특정 docker 버전 설치**

  # yum install -y docker-ce-<VERSION_STRING>

  # yum install -y docker-ce-20.10.6

docker 서비스(도커 데몬) 정보 확인

- **docker 데몬을 시작하고 시스템 부팅 시에 자동 시작하도록 구성**

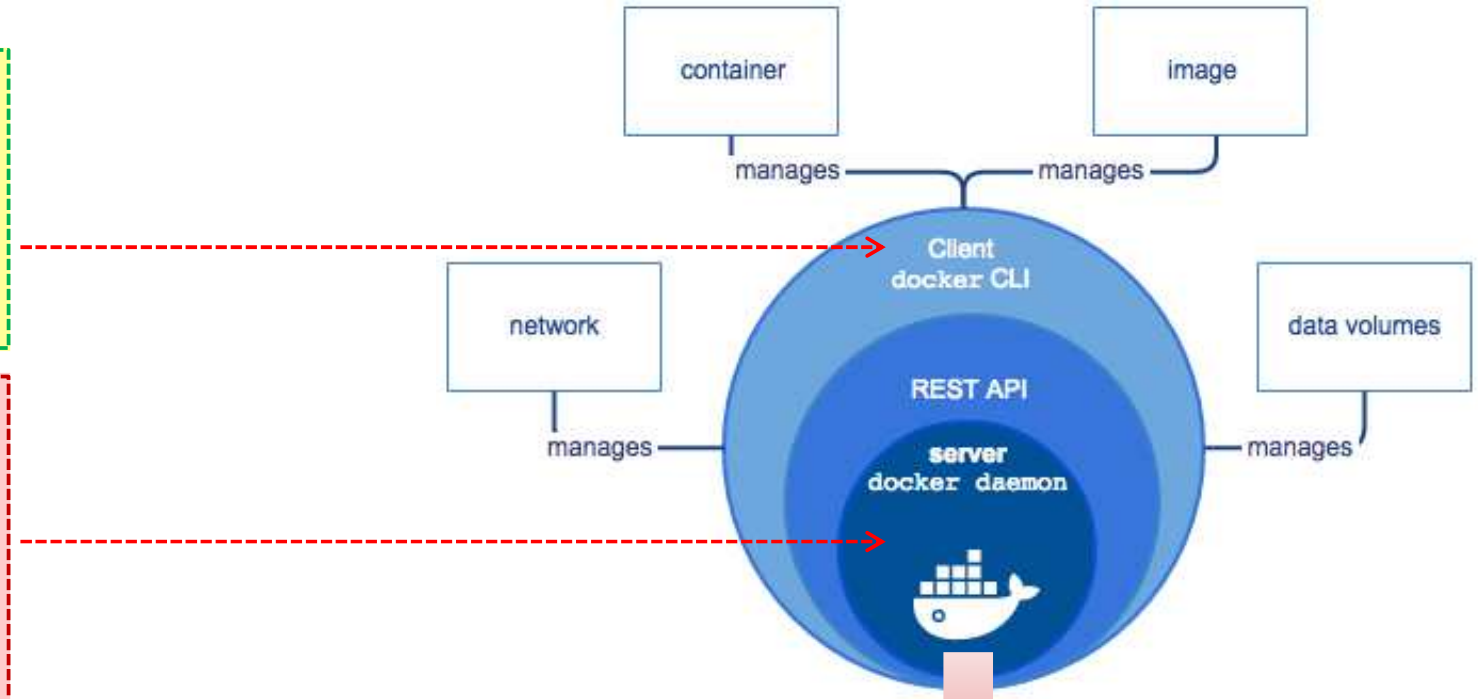
```
# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.

# systemctl start docker

# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2018-10-09 12:02:56 KST; 6s ago
     Docs: https://docs.docker.com
 Main PID: 1558 (dockerd)
    Tasks: 26
   Memory: 47.4M
   CGroup: /system.slice/docker.service
           ├─1558 /usr/bin/dockerd
           └─1567 docker-containerd --config /var/run/docker/containerd/containerd.toml

Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.322752568+09:00" level=info msg="pickfirstBalancer: Han...e=grpc
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.322910329+09:00" level=info msg="pickfirstBalancer: Han...e=grpc
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.322949662+09:00" level=info msg="Loading containers: start."
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.625551984+09:00" level=info msg="Default bridge (docker...dress"
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.832299971+09:00" level=info msg="Loading containers: done."
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.848758956+09:00" level=info msg="Docker daemon" commit=...6.1-ce
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.848875778+09:00" level=info msg="Daemon has completed i...ation"
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.864178106+09:00" level=warning msg="Could not register ...$PATH"
Oct 09 12:02:56 dockeredu dockerd[1558]: time="2018-10-09T12:02:56.871250443+09:00" level=info msg="API listen on /var/run....sock"
Oct 09 12:02:56 dockeredu systemd[1]: Started Docker Application Container Engine.
```

docker 버전 확인

- **docker는 Server(docker 데몬)와 Client(docker CLI)로 구성됨**

▶ docker 버전 확인

```
[root@dockeredu ~]# docker version
Client: Docker Engine - Community
 Version:           20.10.0
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        7287ab3
 Built:             Tue Dec  8 18:57:35 2020
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.0
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       eeddea2
  Built:            Tue Dec  8 18:56:55 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.3
  GitCommit:
269548fa27e0089a8b8278fc4fc781d7f65a93
 runc:
  Version:          1.0.0-rc92
  GitCommit:
ff819c7e9184c13b7c2607fe6c30ae19403a7a
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```



dockerd 프로세스 정보

Process Name	User	ID	Command Line	Control Group
systemd	root	1	/usr/lib/systemd/sys	
dockerd	root	15909	/usr/bin/dockerd -H	/system.slice/docker.service (blkio, cpuacct,cpu, devices, memory, pids, systemd)

```
/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

docker 구성 정보 확인

- **docker info 명령을 통해 docker 구성정보 확인**

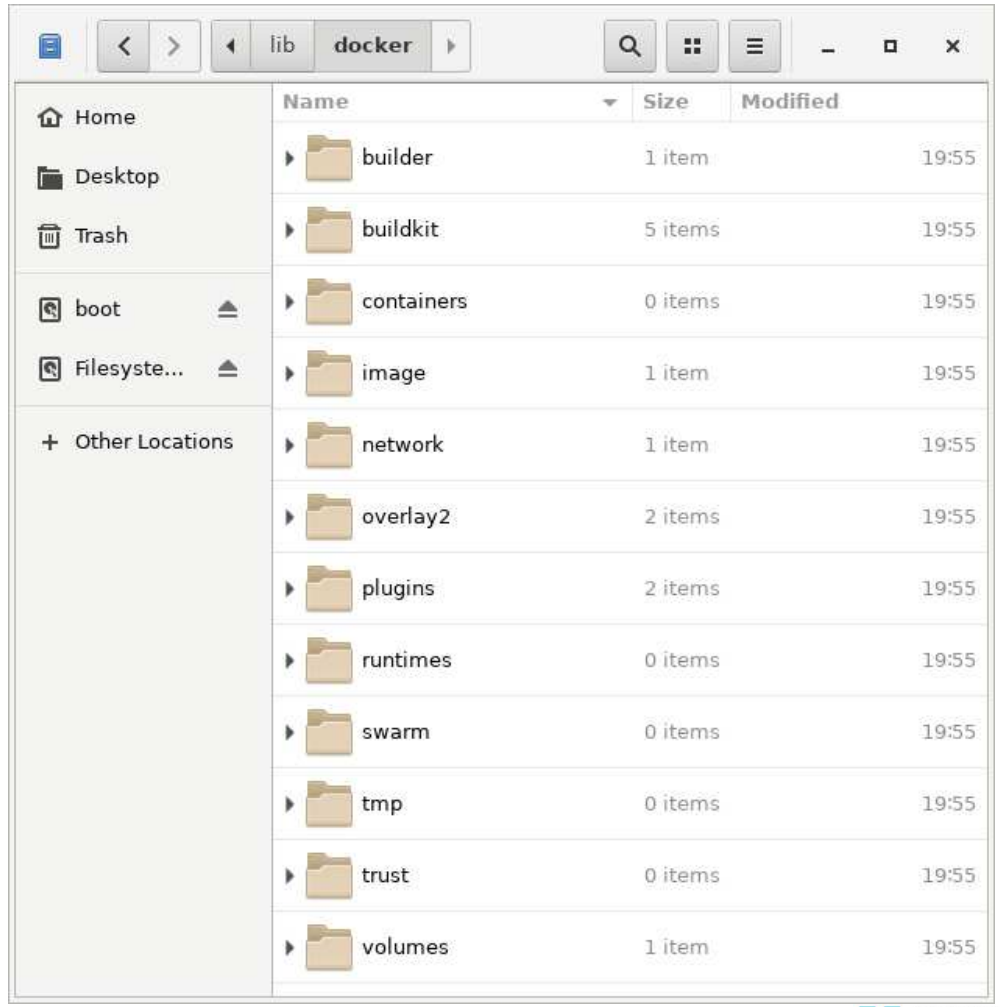
```
[root@dockeredu ~]# docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.4.2-docker)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 20.10.0
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 269548fa27e0089a8b8278fc4fc781d7f65a939b
 runc version: ff819c7e9184c13b7c2607fe6c30ae19403a7aff
 init version: de40ad0
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 3.10.0-957.el7.x86_64
 Operating System: CentOS Linux 7 (Core)
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: dockeredu
 ID: HK4W:P6TG:SNKL:G3MO:H7QS:EPM2:KVKV:7UCB:PMED:S252:7RXZ:3RJY
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
```

docker 디렉토리 계층 구조 확인

- **docker 데몬이 시작할 때 생성한 디렉토리 계층구조 확인**

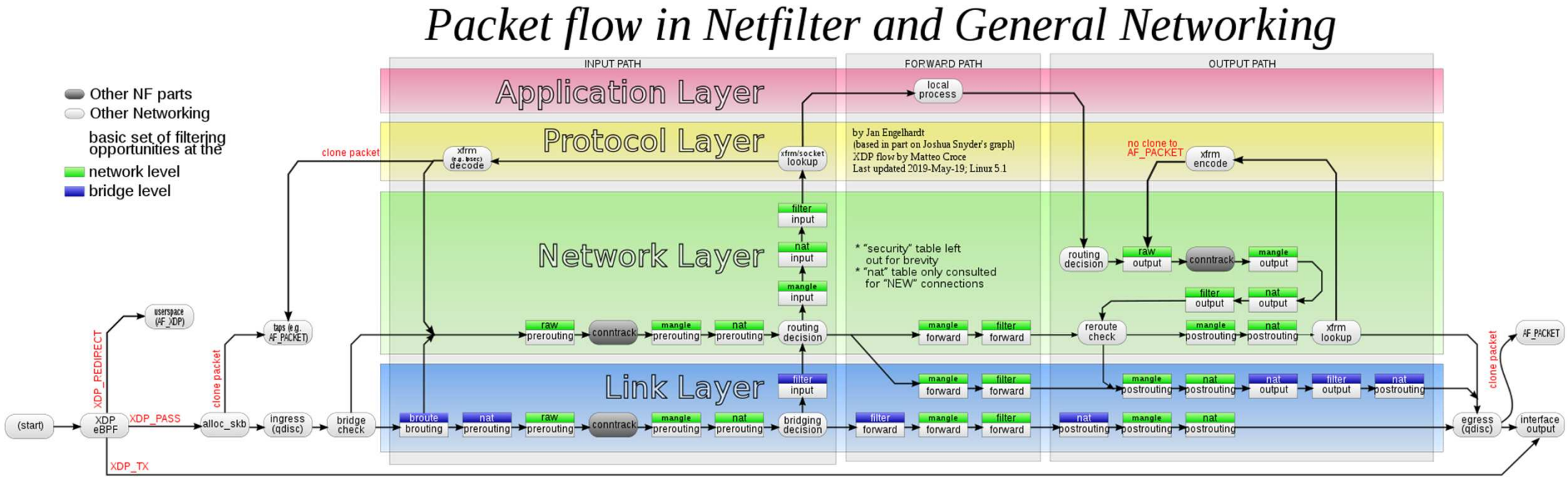
  # tree -L 2 /var/lib/docker

```
[root@dockeredu ~]# tree -L 2 /var/lib/docker
/var/lib/docker
├── builder
│   └── fscache.db
├── buildkit
│   ├── cache.db
│   ├── content
│   ├── executor
│   ├── metadata.db
│   └── snapshots.db
├── containerd
│   └── daemon
├── containers
├── image
│   └── overlay2
├── network
│   └── files
├── overlay2
│   ├── backingFsBlockDev
│   └── l
├── plugins
│   ├── storage
│   └── tmp
├── runtimes
├── swarm
├── tmp
├── trust
└── volumes
    └── metadata.db
```



참고) Linux Netfilter Subsytem

- **넷필터(Netfilter)**
  - 리눅스 커널 내부의 네트워크 관련 프레임워크
  - 다양한 네트워크 관련 연산을 핸들러 형태로 구현할 수 있도록 훅(hook)을 제공
    - https://ko.wikipedia.org/wiki/%EB%84%B7%ED%95%84%ED%84%B0
    - https://en.wikipedia.org/wiki/Netfilter



docker 설치후 네트워크 변화 (1/3)

- **docker0 NIC 추가**

도커 설치 전	도커 설치 후
[root@dockeredu ~]# **ifconfig** enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 inet 192.168.56.91 netmask 255.255.255.0 broadcast 192.168.56.255 inet6 fe80::a00:27ff:fe1b:adc4 prefixlen 64 scopeid 0x20<link> ether 08:00:27:1b:ad:c4 txqueuelen 1000 (Ethernet) RX packets 32369 bytes 47550079 (45.3 MiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 8256 bytes 728358 (711.2 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 inet 127.0.0.1 netmask 255.0.0.0 inet6 ::1 prefixlen 128 scopeid 0x10<host> loop txqueuelen 1000 (Local Loopback) RX packets 32 bytes 2592 (2.5 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 32 bytes 2592 (2.5 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0	[root@dockeredu ~]# **ifconfig** docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500 inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255 ether 02:42:56:1d:0f:78 txqueuelen 0 (Ethernet) RX packets 0 bytes 0 (0.0 B) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 0 bytes 0 (0.0 B) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 inet 192.168.56.91 netmask 255.255.255.0 broadcast 192.168.56.255 inet6 fe80::a00:27ff:fe1b:adc4 prefixlen 64 scopeid 0x20<link> ether 08:00:27:1b:ad:c4 txqueuelen 1000 (Ethernet) RX packets 32369 bytes 47550079 (45.3 MiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 8256 bytes 728358 (711.2 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 inet 127.0.0.1 netmask 255.0.0.0 inet6 ::1 prefixlen 128 scopeid 0x10<host> loop txqueuelen 1000 (Local Loopback) RX packets 32 bytes 2592 (2.5 KiB) RX errors 0 dropped 0 overruns 0 frame 0 TX packets 32 bytes 2592 (2.5 KiB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker 설치후 네트워크 변화 (2/3)

- **패킷 필터링 정책**

도커 설치 전

```
[root@dockeredu ~]# iptables -t filter -vnL
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source   destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source   destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source   destination
```

도커 설치 후

```
[root@dockeredu ~]# iptables -t filter -vnL
Chain INPUT (policy ACCEPT 101 packets, 6664 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target                  prot opt in      out       source     destination
    0     0 DOCKER-USER              all  --  *       *         0.0.0.0/0  0.0.0.0/0
    0     0 DOCKER-ISOLATION-STAGE-1 all  --  *       *         0.0.0.0/0  0.0.0.0/0
    0     0 ACCEPT                   all  --  *       docker0   0.0.0.0/0  0.0.0.0/0    ctstate RELATED,ESTABLISHED
    0     0 DOCKER                   all  --  *       docker0   0.0.0.0/0  0.0.0.0/0
    0     0 ACCEPT                   all  --  docker0 !docker0  0.0.0.0/0  0.0.0.0/0
    0     0 ACCEPT                   all  --  docker0 docker0   0.0.0.0/0  0.0.0.0/0

Chain OUTPUT (policy ACCEPT 60 packets, 6176 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
 pkts bytes target                  prot opt in      out       source               destination
    0     0 DOCKER-ISOLATION-STAGE-2  all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 RETURN                   all  --  *       *         0.0.0.0/0            0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
 pkts bytes target     prot opt in     out      source               destination
    0     0 DROP       all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    0     0 RETURN     all  --  *      *        0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

docker 설치후 네트워크 변화 (3/3)

- **패킷 NAT 정책**

도커 설치 전	도커 설치 후
[root@dockeredu ~]# **iptables -t nat -vnL** Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination Chain INPUT (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination	[root@dockeredu ~]# **iptables -t nat -vnL** Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination 0 0 DOCKER all -- * * 0.0.0.0/0 0.0.0.0/0 ADDRTYPE match dst-type LOCAL Chain INPUT (policy ACCEPT 0 packets, 0 bytes) pkts bytes target prot opt in out source destination Chain OUTPUT (policy ACCEPT 4 packets, 304 bytes) pkts bytes target prot opt in out source destination 0 0 DOCKER all -- * * 0.0.0.0/0 !127.0.0.0/8 ADDRTYPE match dst-type LOCAL Chain POSTROUTING (policy ACCEPT 4 packets, 304 bytes) pkts bytes target prot opt in out source destination 0 0 MASQUERADE all -- * !docker0 172.17.0.0/16 0.0.0.0/0 Chain DOCKER (2 references) pkts bytes target prot opt in out source destination 0 0 RETURN all -- docker0 * 0.0.0.0/0 0.0.0.0/0

도커 명령어 자동 완성

- **Docker command Auto-Completion 기능 사용하기**

```
[root@registry ~]# yum install -y bash-completion
Running transaction
  Installing : 1:bash-completion-2.1-6.el7.noarch                                1/1
  Verifying  : 1:bash-completion-2.1-6.el7.noarch                                1/1

Installed:
  bash-completion.noarch 1:2.1-6.el7

Complete!
```

(optional) Docker 20 버전이상에서는 아래 명령 수행 필요 없음

```
[root@registry ~]# curl https://raw.githubusercontent.com/docker/docker-ce/master/components/cli/contrib/completion/bash/docker -o
/etc/bash_completion.d/docker.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  113k  100  113k    0     0   160k      0 --:--:-- --:--:-- --:--:--  160k
```

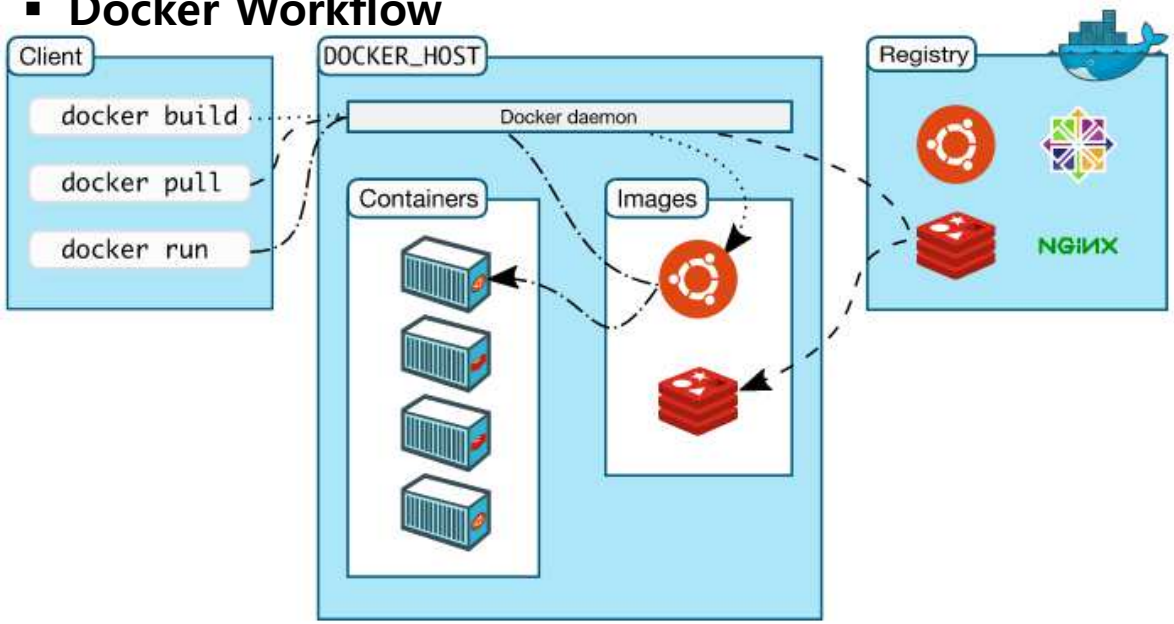
로그 아웃 후 재 접속

컨테이너로 작업하기

도커 용어 정리

- **https://docs.docker.com/glossary/**

- **Docker Workflow**



용어	설명
image	Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it **never changes**.
container	A container is a **runtime instance of a docker image**. A Docker container consists of ▪ A Docker image ▪ An execution environment ▪ A standard set of instructions The concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.
registry	A Registry is a **hosted service containing repositories of images** which responds to the Registry API. The default registry can be accessed using a browser at Docker Hub or using the docker search command.
repository	A repository is a **set of Docker images**. A repository can be shared by pushing it to a registry server. The different images in the repository can be labeled using tags.
tag	A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other.

첫번째 컨테이너 실행 (1/7)

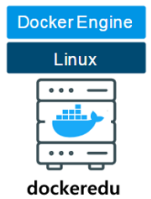
- **docker container run -it centos:7.6.1810**
  - HOST 머신의 CPU 사용률은?

https://hub.docker.com

image : centos
tag : 7.6.1810


```
[root@dockeredu ~]# docker container run -it centos:7.6.1810
Unable to find image 'centos:7.6.1810' locally
7.6.1810: Pulling from library/centos
ac9208207ada: Pulling fs layer
ac9208207ada: Downloading     72MB/75.16MB
ac9208207ada: Verifying Checksum
ac9208207ada: Download complete
ac9208207ada: Extracting  75.16MB/75.16MB
ac9208207ada: Pull complete
Digest: sha256:6ae4cddb2b37f889afd576a17a5286b311dcb
Status: Downloaded newer image for centos:7.6.1810
[root@667f761902d7 /]#
[root@667f761902d7 /]# hostname
667f761902d7
```

1 2 3 4



- **컨테이너 실행 절차**
  - ① 로컬에 centos:7.6.1810 이미지가 없음
  - ② https://hub.docker.com에서 centos 이미지를 로컬에 다운로드
  - ③ centos:7.6.1810 이미지가 정상 다운로드 됨
  - ④ container 안에서 **1번 프로세스**로 /bin/bash 프로세스가 실행됨

첫번째 컨테이너 실행 (2/7)

- 프로세스 격리

HOST

```
[root@dockeredu ~]# pstree
systemd─┬─NetworkManager───2*[{NetworkManager}]
        ├─agetty
        ├─auditd───{auditd}
        ├─chronyd
        ├─crond
        ├─2*[dbus-daemon]
        ├─dbus-launch
        ├─dconf-service───2*[{dconf-service}]
        ├─dockerd─┬─docker-containe─┬─docker-containe─┬─bash
        │         │                 │                 └─9*[{docker-containe}]
        │         │                 └─15*[{docker-containe}]
        │         └─15*[{dockerd}]
        ├─irqbalance
        ├─lvmeta
        ├─master
        ├─polkit
        ├─rsyslo
        ├─sshd─┬
        │      ├─sshd───sftp-server
        │      └─sshd───bash───docker───13*[{docker}]
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-udevd
        ├─tuned───4*[{tuned}]
        ├─xdg-desktop-por───3*[{xdg-desktop-por}]
        ├─xdg-document-po───5*[{xdg-document-po}]
        └─xdg-permission-───2*[{xdg-permission-}]
```

Process Name	User	Command Line	Control Group
dockerd	root	/usr/bin/dockerd	/system.slice/docker.service (blkio, cpuacct,cpu, devices, memory, pids, systemd)
docker-containerd	root	docker-containerd	/system.slice/docker.service (blkio, cpuacct,cpu, devices, memory, pids, systemd)
docker-containerd-shim	root	docker-containerd	/system.slice/docker.service (blkio, cpuacct,cpu, devices, memory, pids, systemd)
bash	root	/bin/bash	/docker/667f761902d747ed03c5e1ba7176a715cea2709681beb14f48438af4c45a5f9a (blkio,

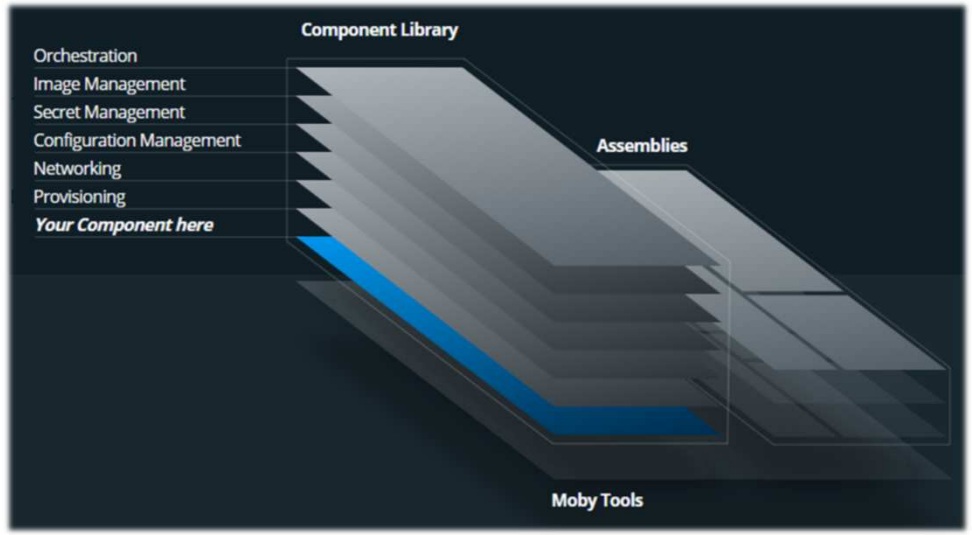
```
docker-containerd-shim -namespace moby -workdir /var/lib/docker/containerd/
daemon/io.containerd.runtime.v1.linux/moby/
667f761902d747ed03c5e1ba7176a715cea2709681beb14f48438af4c45a5f9a -
address /var/run/docker/containerd/docker-containerd.sock -containerd-binary /usr/
bin/docker-containerd -runtime-root /var/run/docker/runtime-runc
```

CONTAINER

```
[root@667f761902d7 /]# yum install -y psmisc

[root@667f761902d7 /]# pstree
bash---pstree

[root@667f761902d7 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Apr16 pts/0    00:00:00 /bin/bash
root        79     1  0 00:14 pts/0    00:00:00 ps -ef
```

첫번째 컨테이너 실행 (3/7)

- **Moby Project**
  - https://mobyproject.org
  - 컨테이너 기술을 베이스로 한 컴포넌트를 조합하여 시스템을 구축할 수 있다는
  - 다양한 컴포넌트가 제공되며 컨테이너 런타임, 오케스트레이션, 네트워킹 보안, 행 가능한 컨테이너 환경을 구축할 수 있습니다.



moby project

swarmKIT infraKIT LinuxKIT containerd

runc VPNKit Registry LibNetwork

Notary DataKit Compose HyperKit

- Moby Project에서 개발이 진행되고 컴포넌트
  - `containerd` : 컨테이너 런타임
  - `LinuxKit` : `containerd`를 작동시키기 위한 `Linux` 환경
  - `InfraKit` : 인프라를 추상화하여 자동화하는 컴포넌트
- Docker는 이 Moby Project에서 개발이 진행되고 있는 툴 중 하나입니다

첫번째 컨테이너 실행 (4/7)

- 네트워크 격리

HOST

```
[root@dockeredu ~]# ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:56ff:fe1d:f78  prefixlen 64  scopeid 0x20<link>
        ether 02:42:56:1d:0f:78  txqueuelen 0  (Ethernet)
        RX packets 1989  bytes 109090 (106.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2053  bytes 10888112 (10.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.91 netmask 255.255.255.0 broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe1b:adc4  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:1b:ad:c4  txqueuelen 1000  (Ethernet)
        RX packets 126559  bytes 140523581 (134.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 58457  bytes 20709646 (19.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 57770  bytes 90182852 (86.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 57770  bytes 90182852 (86.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth6436f30: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::f073:9bff:fe3c:f2dd  prefixlen 64  scopeid 0x20<link>
        ether f2:73:9b:3c:f2:dd  txqueuelen 0  (Ethernet)
        RX packets 1989  bytes 136936 (133.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2061  bytes 10888768 (10.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
[root@dockeredu ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242b5366f14       no              vetha1d505f
```

CONTAINER

```
[root@667f761902d7 /]# yum install -y net-tools
[root@667f761902d7 /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 2061  bytes 10888768 (10.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1989  bytes 136936 (133.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@667f761902d7 /]# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      667f761902d7
[root@667f761902d7 /]#
[root@667f761902d7 /]# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.56.1
```

첫번째 컨테이너 실행 (5/7)

▪ 파일 시스템 격리 (1/3)

▸ / 파일 시스템 목록 비교

HOST

```
[root@dockeredu ~]# ll -h /
total 24K
drwxr-xr-x.   6 root root  154 Apr 16 21:17 app
lrwxrwxrwx.   1 root root    7 Apr 16 20:40 bin -> usr/bin
dr-xr-xr-x.   5 root root 4.0K Apr 16 20:44 boot
drwxr-xr-x   19 root root 3.0K Apr 16 21:40 dev
drwxr-xr-x.  97 root root 8.0K Apr 16 21:45 etc
drwxr-xr-x.   2 root root    6 Apr 11  2018 home
lrwxrwxrwx.   1 root root    7 Apr 16 20:40 lib -> usr/lib
lrwxrwxrwx.   1 root root    9 Apr 16 20:40 lib64 -> usr/lib64
drwxr-xr-x.   2 root root    6 Apr 11  2018 media
drwxr-xr-x.   2 root root    6 Apr 11  2018 mnt
drwxr-xr-x.   2 root root    6 Apr 16 21:07 opt
dr-xr-xr-x  141 root root    0 Apr 16 21:40 proc
dr-xr-x---.  14 root root 4.0K Apr 16 21:42 root
drwxr-xr-x   27 root root  820 Apr 17 09:53 run
lrwxrwxrwx.   1 root root    8 Apr 16 20:40 sbin -> usr/sbin
drwxr-xr-x.   2 root root    6 Apr 11  2018 srv
dr-xr-xr-x   13 root root    0 Apr 17 09:54 sys
drwxrwxrwt.  14 root root 4.0K Apr 17 03:07 tmp
drwxr-xr-x.  13 root root  155 Apr 16 20:40 usr
drwxr-xr-x.  19 root root  267 Apr 16 20:56 var
```

CONTAINER

```
[root@667f761902d7 /]# ll -h /
total 12K
-rw-r--r--   1 root root 12K Dec  4 14:39 anaconda-post.log
lrwxrwxrwx   1 root root   7 Dec  4 14:38 bin -> usr/bin
drwxr-xr-x   5 root root 360 Apr 16 12:58 dev
drwxr-xr-x   1 root root  66 Apr 16 12:58 etc
drwxr-xr-x   2 root root   6 Apr 11  2018 home
lrwxrwxrwx   1 root root   7 Dec  4 14:38 lib -> usr/lib
lrwxrwxrwx   1 root root   9 Dec  4 14:38 lib64 -> usr/lib64
drwxr-xr-x   2 root root   6 Apr 11  2018 media
drwxr-xr-x   2 root root   6 Apr 11  2018 mnt
drwxr-xr-x   2 root root   6 Apr 11  2018 opt
dr-xr-xr-x 141 root root   0 Apr 16 12:58 proc
dr-xr-x---   2 root root 114 Dec  4 14:39 root
drwxr-xr-x   1 root root  19 Apr 17 00:54 run
lrwxrwxrwx   1 root root   8 Dec  4 14:38 sbin -> usr/sbin
drwxr-xr-x   2 root root   6 Apr 11  2018 srv
dr-xr-xr-x  13 root root   0 Apr 17 00:54 sys
drwxrwxrwt   1 root root   6 Apr 17 00:54 tmp
drwxr-xr-x   1 root root  40 Dec  4 14:38 usr
drwxr-xr-x   1 root root  52 Dec  4 14:38 var
```

첫번째 컨테이너 실행 (6/7)

- **파일 시스템 격리 (2/3)**
  - / 파일 시스템 용량 비교

HOST	CONTAINER
[root@dockeredu ~]# **du -hs /* 2> /dev/null** 880M /app 0 /bin 94M /boot 0 /dev 35M /etc 0 /home 0 /lib 0 /lib64 0 /media 0 /mnt 0 /opt 0 /proc 118M /root 8.7M /run 0 /sbin 0 /srv 0 /sys 1.1M /tmp 2.1G /usr 707M /var	[root@667f761902d7 /]# **du -hs /* 2> /dev/null** 12K /anaconda-post.log 0 /bin 0 /dev 2.3M /etc 0 /home 0 /lib 0 /lib64 0 /media 0 /mnt 0 /opt 0 /proc 24K /root 0 /run 0 /sbin 0 /srv 0 /sys 4.0K /tmp 184M /usr 83M /var

첫번째 컨테이너 실행 (7/7)

- **파일 시스템 격리 (3/3)**
  - 파일시스템 용량 및 mount 경로 확인

HOST

```
[root@dockeredu ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        96G  3.6G   92G   4% /
devtmpfs        2.0G     0  2.0G   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           2.0G  8.7M  2.0G   1% /run
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/sda1      1014M  127M  888M  13% /boot
tmpfs           396M  8.0K  396M   1% /run/user/0
overlay          96G  3.6G   92G   4% /var/lib/docker/overlay2/2673a96456cedf51089c2e42294f951e66b4e21c41cd7466b9d9aac8f15ec765/merged
shm              64M     0   64M   0% /var/lib/docker/containers/667f761902d747ed03c5e1ba7176a715cea2709681beb14f48438af4c45a5f9a/mounts/shm
```

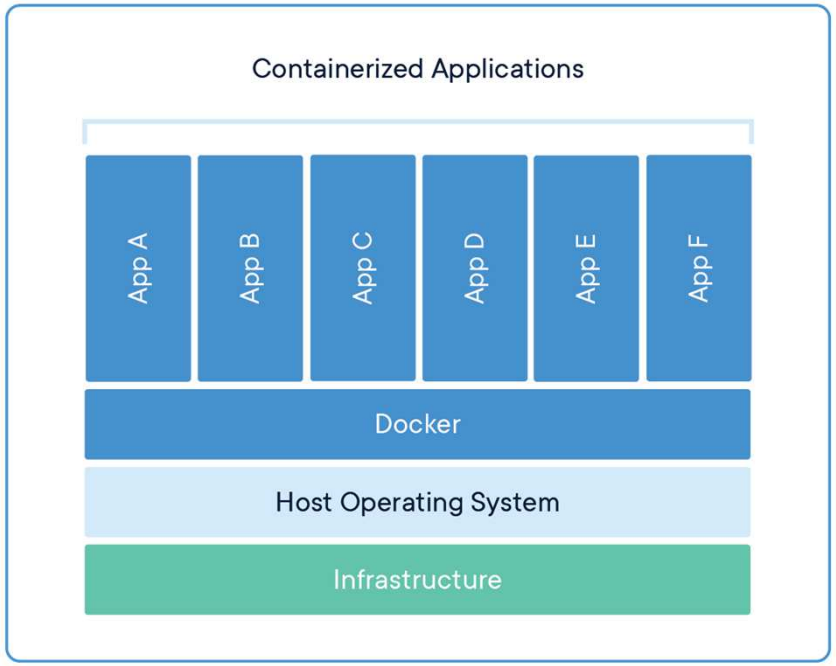
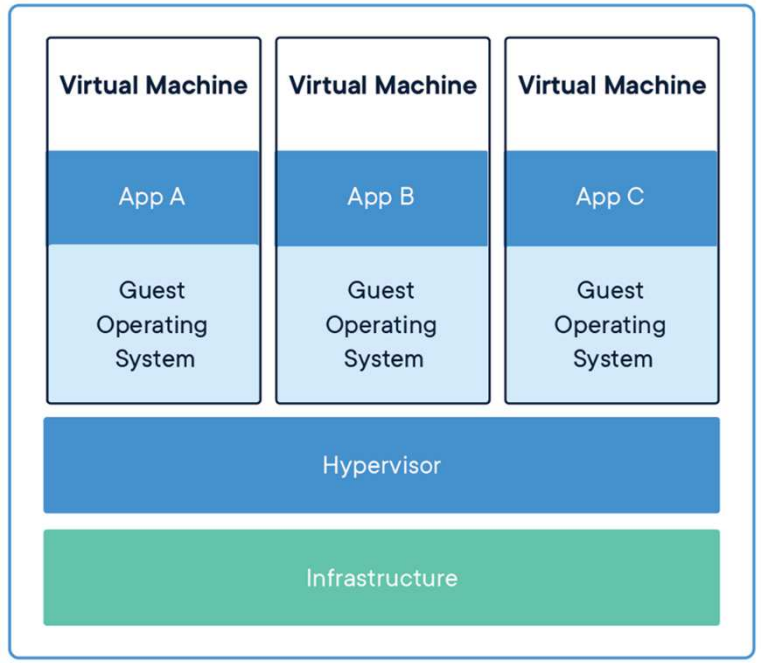
```
[root@dockeredu ~]# ll /var/lib/docker/overlay2/2673a96456cedf51089c2e42294f951e66b4e21c41cd7466b9d9aac8f15ec765/merged
total 12
-rw-r--r-- 1 root root 12053 Dec  4 23:39 anaconda-post.log
lrwxrwxrwx 1 root root     7 Dec  4 23:38 bin -> usr/bin
drwxr-xr-x 1 root root    43 Apr 16 21:58 dev
drwxr-xr-x 1 root root    66 Apr 16 21:58 etc
drwxr-xr-x 2 root root     6 Apr 11  2018 home
lrwxrwxrwx 1 root root     7 Dec  4 23:38 lib -> usr/lib
lrwxrwxrwx 1 root root     9 Dec  4 23:38 lib64 -> usr/lib64
drwxr-xr-x 2 root root     6 Apr 11  2018 media
drwxr-xr-x 2 root root     6 Apr 11  2018 mnt
drwxr-xr-x 2 root root     6 Apr 11  2018 opt
drwxr-xr-x 2 root root     6 Dec  4 23:38 proc
dr-xr-x--- 2 root root   114 Dec  4 23:39 root
drwxr-xr-x 1 root root    19 Apr 17 09:54 run
lrwxrwxrwx 1 root root     8 Dec  4 23:38 sbin -> usr/sbin
drwxr-xr-x 2 root root     6 Apr 11  2018 srv
drwxr-xr-x 2 root root     6 Dec  4 23:38 sys
drwxrwxrwt 1 root root     6 Apr 17 09:54 tmp
drwxr-xr-x 1 root root    40 Dec  4 23:38 usr
drwxr-xr-x 1 root root    52 Dec  4 23:38 var
```

CONTAINER

```
[root@667f761902d7 /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          96G  3.6G   92G   4% /
tmpfs            64M     0   64M   0% /dev
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/sda3        96G  3.6G   92G   4% /etc/hosts
shm              64M     0   64M   0% /dev/shm
tmpfs           2.0G     0  2.0G   0% /proc/acpi
tmpfs           2.0G     0  2.0G   0% /proc/scsi
tmpfs           2.0G     0  2.0G   0% /sys/firmware
```

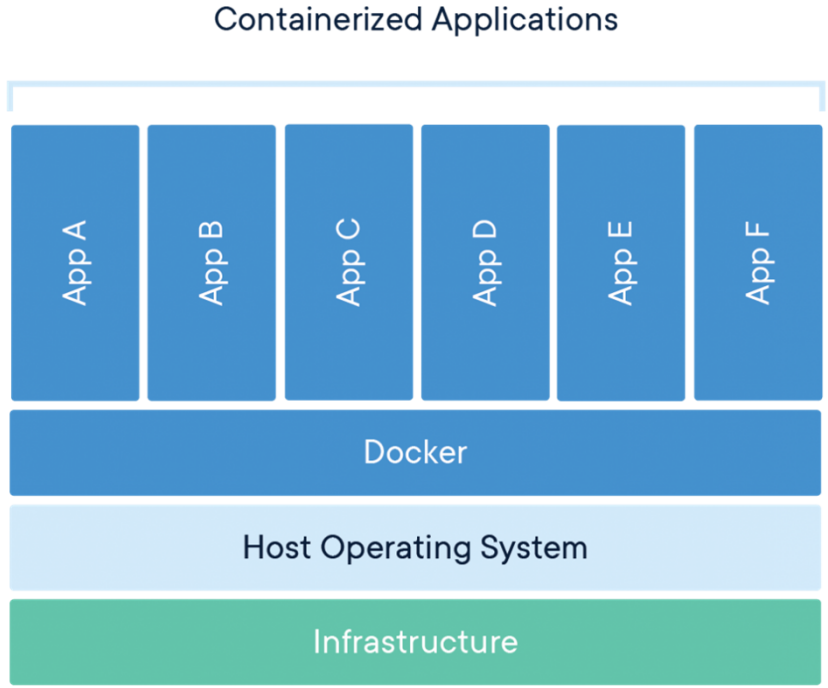
Container -vs- VM

- **컨테이너는 애플리케이션과 그 실행환경을 모두 포함한 소프트웨어 패키지로 Host OS에 상관없이 배포가 쉽고, 유연하며 Host OS와 애플리케이션을 분리**

CONTAINER	Virtual Machine
	
✓ 낮은 비용 ✓ 빠른 어플리케이션 개발 ✓ 보안 간편성 ✓ 새로운 IT에 적용가능(하이브리드 클라우드) ✓ 다른 버전도 같은 이미지에서 운영가능	✓ Booting 시 수분이 걸림(부팅동안, 많은 취약점과 장애발생 소지) ▪ http://www.makelinux.net/kernel_map/ ✓ 패치하고, 버전관리하는데 많은 노동력 집중 ✓ /hypervisor와 guestOS를 통시에 관리하고, 보안취약점을 검증하여야 함 ✓ 간단한 OS 프로세스를 위해서도 새로운 가상머신이 필요함. (비실용적) ✓ 리소스사용량이 많음.

What is a Container?

- **A standardized unit of software**
  - Package Software into Standardized Units for Development, Shipment and Deployment



A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on **Docker Engine**. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Docker containers that run on Docker Engine:

- **Standard**: Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight**: Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure**: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

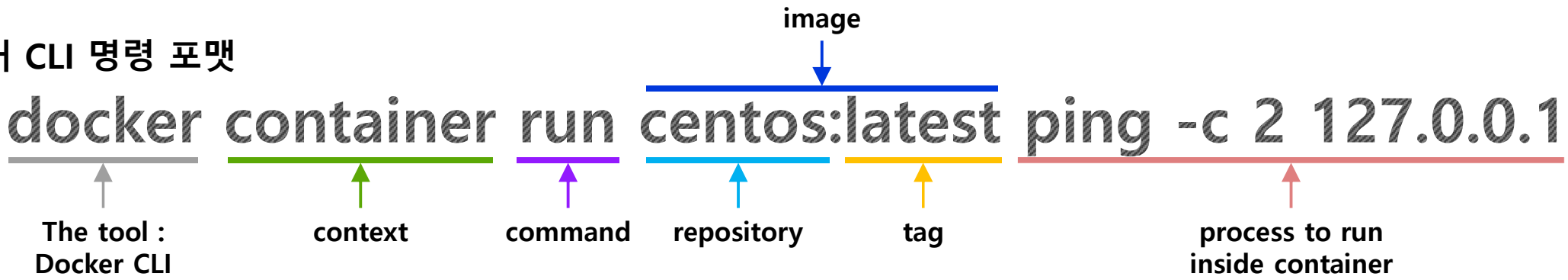
- **클라우드 업체별 컨테이너 정의**
  - https://aws.amazon.com/ko/containers/
  - https://cloud.google.com/kubernetes-engine/?hl=ko
  - https://www.redhat.com/ko/technologies/cloud-computing/openshift
  - https://www.ibm.com/kr-ko/cloud/container-service

2번째 컨테이너 실행

```
[root@dockeredu ~]# docker container run centos:latest ping -c 2 127.0.0.1
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
8ba884070f61: Pulling fs layer
8ba884070f61: Downloading    527kB/75.4MB
8ba884070f61: Verifying Checksum
8ba884070f61: Download complete
8ba884070f61: Extracting  557.1kB/75.4MB
8ba884070f61: Pull complete
Digest: sha256:8d487d68857f5bc9595793279b33d082b03713341ddec91054382641d14db861
Status: Downloaded newer image for centos:latest
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.053 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.136 ms

--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 4012ms
rtt min/avg/max/mdev = 0.049/0.075/0.136/0.033 ms
[root@dockeredu ~]#
```

ping 명령 실행 후 컨테이너가 바로 정지되는 이유는?

- 도커 CLI 명령 포맷



docker command 명령어 도움말

- **docker --help**

```
[root@dockeredu ~]# docker --help

Usage:  docker [OPTIONS] COMMAND

Management Commands:
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error strea
~~~
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed t
```

최신 스타일

context별

예전 스타일

하위호환용

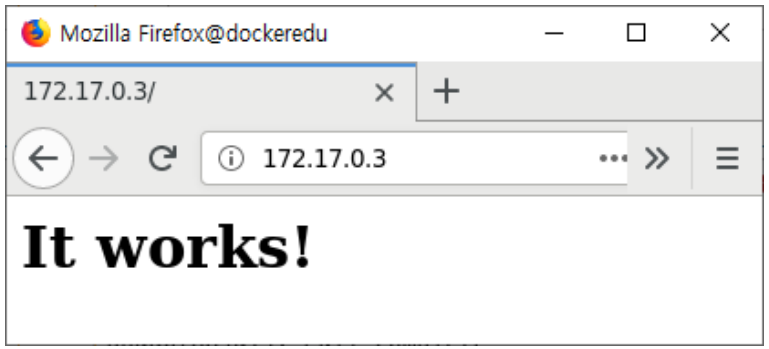
```
[root@dockeredu ~]# docker container --help

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
~~~
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
~~~
```

```
[root@dockeredu ~]# docker container run --help

Usage:  docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

Options:
      --add-host list                  Add a custom host-to-IP mapping (host:ip)
  -a, --attach list                    Attach to STDIN, STDOUT or STDERR
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
      --cgroup-parent string           Optional parent cgroup for the container
      --cidfile string                 Write the container ID to the file
      --cpu-period int                 Limit CPU CFS (Completely Fair Scheduler) period
      --cpu-quota int                  Limit CPU CFS (Completely Fair Scheduler) quota
      --cpu-rt-period int              Limit CPU real-time period in microseconds
      --cpu-rt-runtime int             Limit CPU real-time runtime in microseconds
  -c, --cpu-shares int                 CPU shares (relative weight)
~~~
```

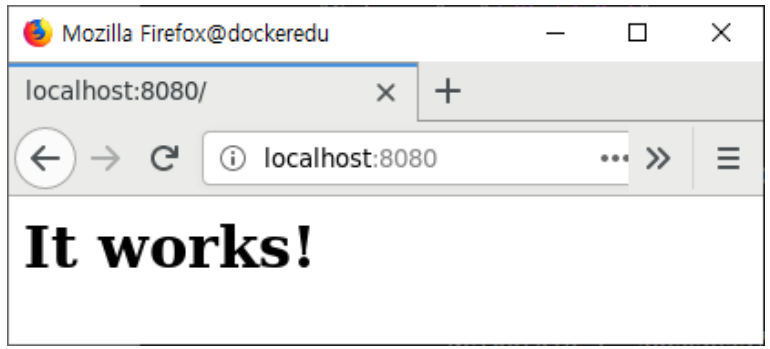
3번째 컨테이너 실행

```
[root@dockeredu ~]# docker run -d -p 8080:80 httpd:2.4
Unable to find image 'httpd:2.4' locally
2.4: Pulling from library/httpd
27833a3ba0a5: Pull complete
7df2f4a2bf95: Pull complete
bbda6f884d14: Pull complete
4d3dcf503f89: Pull complete
b2f11da8a23e: Pull complete
Digest: sha256:b4096b744d92d1825a36b3ace61ef4caa2ba57d0307b985cace4621139c285f7
Status: Downloaded newer image for httpd:2.4
1032437f8ed97e7de4517aa9640987166757acaf274c45aff27939e500533824

[root@dockeredu ~]# firefox
```

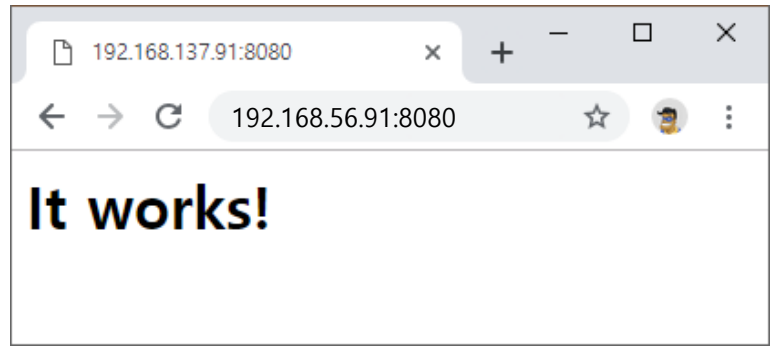
- 웹서버 실행 후 웹페이지 접속



컨테이너 IP로 접속



HOST에서 localhost로 접속



외부에서 접속

컨테이너 네트워크 과정


```
[root@dockeredu ~]# iptables -t filter -vnL
Chain INPUT (policy ACCEPT 216K packets, 11M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target                  prot opt in      out       source               destination
 4073   11M DOCKER-USER              all  --  *       *         0.0.0.0/0            0.0.0.0/0
 4073   11M DOCKER-ISOLATION-STAGE-1  all  --  *       *         0.0.0.0/0            0.0.0.0/0
 2062   11M ACCEPT                   all  --  *       docker0   0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
    2   104 DOCKER                   all  --  *       docker0   0.0.0.0/0            0.0.0.0/0
 2009  111K ACCEPT                   all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT                   all  --  docker0 docker0   0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 156K packets, 471M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER (1 references)
 pkts bytes target     prot opt in       out      source               destination
    2   104 ACCEPT     tcp  --  !docker0 docker0  0.0.0.0/0            172.17.0.3           tcp dpt:80

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
 pkts bytes target                  prot opt in      out       source               destination
 2009  111K DOCKER-ISOLATION-STAGE-2  all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
 4073   11M RETURN                   all  --  *       *         0.0.0.0/0            0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
 pkts bytes target     prot opt in     out      source               destination
    0     0 DROP       all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
 2009  111K RETURN     all  --  *      *        0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
 4073   11M RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

[root@dockeredu ~]# iptables -t nat -vnL
Chain PREROUTING (policy ACCEPT 1 packets, 229 bytes)
 pkts bytes target     prot opt in     out     source               destination
   12   536 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 1 packets, 229 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 35 packets, 2185 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 37 packets, 2289 bytes)
 pkts bytes target     prot opt in     out        source               destination
   66  4079 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0
    0     0 MASQUERADE  tcp  --  *      *         172.17.0.3           172.17.0.3           tcp dpt:80

Chain DOCKER (2 references)
 pkts bytes target     prot opt in       out     source               destination
    0     0 RETURN     all  --  docker0  *       0.0.0.0/0            0.0.0.0/0
    2   104 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0            tcp dpt:8080 to:172.17.0.3:80
```

docker container ls

▪ 컨테이너 목록 표시

▸ Usage: docker container ls [OPTIONS]

옵션		설명
-a	--all	✓ Show all containers (default shows just running)
-f	--filter filter	✓ Filter output based on conditions provided ✓ docker container ls -a -f status=exited
	--no-trunc	✓ Don't truncate output
-q	--quiet	✓ Only display numeric IDs ✓ ID만 조회후 일괄 삭제 또는 일괄 실행시 유용함 • docker container rm -f $(docker container ls -aq)

```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE            COMMAND                  CREATED          STATUS                      PORTS                    NAMES
1032437f8ed9   httpd:2.4        "httpd-foreground"       25 minutes ago   Up 25 minutes               0.0.0.0:8080->80/tcp     mystifying_northcutt
218a572be281   centos           "ping -c 5 127.0.0.1"    4 hours ago      Exited (0) 4 hours ago                               dazzling_hugle
667f761902d7   centos:7.6.1810  "/bin/bash"              17 hours ago     Up 17 hours                                          goofy_varahamihira
```

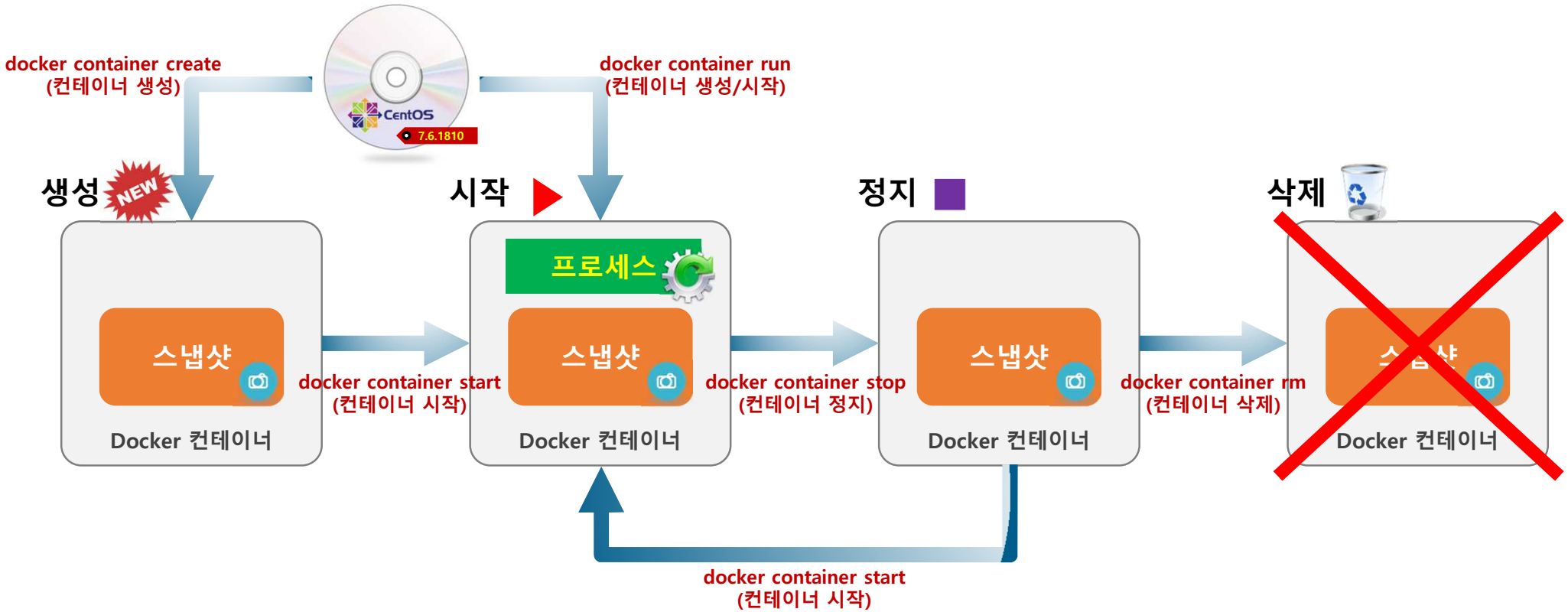
▸ 명령 결과 설명

항목	설명
CONTAINER ID	컨테이너 ID
IMAGE	컨테이너의 바탕이 되는 이미지
COMMAND	컨테이너 안에서 실행되고 있는 명령 → 1번 프로세스
CREATED	컨테이너 생성 후 경과 시간

항목	설명
STATUS	컨테이너의 상태 (running | exited | paused | restarting)
PORTS	할당된 포트
NAMES	컨테이너 이름

컨테이너 라이프 사이클 (1/6)

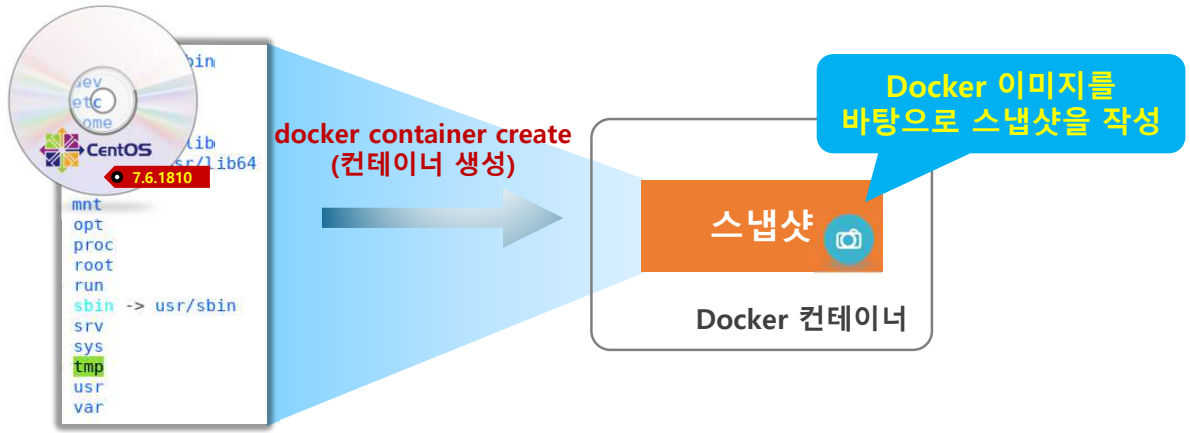
- 컨테이너 상태 전이



컨테이너 라이프 사이클 (2/6)

- **docker container create**
  - Usage: docker container create [OPTIONS] IMAGE [COMMAND] [ARG...]
  - 컨테이너를 작성만 할 뿐 시작하지는 않음
- **스냅샷 (snapshot)**
  - 도커 이미지 안에 존재하는 파일과 디렉토리를 특정 타이밍에서 추출한 것
  - Linux의 작동에 필요한 /etc나 /bin 등과 같은 디렉토리 및 파일들

```
[root@dockeredu ~]# docker container create -it centos:7.6.1810
cc73482275dacce9f0cf18940121b3c8578b83847dd4da24f4d02fb1bd44f7bf

[root@dockeredu ~]# docker container ls -a
CONTAINER ID    IMAGE           COMMAND       CREATED          STATUS     PORTS    NAMES
cc73482275da    centos7.6.1810 "/bin/bash"    4 seconds ago    Created             jovial_gates
```



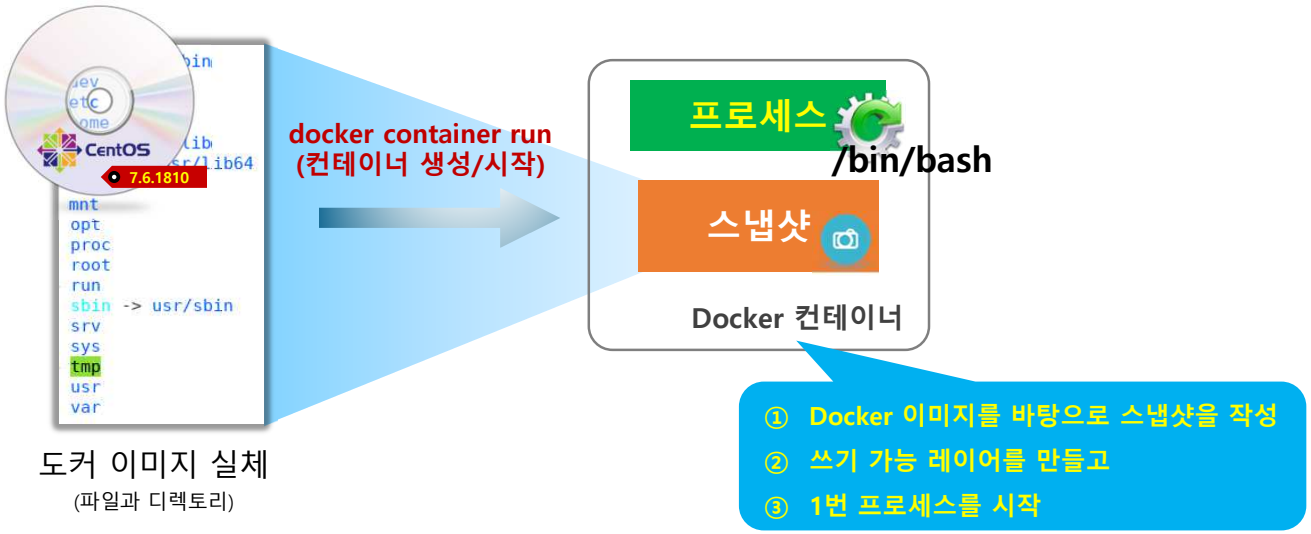
컨테이너 라이프 사이클 (3/6)

- **docker container run**
  - Usage: docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
  - 컨테이너를 생성하고 1번 프로세스를 시작

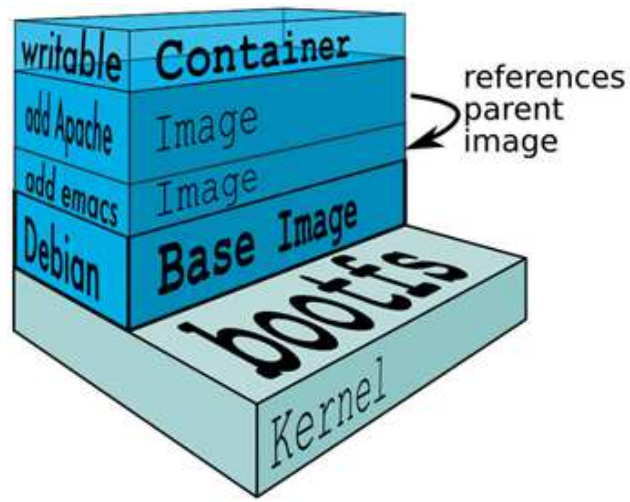
```
[root@dockeredu ~]# docker container run -it centos:7.6.1810
[root@49d039380799 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:48 pts/0    00:00:00 /bin/bash
root        16     1  0 02:48 pts/0    00:00:00 ps -ef
[root@49d039380799 /]#
```

ctrl + pq

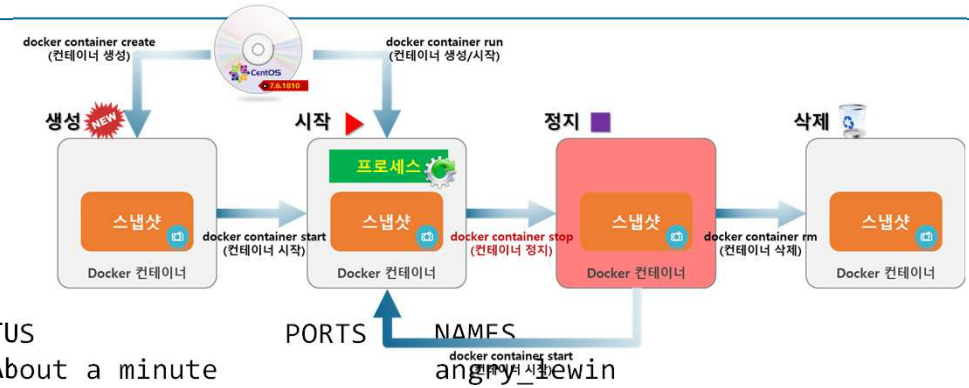
```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID  IMAGE           COMMAND       CREATED             STATUS              PORTS     NAMES
49d039380799  centos:7.6.1810  "/bin/bash"   About a minute ago  Up About a minute             angry_lewin
```



도커 이미지 실체
(파일과 디렉토리)



컨테이너 라이프 사이클 (4/6)



- **docker container stop**
  - Usage: docker container stop [OPTIONS] CONTAINER [CONTAINER...]
  - 컨테이너 정지
  - 컨테이너안의 프로세스에서 SIGKILL을 전송하여 종료시킴

```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE           COMMAND       CREATED              STATUS                  PORTS   NAMES
49d039380799   centos:7.6.1810  "/bin/bash"   About a minute ago   Up About a minute               angry_lewin
[root@dockeredu ~]# docker container stop 49d039380799
49d039380799
[root@dockeredu ~]# docker container stop angry_lewin
angry_lewin
[ [root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE           COMMAND       CREATED              STATUS                          PORTS   NAMES
49d039380799   centos:7.6.1810  "/bin/bash"   About a minute ago   Exited (137) About a minute ago         angry_lewin
```

- **Exited (137) 에서 137 의미**
  - 137은 프로세스의 Exit Code를 의미

Exit Code	설명
0	Successful completion of the command
1	General unknown error
2	Misuse of shell command
126	The command can't execute
127	Command not found

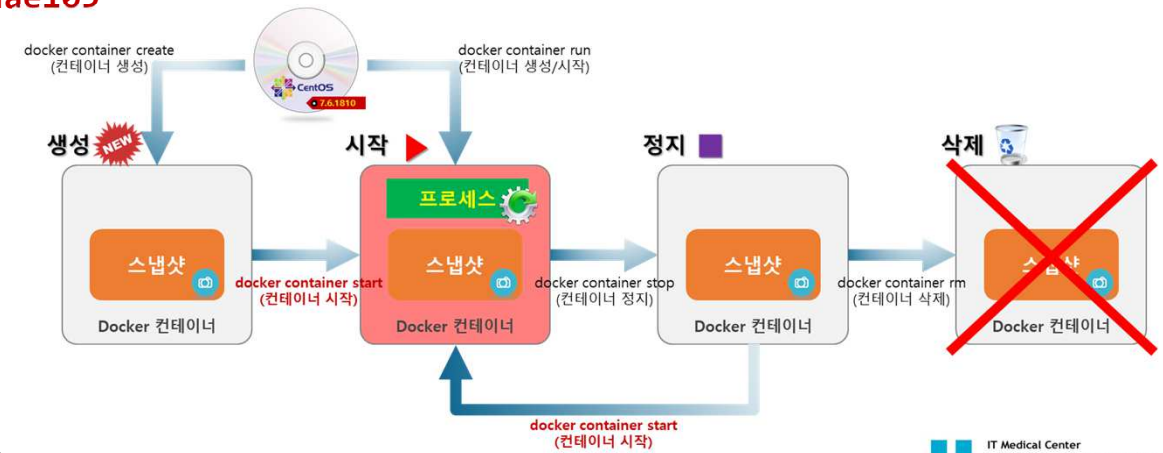
Exit Code	설명
128	Invalid exit argument
128+x	Fatal error with Linux signal x ✓ **137 = 128 + 9 (SIGKILL)**
130	Command terminated with Ctrl-C
255	Exit status out of range

컨테이너 라이프 사이클 (5/6)

- **docker container start**
  - Usage: docker container start [OPTIONS] CONTAINER [CONTAINER...]
  - 컨테이너 시작

옵션		설명
-a	--attach	✓ Attach STDOUT/STDERR and forward signals
-i	--interactive	✓ Attach container's STDIN

```
[root@dockeredu ~]# docker container start c7d856dae109
C7d856dae109
[root@dockeredu ~]# docker container ls -a
CONTAINER ID    IMAGE     COMMAND         CREATED            STATUS            PORTS    NAMES
25aa3cf3468d    centos    "/bin/bash"     13 minutes ago     Up 13 minutes              trusting_mclean
c7d856dae109    centos    "/bin/bash"     17 minutes ago     Up 2 minutes               relaxed_allen
[root@dockeredu ~]# docker container stop c7d856dae109
C7d856dae109
[root@dockeredu ~]# docker container start -ai c7d856dae109
[root@c7d856dae109 /]# ps -ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 02:21 pts/0    00:00:00 /bin/bash
root         16     1  0 02:21 pts/0    00:00:00 ps -ef
```

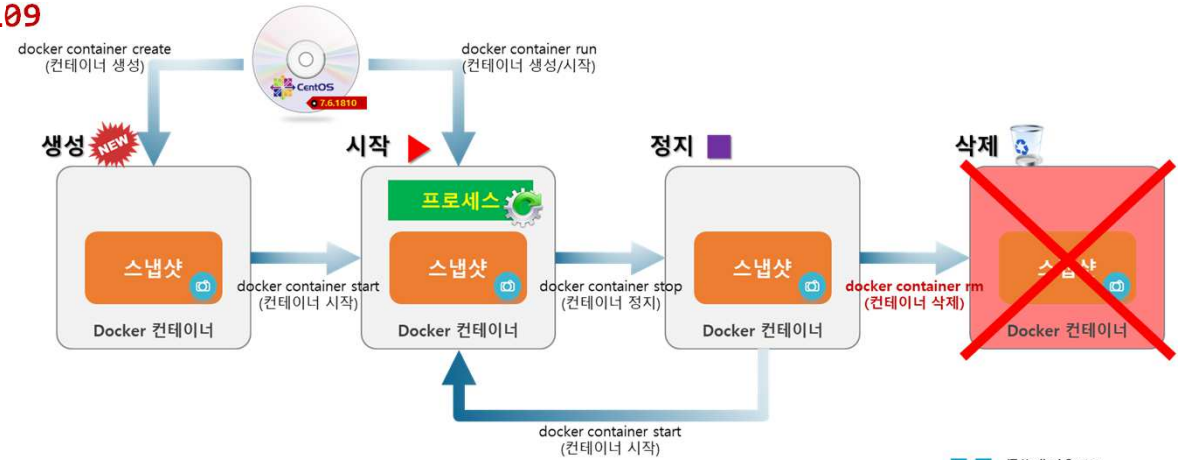


컨테이너 라이프 사이클 (6/6)

- **docker container rm**
  - Usage: docker container rm [OPTIONS] CONTAINER [CONTAINER...]
  - 컨테이너 삭제

옵션		설명
-f	--force	✓ Force the removal of a running container (uses SIGKILL)
-v	--volumes	✓ Remove the volumes associated with the container

```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID    IMAGE     COMMAND        CREATED           STATUS           PORTS    NAMES
c7d856dae109    centos    "/bin/bash"    17 minutes ago    Up 2 minutes              relaxed_allen

[root@dockeredu ~]# docker container rm c7d856dae109
Error response from daemon: You cannot remove a running container
c7d856dae109e344856c04db6acb9633364077b35a3809b4e794b7170218bfdc. Stop the container before attempting removal or force remove

[root@dockeredu ~]# docker container rm -f c7d856dae109
```



docker container run (1/7)

▪ 대화식 실행 옵션

옵션		설명
-i	--interactive	✓ Keep STDIN open even if not attached ✓ PID 1이 쉘이라면 명령은 입력할 수 있으나, 출력(에러)은 볼수 없다. ✓ 보통 -t 옵션과 함께 사용
-t	--tty	✓ Allocate a pseudo-TTY

```
[root@dockeredu ~]# docker container run -it centos
[root@4f6fee050fe4 /]# cal
    October 2018
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

[root@4f6fee050fe4 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:14 pts/0    00:00:00 /bin/bash
root        18     1  0 03:14 pts/0    00:00:00 ps -ef

[root@4f6fee050fe4 /]# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.56.1

[root@4f6fee050fe4 /]# exit
exit
```

docker container run (2/7)

- **컨테이너 이름 지정**

옵션		설명
	--name string	✓ Assign a name to the container ✓ 이름을 할당하면 CONTAINER_ID를 사용하지 않고 CONTAINER_NAME을 통해 컨테이너 제어가 가능하므로 관리하기가 훨씬 용이함 ✓ 이름을 지정하지 않으면 docker가 임의의 2단어(동사+명사)를 조합하여 네이밍함

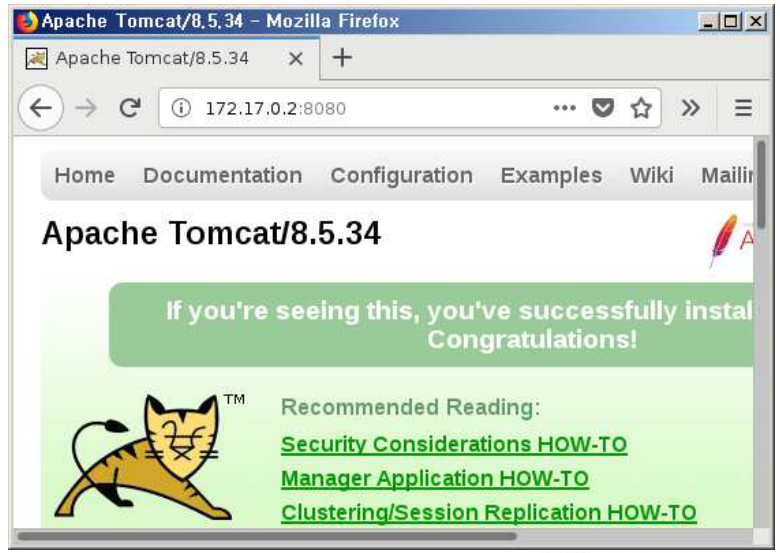
```
[root@dockeredu ~]# docker container run centos
[root@dockeredu ~]# docker container run --name con01 centos
[root@dockeredu ~]# docker container ls -a
CONTAINER ID    IMAGE     COMMAND        CREATED          STATUS                      PORTS     NAMES
14753e9fef36    centos    "/bin/bash"    14 seconds ago   Exited (0) 12 seconds ago             con01
c89018c89983    centos    "/bin/bash"    23 seconds ago   Exited (0) 23 seconds ago             angry_kirch
```

docker container run (3/7)

- 백그라운드 실행

옵션		설명
-d	--detach	✓ Run container in background and print container ID

```
[root@dockeredu ~]# docker container run -d --name=T01 tomcat:6
1b20ea3abf710e2afcb37c21f6b9f346c449bd9c7c4f629fd4348fcf63448ef0

[root@dockeredu ~]# docker container exec T01 cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      1b20ea3abf71

[root@dockeredu ~]# curl http://172.17.0.2:8080
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>Apache Tomcat/8.5.34</title>
        <link href="favicon.ico" rel="icon" type="image/x-icon" />
        <link href="favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <link href="tomcat.css" rel="stylesheet" type="text/css" />
```



docker container run (4/7)

- **컨테이너 로그 확인**
  - Usage: docker container logs [OPTIONS] CONTAINER
  - 컨테이너안의 1번 프로세스의 STDOUT/STDERR에 대해 출력

옵션		설명
-f	--follow	✓ Follow log output
-t	--timestamps	✓ Show timestamps

```
[root@dockeredu ~]# docker container run -d --name=T01 tomcat:6
1b20ea3abf710e2afcb37c21f6b9f346c449bd9c7c4f629fd4348fcf63448ef0

[root@dockeredu ~]# docker container ls -a
CONTAINER ID    IMAGE     COMMAND             CREATED          STATUS          PORTS      NAMES
1b20ea3abf71    tomcat    "catalina.sh run"   11 minutes ago   Up 11 minutes   8080/tcp   T01

[root@dockeredu ~]# docker container logs -ft T01
2018-10-24T04:57:24.436718528Z 24-Oct-2018 04:57:24.434 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:        Apache Tomcat/8.5.34
2018-10-24T04:57:24.436764906Z 24-Oct-2018 04:57:24.435 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:          Sep 4 2018 22:28:22 UTC
2018-10-24T04:57:24.436770592Z 24-Oct-2018 04:57:24.436 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:        8.5.34.0
2018-10-24T04:57:24.436774670Z 24-Oct-2018 04:57:24.436 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:             Linux
2018-10-24T04:57:24.436778787Z 24-Oct-2018 04:57:24.436 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:          3.10.0-862.el7.x86_64
```

docker container run (5/7)

- **포트 노출**
  - 호스트 OS의 넷필터 NAT룰에 Docker Container IP가 포트 포워딩 됨
  - iptables -t nat -vnL

옵션		설명
-p	--publish list	✓ Publish a container's port(s) to the host

```
[root@dockeredu ~]# docker container run -d --name T01 -p 7070:8080 tomcat:6
A1248d833de061ebad25b339b8943f5125af213b44ee0e6b114548afd2f9e0ea

[root@dockeredu ~]# iptables -t nat -vnL
Chain PREROUTING (policy ACCEPT 1 packets, 229 bytes)
 pkts bytes target     prot opt in     out     source               destination
   17   741 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0           ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 1 packets, 229 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 1 packets, 70 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 1 packets, 70 bytes)
 pkts bytes target     prot opt in     out        source               destination
   66  4079 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0
    0     0 MASQUERADE  tcp  --  *      *         172.17.0.2           172.17.0.2           tcp dpt:8080

Chain DOCKER (2 references)
 pkts bytes target     prot opt in       out     source               destination
    0     0 RETURN     all  --  docker0  *       0.0.0.0/0            0.0.0.0/0
    0     0 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0           tcp dpt:7070 to:172.17.0.2:8080
```

호스트의 7070포트를
컨테이너의 8080 포트로 매핑

docker container run (6/7)

- 컨테이너 환경변수 설정

옵션		설명
-e	--env list	✓ Set environment variables
	--env-file	✓ Read in a file of environment variables

```
[root@dockeredu ~]# docker container run -it -e FOO=BAR -e JAVA_HOME=/usr/local/jdk1.8 centos
[root@317767a03f6a /]# printenv
HOSTNAME=317767a03f6a
TERM=xterm
...
FOO=BAR
JAVA_HOME=/usr/local/jdk1.8
HOME=/root
_=/usr/bin/printenv

[root@dockeredu ~]# vi env_list
FOO=BAR
JAVA_HOME=/usr/local/jdk1.8
[root@dockeredu ~]# docker container run -it --env-file=env_list centos
[root@de1ca1c57c82 /]# printenv
HOSTNAME=de1ca1c57c82
TERM=xterm
FOO=BAR
JAVA_HOME=/usr/local/jdk1.8
...
_=/usr/bin/printenv
```

CLOUDFOUNDRY

IT Medical Center JADECROSS

docker container run (7/7)

- **현재 작업 디렉터리(Current Working Directory) 설정**
  - 해당 디렉토리가 없으면 생성

옵션		설명
-w	--workdir string	✓ Working directory inside the container

```
[root@dockeredu ~]# docker container run -it -w=/tensorflow centos
[root@c03fca7cc0eb tensorflow]# pwd
/tensorflow
[root@c03fca7cc0eb tensorflow]# printenv
HOSTNAME=c03fca7cc0eb
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/tensorflow
SHLVL=1
HOME=/root
_=/usr/bin/printenv
```

docker container prune

- **정지된 컨테이너 일괄 삭제**
  - Usage: docker container prune [OPTIONS]

옵션		설명
-f	--force	✓ Do not prompt for confirmation

```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE     COMMAND             CREATED          STATUS                       PORTS   NAMES
51c2aac01378   centos    "/bin/bash"         10 minutes ago   Exited (0) 9 minutes ago             unruffled_murdock
ee947812fb5e   tomcat    "catalina.sh run"   20 minutes ago   Exited (143) 2 seconds ago           T01
[root@dockeredu ~]#
[root@dockeredu ~]# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
51c2aac013782dc0e8d42aa73b91fe55e5f9dbd17e9c5464d91ac5d7373bebc0
ee947812fb5e3dce2ca78928d633786427f4d4313eeeb1e0d179dfc0f7b44fa5

Total reclaimed space: 8.5kB
[root@dockeredu ~]#
```

docker container pause/unpause

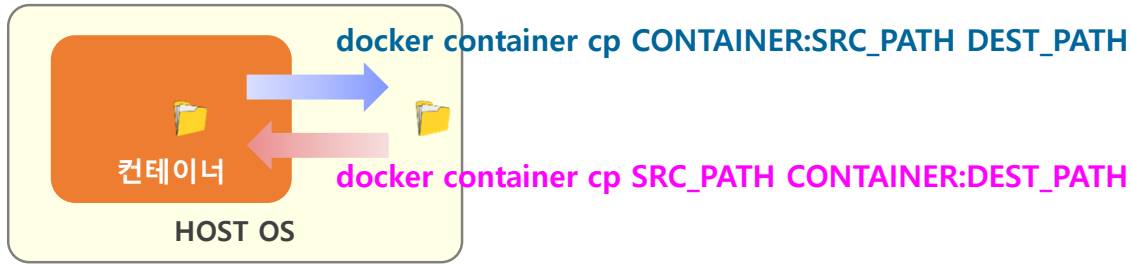
- 컨테이너 일시 중지
  - Usage: docker container pause CONTAINER [CONTAINER...]

- 컨테이너 재개
  - Usage: docker container unpause CONTAINER [CONTAINER...]

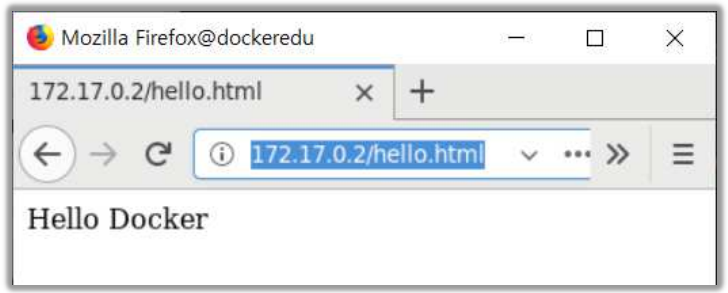
```
[root@dockeredu ~]# docker container run -d --name T01 tomcat:6
5d8a96156f4550aa06412f502edebc352914e51b3b5d27cd0a6820ac773e7c46

[root@dockeredu ~]# docker container run -d --name T02 tomcat:6
1feca4af9ec90e1ddaff821a79937572244c3f69a12fd51ef3cf33e5884758d2

[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE    COMMAND             CREATED         STATUS         PORTS      NAMES
1feca4af9ec9   tomcat   "catalina.sh run"   3 seconds ago   Up 2 seconds   8080/tcp   T02
5d8a96156f45   tomcat   "catalina.sh run"   5 seconds ago   Up 4 seconds   8080/tcp   T01

[root@dockeredu ~]# docker container pause T01 T02
T01
T02
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE    COMMAND             CREATED          STATUS                    PORTS      NAMES
1feca4af9ec9   tomcat   "catalina.sh run"   24 seconds ago   Up 23 seconds (Paused)    8080/tcp   T02
5d8a96156f45   tomcat   "catalina.sh run"   26 seconds ago   Up 25 seconds (Paused)    8080/tcp   T01

[root@dockeredu ~]# docker container unpause T01
T01
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE    COMMAND             CREATED          STATUS                    PORTS      NAMES
1feca4af9ec9   tomcat   "catalina.sh run"   45 seconds ago   Up 44 seconds (Paused)    8080/tcp   T02
5d8a96156f45   tomcat   "catalina.sh run"   47 seconds ago   Up 46 seconds             8080/tcp   T01
```

docker container cp

- HOST와 컨테이너간의 파일 복사




```
[root@dockeredu ~]# echo "Hello Docker" > hello.html
[root@dockeredu ~]# docker container run -d --name=web httpd:2.4
ebd6e1c42adc8966da7768f711ad71272f85c6a982eda2ea9463b968f5ec263d

[root@dockeredu ~]# docker container cp hello.html web:/usr/local/apache2/htdocs

[root@dockeredu ~]# curl http://172.17.0.2/hello.html
Hello Docker
```




```
[root@dockeredu ~]# docker container cp web:/usr/local/apache2/htdocs/index.html ~/
[root@dockeredu ~]# ll ~/index.html
-rw-r--r-- 1 root root 45 Jun 12  2007 /root/index.html
```

docker container exec

- **가동 컨테이너에서 새로운 프로세스 실행**
  - 백그라운드에서 실행되고 있는 컨테이너에 액세스하고 싶을 때
  - docker container attach 명령으로 연결해도 쉘이 작동하지 않는 경우에 유용함
  - Usage: docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]

옵션		설명
-i	--interactive	✓ Keep STDIN open even if not attached
-t	--tty	✓ Allocate a pseudo-TTY

```
[root@dockeredu ~]# docker container run -d --name=web httpd:2.4
2a4ea9800b37c03be01938b797b7521ccfd83bd031989320d93e00486a9c18e6
[root@dockeredu ~]# docker container attach web
-- 컨테이너의 1번 process인 httpd에게 STDIN, STDOUT을 연결해도 기타 명령을 내릴 수가 없다.
-- "ctrl + c" httpd 프로세스를 종료하면, 결국에는 1번 proces가 종료되기 때문에 컨테이너도 종료되게 된다.
^C[Wed Apr 17 15:00:36.763168 2019] [mpm_event:notice] [pid 1:tid 140150371184704] AH00491: caught SIGTERM, shutting down

[root@dockeredu ~]# docker container ls -a -f name=web
CONTAINER ID    IMAGE         COMMAND              CREATED             STATUS                        PORTS        NAMES
2a4ea9800b37    httpd:2.4     "httpd-foreground"   28 seconds ago      Exited (0) 19 seconds ago                  web
[root@dockeredu ~]# docker container start web
web
[root@dockeredu ~]# docker container exec -it web /bin/bash
root@2a4ea9800b37:/usr/local/apache2# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      2a4ea9800b37
```

docker container diff

- **컨테이너 파일시스템의 변경내역 확인**
  - 컨테이너의 쓰기가능 레이어(Writable Layer)의 변경 내역 확인
  - Usage: docker container diff CONTAINER

```
[root@dockeredu ~]# docker container run -it --name=c01 centos
[root@b4824c542cc9 /]# useradd newuser
[root@b4824c542cc9 /]# exit
exit
[root@dockeredu ~]# docker container diff c01
C /home
A /home/newuser
A /home/newuser/.bashrc
A /home/newuser/.bash_logout
A /home/newuser/.bash_profile
C /var
C /var/spool
C /var/spool/mail
A /var/spool/mail/newuser
C /var/log
C /var/log/lastlog
C /root
A /root/.bash_history
C /etc
C /etc/shadow-
C /etc/passwd
C /etc/gshadow-
C /etc/passwd-
C /etc/shadow
C /etc/group-
C /etc/gshadow
C /etc/group
```

  - 변경의 구분

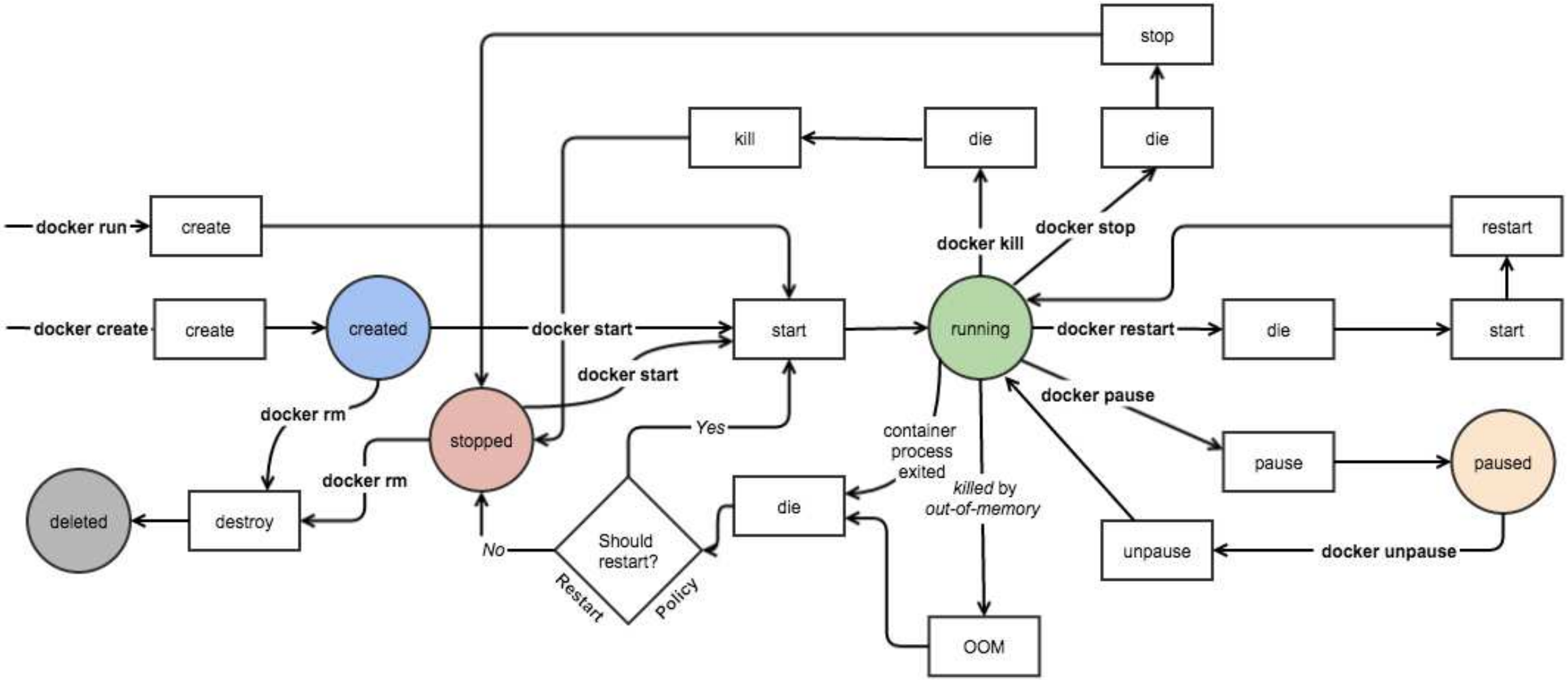
구분	설명
A	Append
D	Delete
C	Change

docker container inspect

- 컨테이너 상세 정보 확인
  - Usage: docker container inspect [OPTIONS] CONTAINER [CONTAINER...]

```
[root@dockeredu ~]# docker container run -d --name=web httpd:2.4
8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d
[root@dockeredu ~]# docker container inspect web
[
    {
        "Id": "8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d",
        "Created": "2019-04-17T14:50:45.001625846Z",
        "Path": "httpd-foreground",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 16144,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2019-04-17T14:50:45.417569299Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:d4a07e6ce470b5888ec230eaa25a9b7ebcc0b2d8722bf0720464770867b8cf32",
        "ResolvConfPath": "/var/lib/docker/containers/8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d/hostname",
        "HostsPath": "/var/lib/docker/containers/8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d/hosts",
        "LogPath": "/var/lib/docker/containers/8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d/8b6aa7d83c0c10ae939ec6942c2d07a7d7a5f23df589d-json.log",
        "Name": "/web",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
```

참고) 컨테이너 라이프사이클



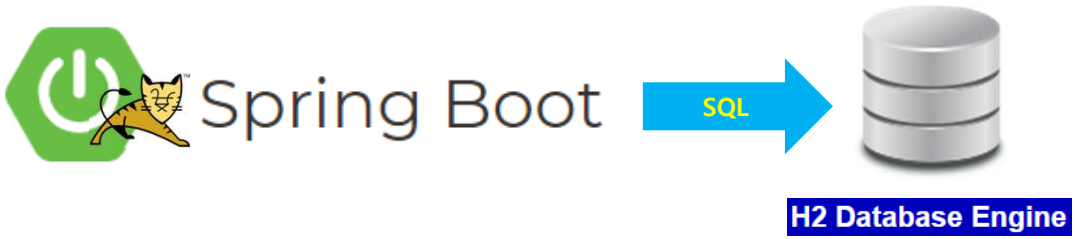
도커 이미지

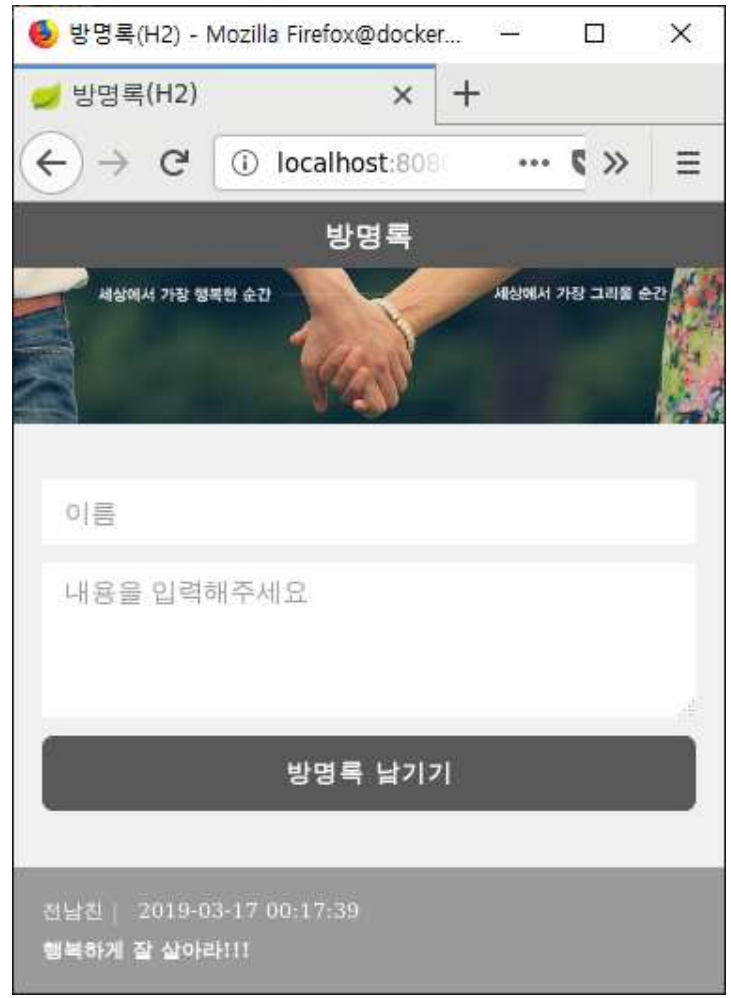
방명록 샘플 어플리케이션

- **방명록 웹 어플리케이션 실행**
  - 실행 후 http://localhost:8080 으로 접속

```
[root@dockeredu ~]# cd ~/guestbook/
[root@dockeredu guestbook]# java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-19 10:23:30.323  INFO 3917 --- [           main] c.j.guestbook.GuestbookH2Application
     : Starting GuestbookH2Application v0.0.1-SNAPSHOT on dockeredu with PID 3917 (/root/gues
tbook/guestbook_H2.jar started by root in /root/guestbook)
2019-04-19 10:23:30.327  INFO 3917 --- [           main] c.j.guestbook.GuestbookH2Application
     : No active profile set, falling back to default profiles: default
2019-04-19 10:23:31.935  INFO 3917 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebSer
ver  : Tomcat initialized with port(s): 8080 (http)
2019-04-19 10:23:31.975  INFO 3917 --- [           main] o.apache.catalina.core.StandardServi
ce   : Starting service [Tomcat]
```

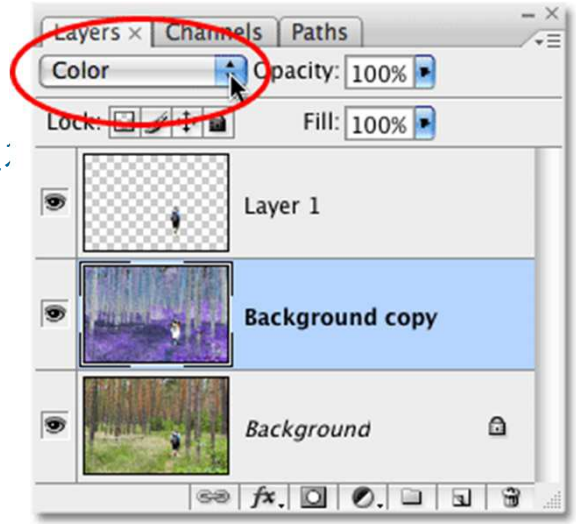
- **어플리케이션 구조**
  - Spring Boot + H2 (Memory DB)





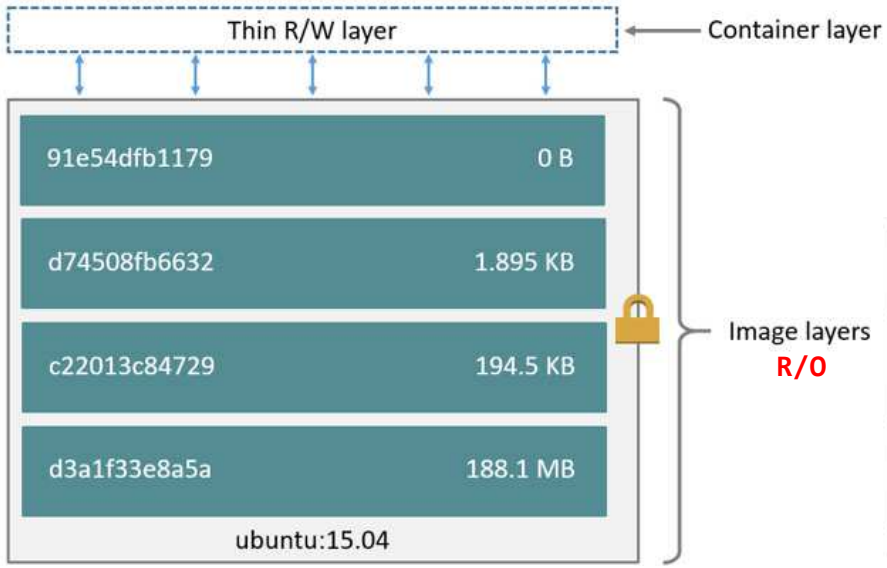
UFS(Union File System) 과 Layer

- **UFS(Union File System)**
  - "union mount"를 구현하고 Layer를 생성하여 작동하는 파일 시스템
  - 도커는 "copy-on-write"기술과 UFS를 사용하여 컨테이너에 대한 빌딩 블록을 제공하여 매우 가볍고 빠르
  - UFS 구현체
    - OverlayFS - 도커는 overlay, overlay2 storage driver를 제공
    - AUFS - Advanced multi layered Unification FileSystem
    - Btrfs - B-tree file system

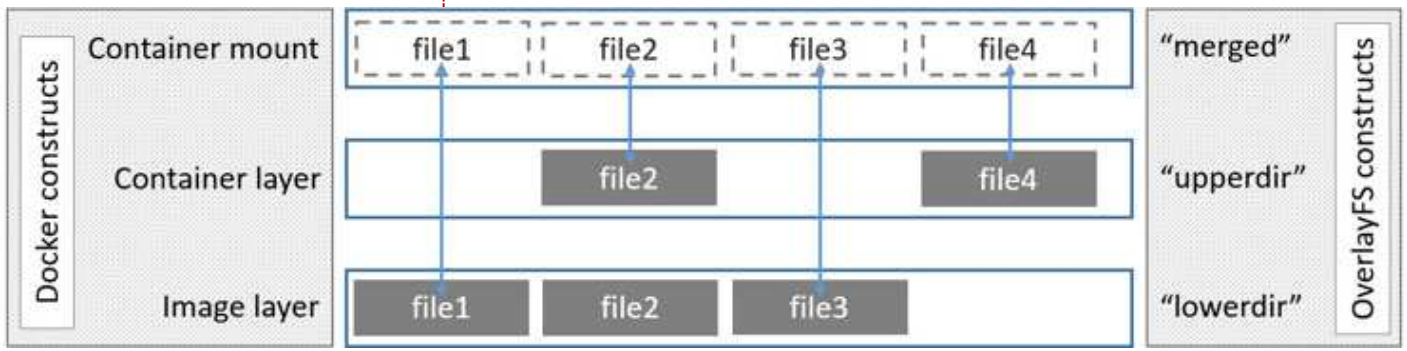


< 포토샵의 레이어 기능>

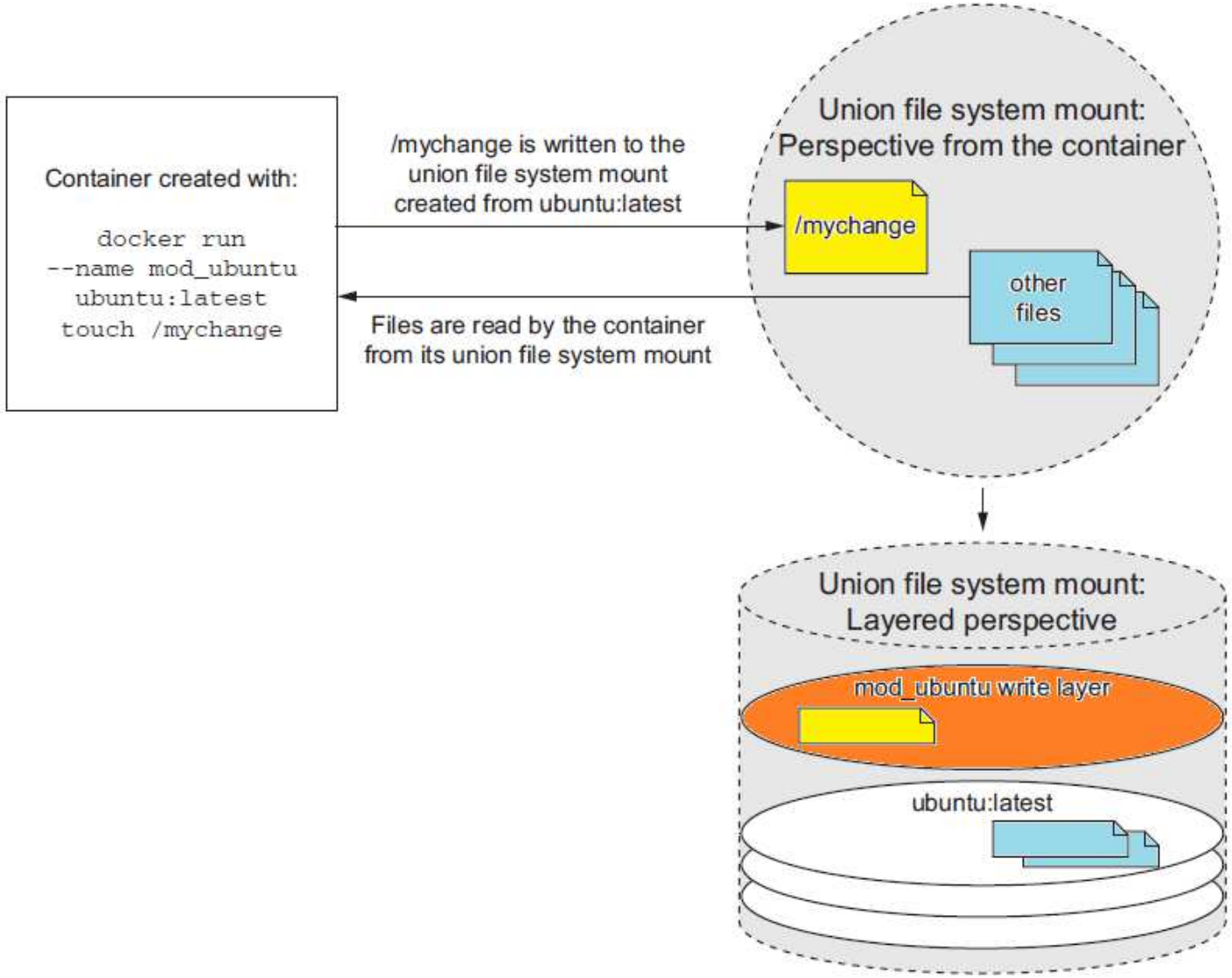
- **컨테이너와 쓰기 가능(Writable) 레이어**
  - 컨테이너가 생성되면 최상위에 쓰기 가능한 레이어가 생성되고, 컨테이너에서 변경된 파일을 이 레이어에 저장함



CoW (Copy on Write)
- 컨테이너 파일시스템상에서 file1, file2, file3, file4는 읽기전용으로 보이고 수정을 하는 시점에 Writable Layer로 Copy하여 수정
- 컨테이너에서 보이는 파일들은 서로 다른 레이어에 속해 있음

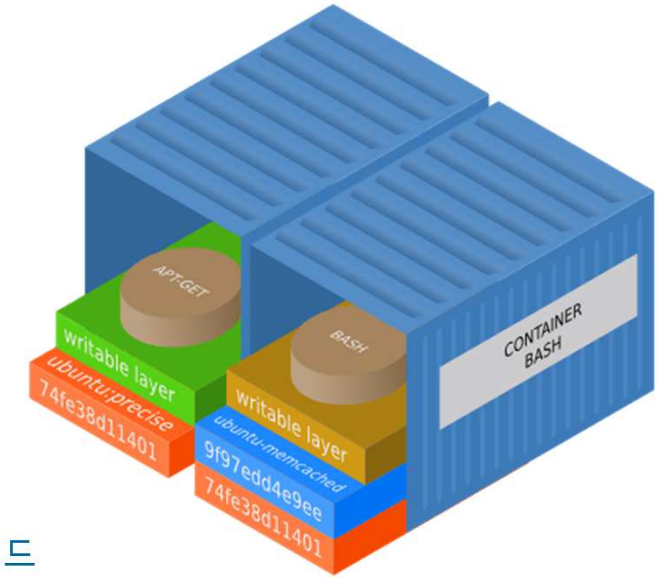


컨테이너와 쓰기 가능(Writable) 레이어

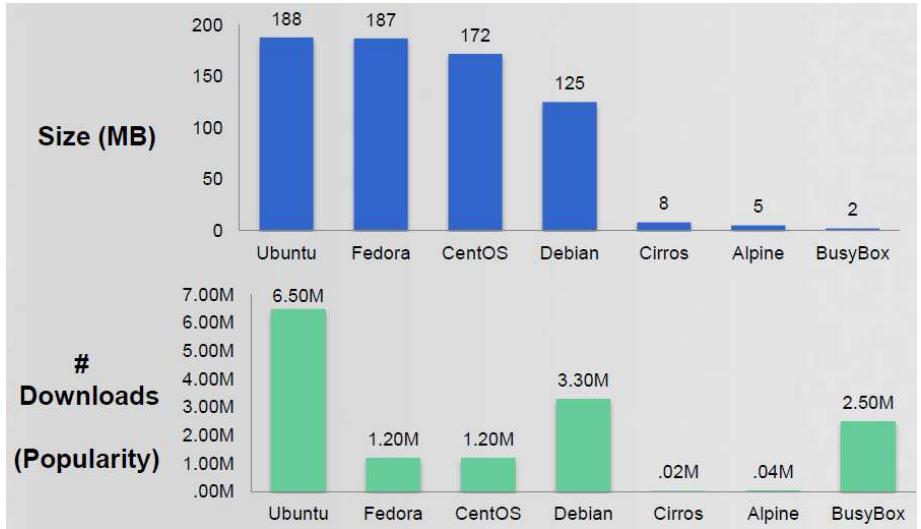


도커 이미지와 레이어

- **도커 이미지**
  - 컨테이너 실행을 위한 기반을 제공하는 읽기전용 템플릿
  - 애플리케이션 실행에 필요한 모든것을 포함하는 템플릿
  - 레이어 기반의 union 파일시스템인 overlay2를 사용
  - Dockerfile로 자신만의 이미지 생성

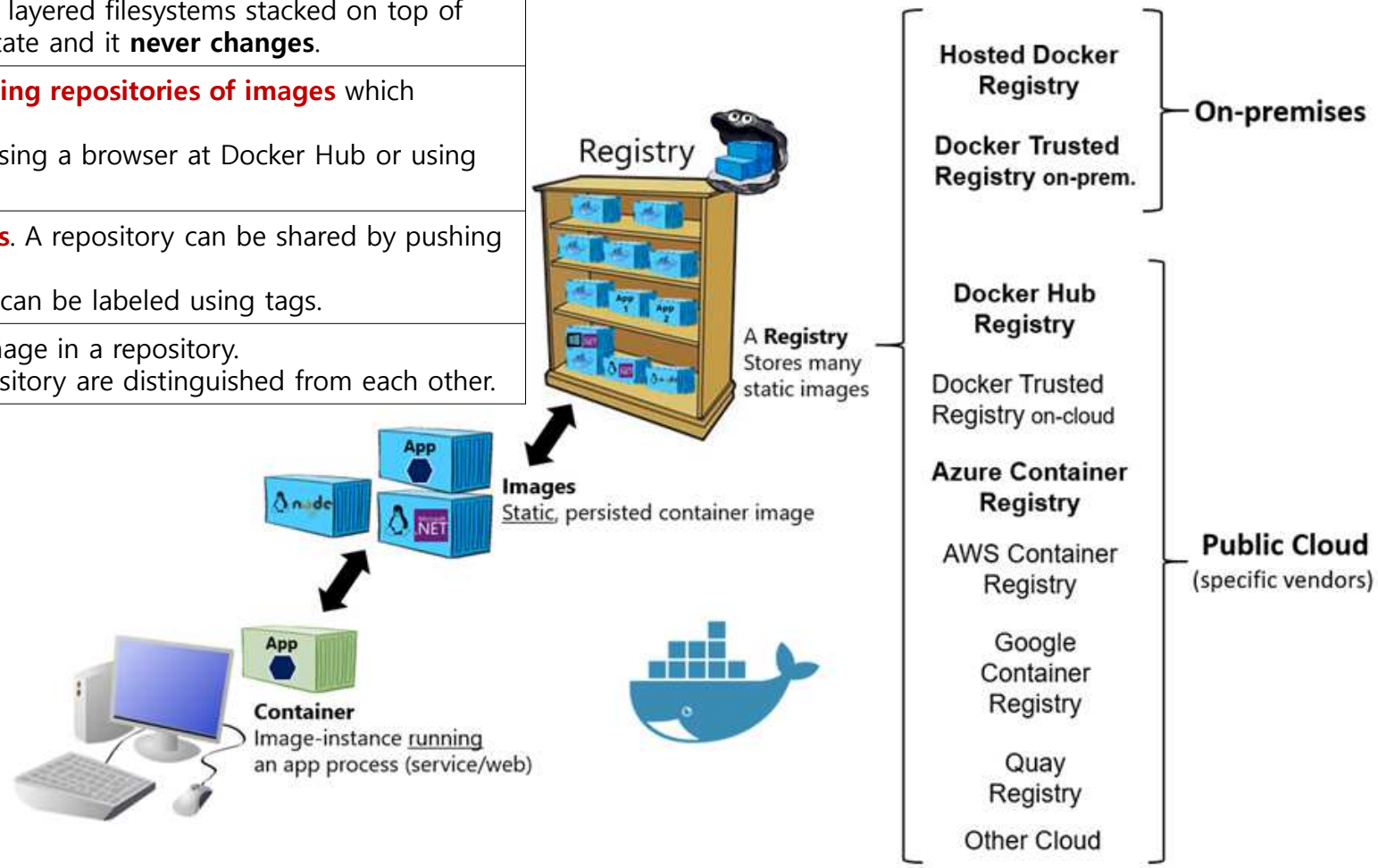


- **레이어**
  - 이미지는 연결된 여러 레이어로 구성됨
  - overlay2라는 레이어 파일 시스템을 사용
  - 레이어는 읽기전용이며, 이미지 다운로드시 기존 레이어를 재활용하며 변경 부분만 다운로드
    - 네트워크 대역폭, 디스크 저장 크기, 이미지 처리 시간 단축
  - 이미지 생성 프로세스를 레이어로 기록
  - 이미지의 레이어 작업 내용 보기
    - docker image history
  - 이미지를 단일 레이어로 만들기
    - docker container export 후 docker import



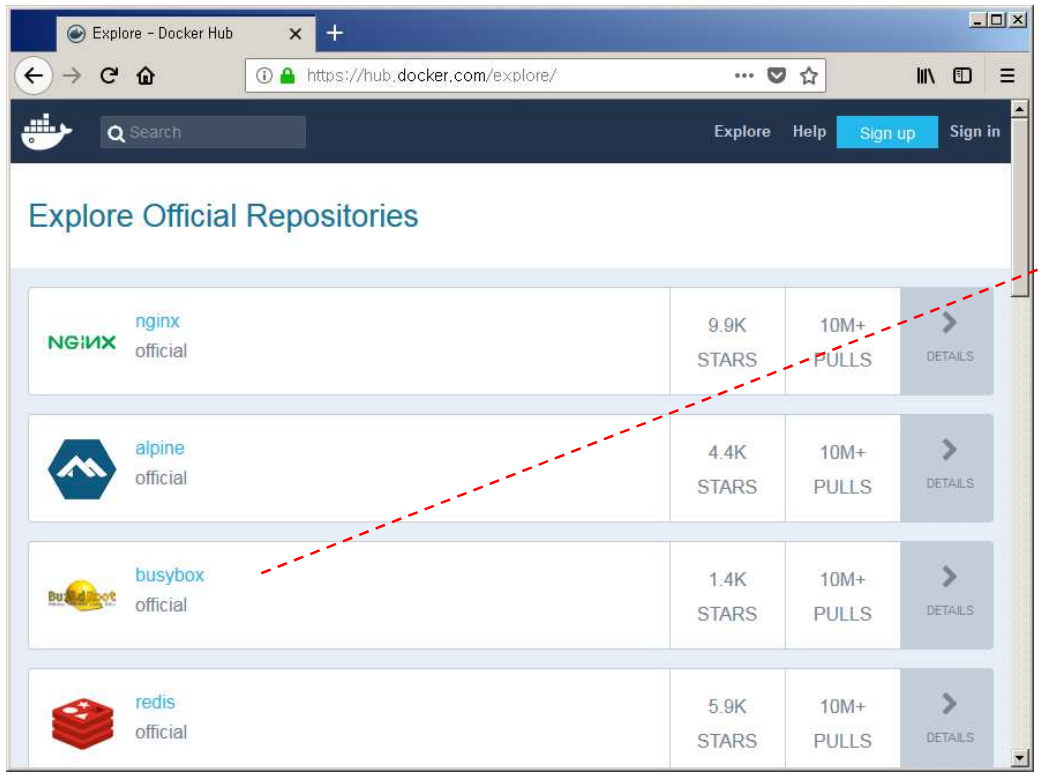
도커 이미지 관련 용어 정리

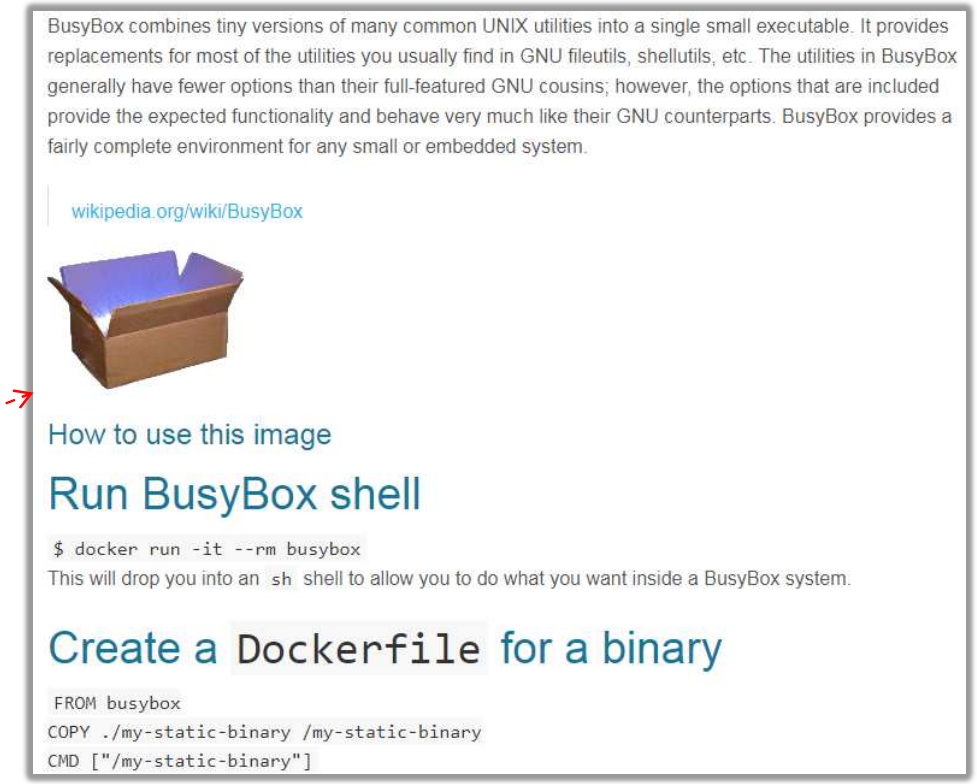
용어	설명
image	Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it **never changes**.
registry	A Registry is a **hosted service containing repositories of images** which responds to the Registry API. The default registry can be accessed using a browser at Docker Hub or using the docker search command.
repository	A repository is a **set of Docker images**. A repository can be shared by pushing it to a registry server. The different images in the repository can be labeled using tags.
tag	A tag is a label applied to a Docker image in a repository. Tags are how various images in a repository are distinguished from each other.



Docker Hub

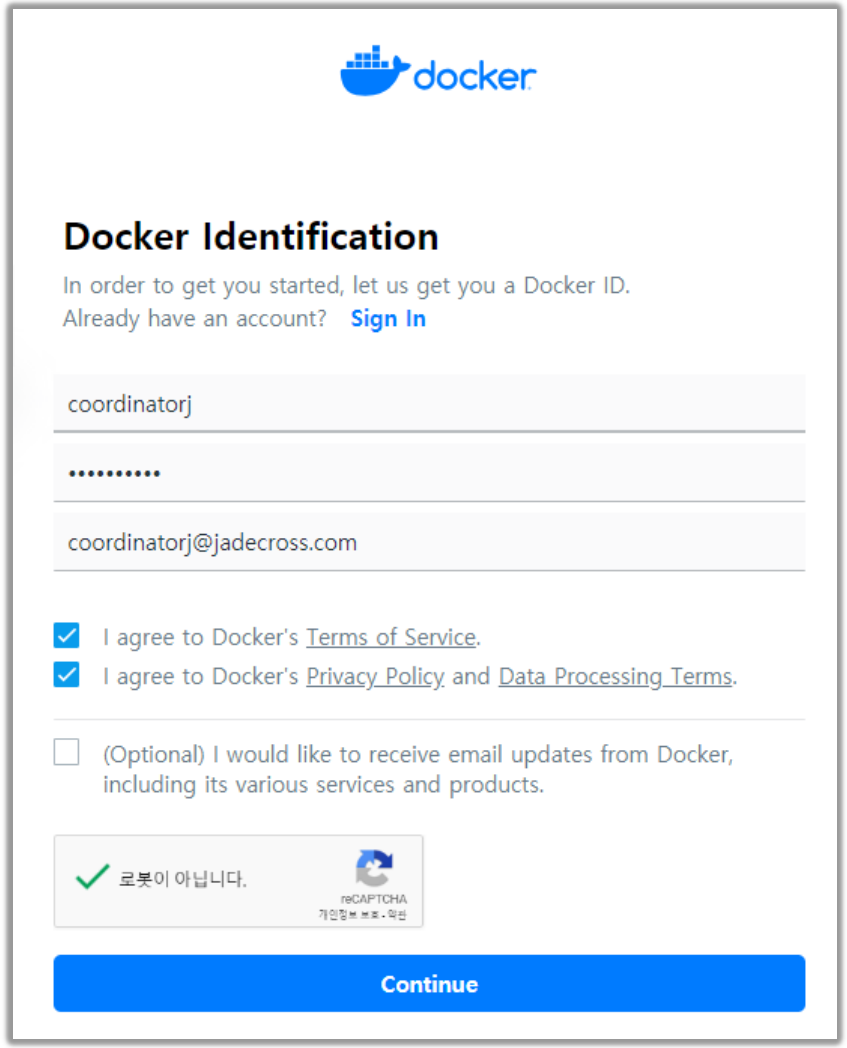
- **https://hub.docker.com**
  - docker 공식 이미지를 관리하는 무료 레지스트리
  - 개인이 작성한 Docker 이미지도 공개 가능
- **Repository Info**
  - Docker Hub에 공개되어 있는 이미지의 상세 정보 표시
  - 버전정보나 주의사항, 지원하는 Docker 버전 등.





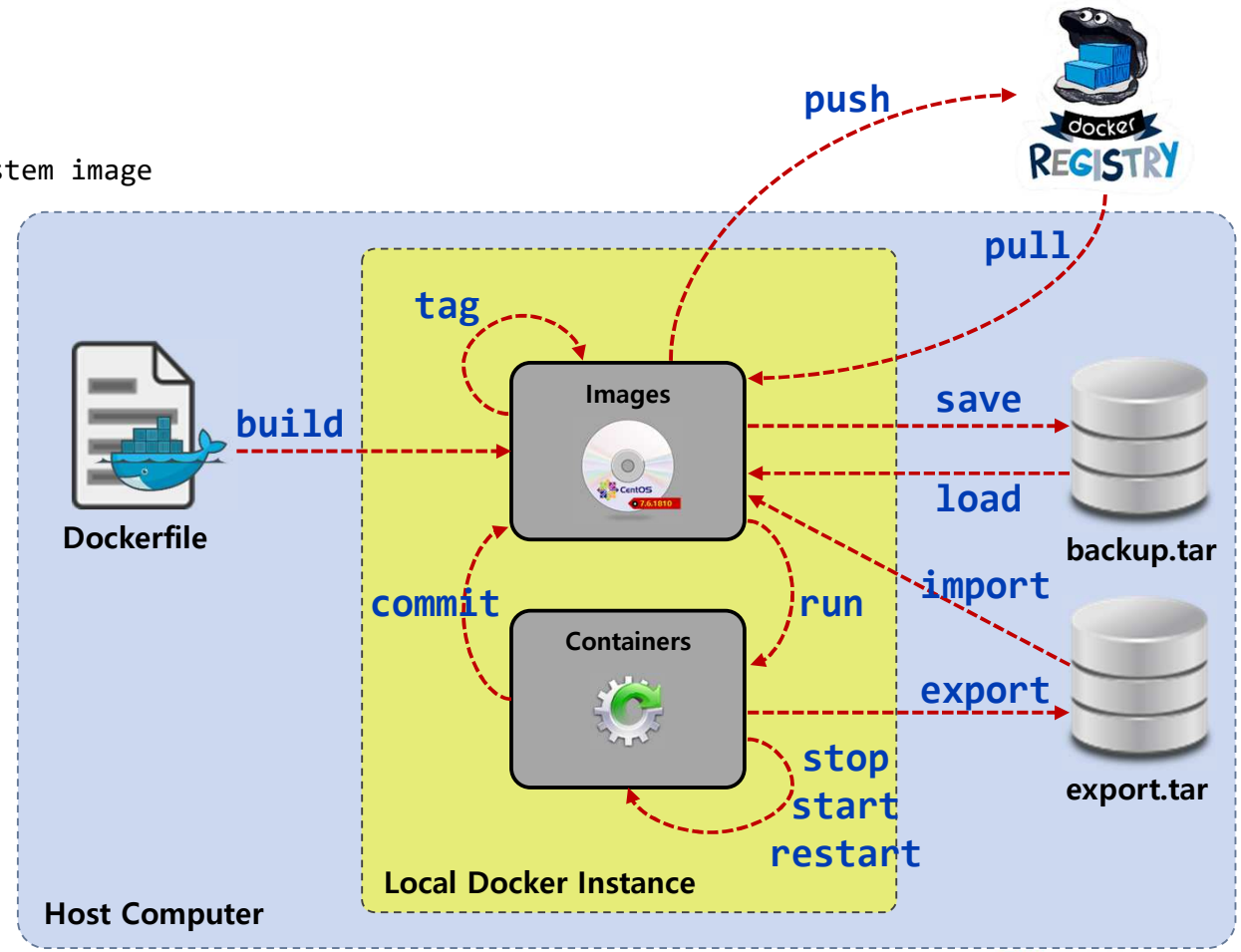
도커 허브 가입

- https://hub.docker.com/signup



도커 이미지 관련 작업


```
[root@dockeredu ~]# docker image --help

Usage:  docker image COMMAND

Manage images

Commands:
  build     Build an image from a Dockerfile
  history   Show the history of an image
  import    Import the contents from a tarball to create a filesystem image
  inspect   Display detailed information on one or more images
  load      Load an image from a tar archive or STDIN
  ls        List images
  prune     Remove unused images
  pull      Pull an image or a repository from a registry
  push      Push an image or a repository to a registry
  rm        Remove one or more images
  save      Save one or more images to a tar archive
  tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

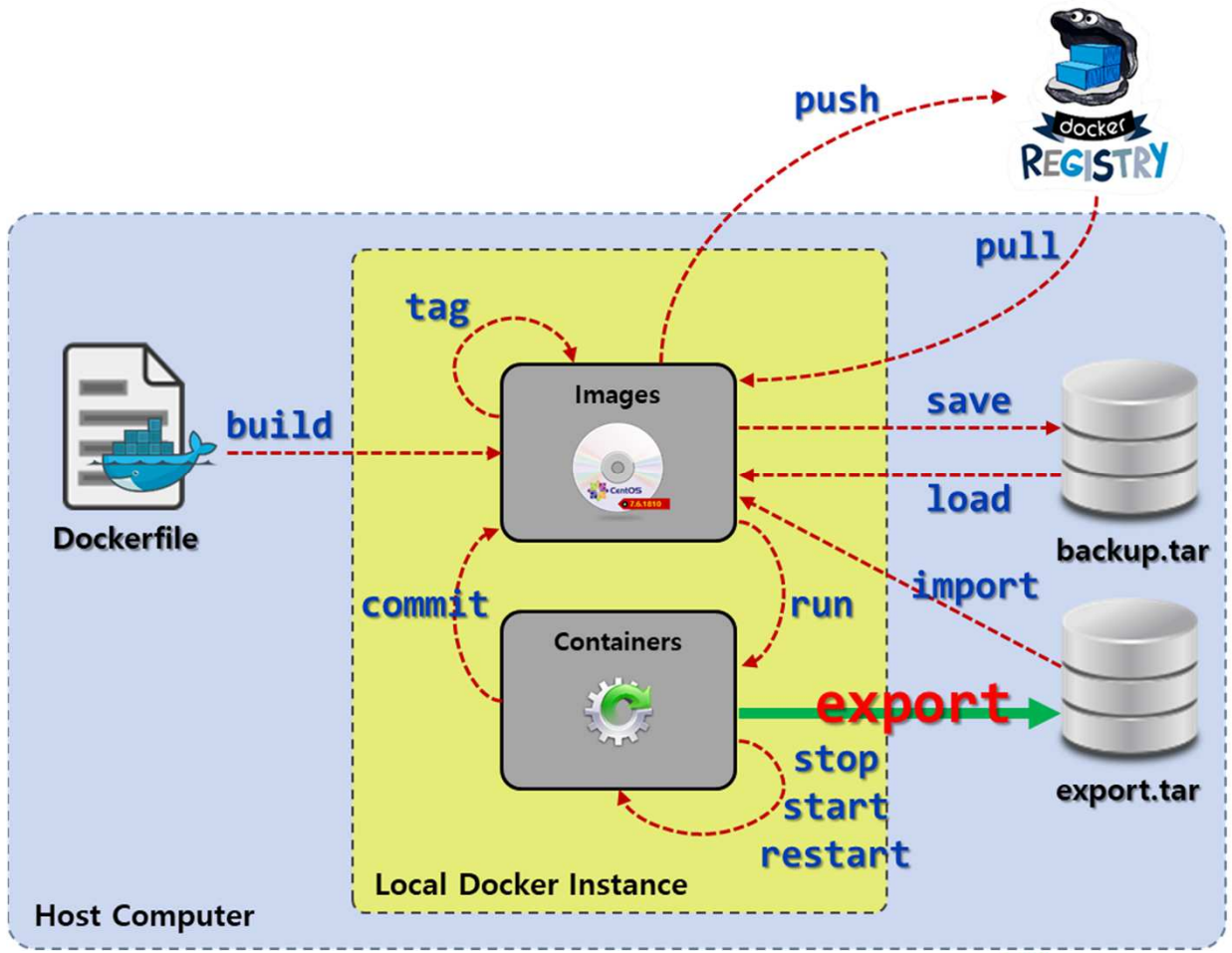


docker container export (1/4)

- **컨테이너안의 File System을 tar 파일로 출력**

```
[root@dockeredu ~]# docker container export --help

Usage:  docker container export [OPTIONS] CONTAINER

Export a container's filesystem as a tar archive

Options:
  -o, --output string   Write to a file, instead of STDOUT
```

export 를 하게 되면 모든 Layer가 하나의 Layer로 묶이게 됨



docker container export (2/4)

- **LAB 1. 방명록을 서비스하는 guestbookH2_c01 컨테이너 만들기 (1/3)**

① openjdk:8 이미지로 guestbookH2_c01 컨테이너 생성하고 시작

```
[root@dockeredu ~]# docker container run -it --name guestbookH2_c01 openjdk:8
root@116742a19b81:/#
```

② guestbookH2_c01 의 IP 확인

```
root@116742a19b81:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      116742a19b81
```

③ ctrl + P Q 를 입력하여 컨테이너 쉘에서 빠져나오기

```
root@116742a19b81:/#  ctrl + p q 입력
```

④ guestbook_H2.jar Spring Boot 웹어플리케이션을 guestbookH2_c01 컨테이너의 / 폴더에 복사

```
[root@dockeredu ~]# docker container cp ~/guestbook/guestbook_H2.jar guestbookH2_c01:/
```

⑤ attach 명령을 이용해 컨테이너의 1번 프로세스인 bash 쉘에 진입 후 guestbook_H2.jar 파일이 복사되었는지 확인

```
[root@dockeredu ~]# docker container attach guestbookH2_c01
root@116742a19b81:/# ls -l /guestbook_H2.jar
-rw-r--r-- 1 root root 22905356 Apr 19 01:14 /guestbook_H2.jar
```

docker container export (3/4)

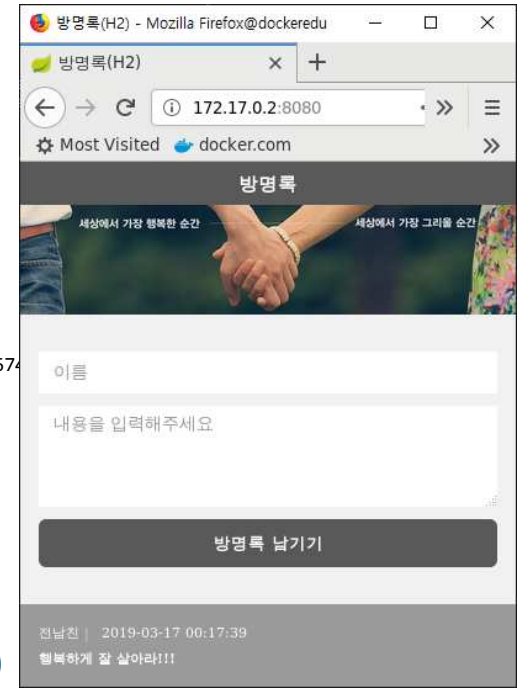
- **LAB 1. 방명록을 서비스하는 guestbookH2_c01 컨테이너 만들기 (2/3)**

⑥ guestbook_H2.jar Spring Boot 웹 어플리케이션 시작

```
root@116742a19b81:/# java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-19 04:10:12.777  INFO 27 --- [           main] c.j.guestbook.GuestbookH2Application     : Starting GuestbookH2Application v0.0.1-SNAPSHOT on 11674
started by root in /) ...
```

⑦ 웹브라우저로 방명록 웹 접속

```
http://172.17.0.2:8080
```



⑧ "ctrl + c" 를 입력하여 SpringBoot 어플리케이션을 종료하고 exit 명령으로 쉘 종료(결국은 컨테이너 종료)

```
^C 2019-04-19 04:20:11.546  INFO 27 --- [       Thread-4] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorService 'applicationTaskExecutor'
2019-04-19 04:20:11.547  INFO 27 --- [       Thread-4] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2019-04-19 04:20:11.552  INFO 27 --- [       Thread-4] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
root@116742a19b81:/# exit
exit
```

⑨ guestbookH2_c01 컨테이너 파일시스템 내용을 tar 파일로 출력

```
[root@dockeredu ~]# docker container export -o guestbookH2_c01.tar guestbookH2_c01
                  = docker container export guestbookH2_c01 > guestbookH2_c01.tar
```

표준출력을 리다이렉션하여 tar파일 생성

docker container export (4/4)

▪ LAB 1. 방명록을 서비스하는 guestbookH2_c01 컨테이너 만들기 (3/3)

⑩ export 된 guestbookH2_c01.tar 파일 내용 보기

```
[root@dockeredu ~]# tar tvf guestbookH2_c01.tar | more
-rwxr-xr-x 0/0               0 2019-04-19 16:03 .dockerenv
drwxr-xr-x 0/0               0 2019-03-27 09:43 bin/
-rwxr-xr-x 0/0         1099016 2017-05-16 04:45 bin/bash
-rwxr-xr-x 0/0           35448 2017-01-30 03:30 bin/bunzip2
hrwxr-xr-x 0/0               0 2017-01-30 03:30 bin/bzcat link to bin/bunzip2
lrwxrwxrwx 0/0               0 2017-01-30 03:30 bin/bzcmp -> bzdiff
-rwxr-xr-x 0/0            2140 2017-01-30 03:30 bin/bzdiff
lrwxrwxrwx 0/0               0 2017-01-30 03:30 bin/bzegrep -> bzgrep
-rwxr-xr-x 0/0            4877 2017-01-30 03:30 bin/bzexe
lrwxrwxrwx 0/0               0 2017-01-30 03:30 bin/bzfgrep -> bzgrep
-rwxr-xr-x 0/0            3642 2017-01-30 03:30 bin/bzgrep
hrwxr-xr-x 0/0               0 2017-01-30 03:30 bin/bzip2 link to bin/bunzip2
-rwxr-xr-x 0/0           14664 2017-01-30 03:30 bin/bzip2recover
lrwxrwxrwx 0/0               0 2017-01-30 03:30 bin/bzless -> bzmore
-rwxr-xr-x 0/0            1297 2017-01-30 03:30 bin/bzmore
-rwxr-xr-x 0/0           35688 2017-02-22 21:23 bin/cat
-rwxr-xr-x 0/0           64424 2017-02-22 21:23 bin/chgrp
-rwxr-xr-x 0/0           60296 2017-02-22 21:23 bin/chmod
-rwxr-xr-x 0/0           64456 2017-02-22 21:23 bin/chown
```

Overlay 기반의 UFS가 아닌
표준 파일시스템 형식으로 저장됨

⑪ 파일 복사 효율을 위해 gzip으로 압축

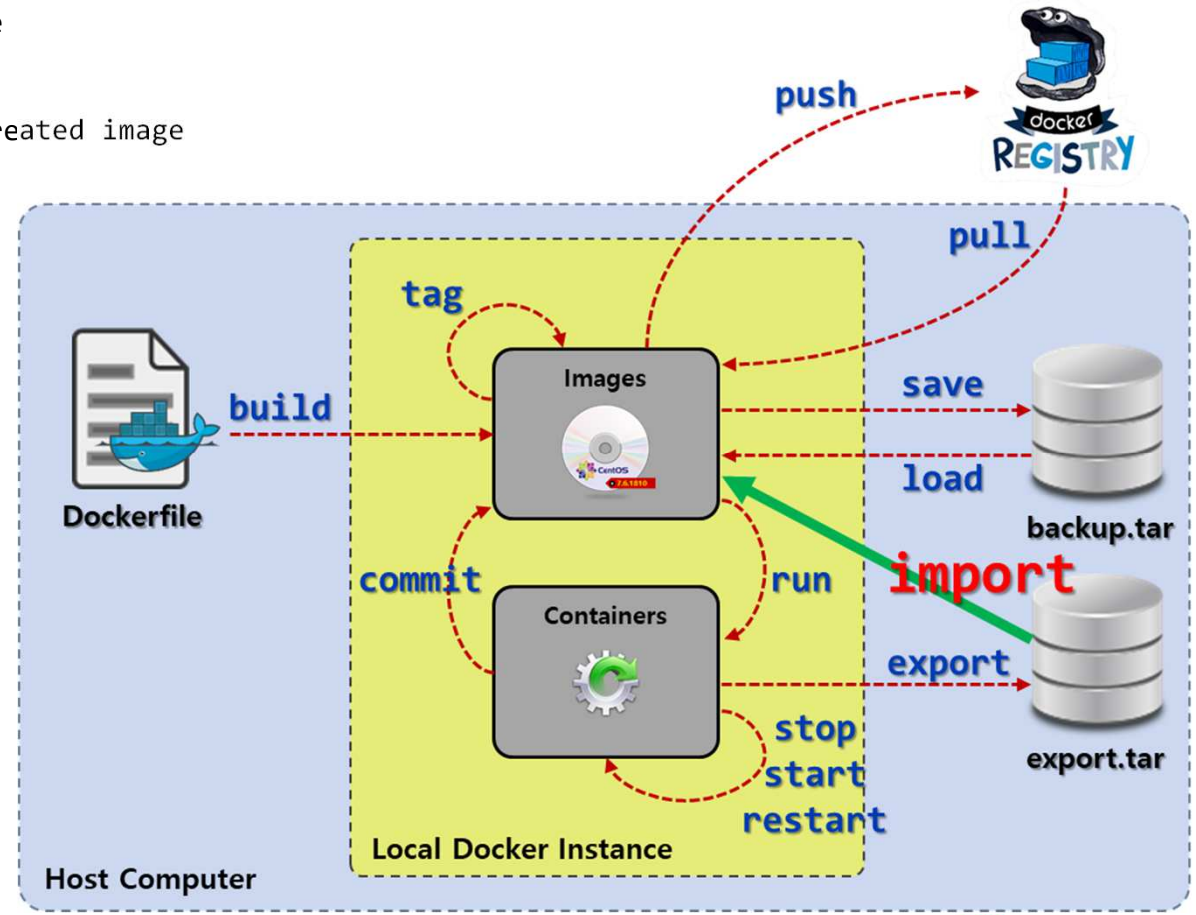
```
[root@dockeredu ~]# gzip guestbookH2_c01.tar
```

docker image import (1/2)

- **tar 파일로부터 도커 이미지 생성**

```
[root@dockeredu ~]# docker image import --help

Usage:  docker image import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

Import the contents from a tarball to create a filesystem image

Options:
  -c, --change list      Apply Dockerfile instruction to the created image
  -m, --message string   Set commit message for imported image
```

tar파일로부터 import한 이미지는 하나의 Layer만 존재함



docker image import (2/2)

- **LAB 2. tar 파일로부터 도커 이미지 생성**

① LAB1에서 export한 guestbookH2_c01.tar.gz 파일로부터 도커 이미지 생성 - <수강생ID>/guestbook_h2:1.0

```
[root@dockeredu ~]# docker image import guestbookH2_c01.tar.gz yu3papa/guestbook_h2:1.0
sha256:5551f4cbcde61f95ec9f4b5b57dd3f8f21b1431d66a98cf1f881fdd9ea1bb445
```

② import 된 이미지 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        6 seconds ago       640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
```

③ 강사가 제공해주는 URL의 tar 파일로 부터 도커 이미지 생성


```
[root@dockeredu ~]# docker image import http://jadecross.iptime.org:7778/dockeredu/guestbookH2_c01.tar.gz
yu3papa/guestbook_h2:2.0
Downloading from http://jadecross.iptime.org/owncloud/index.php/s/2ZK2hn7Cx47XeKR/download
sha256:6a01ab849604f0b218dcdef5dea4a9cd19b4debd77f3f367c59b0af8e5375ce7B/236.7MB
```

④ import 된 이미지 조회

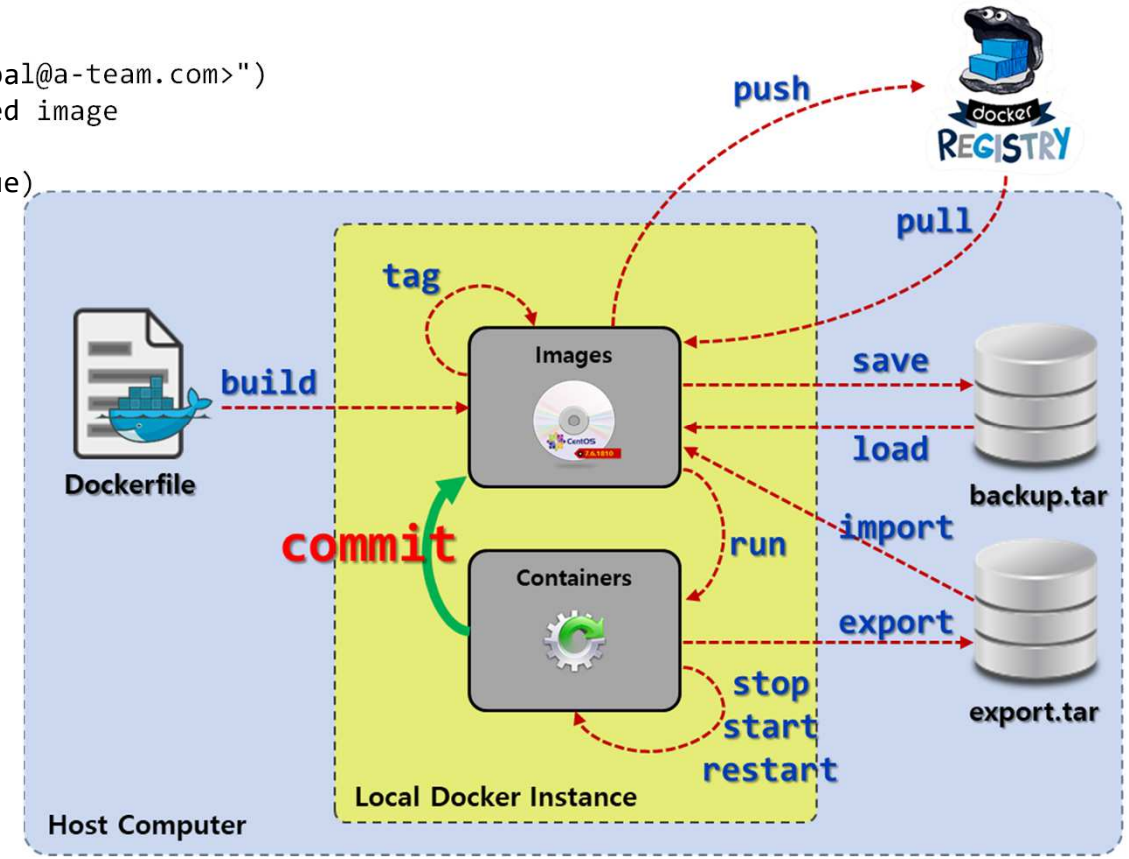
```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    2.0                 a2056ac008f7        10 seconds ago      640MB
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        13 minutes ago      640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
```

docker container commit (1/2)

- 컨테이너로부터 이미지 작성

```
# docker container commit --help

Usage:  docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

Create a new image from a container's changes

Options:
  -a, --author string    Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
  -c, --change list      Apply Dockerfile instruction to the created image
  -m, --message string   Commit message
  -p, --pause            Pause container during commit (default true)
```

컨테이너의 Writable Layer의 내용을
이미지로 저장하는 작업으로
새로운 Read-only Layer가 생성됨



docker container commit (2/2)

- **LAB 3. guestbook_h2:3.0 이미지 만들기**

Docker Hub에 PUSH를 고려하여 사용자ID 추가

① LAB1에서 생성한 guestbookH2_c01 컨테이너를 commit하여 새로운 **사용자ID**/guestbook_h2:3.0 이미지 작성

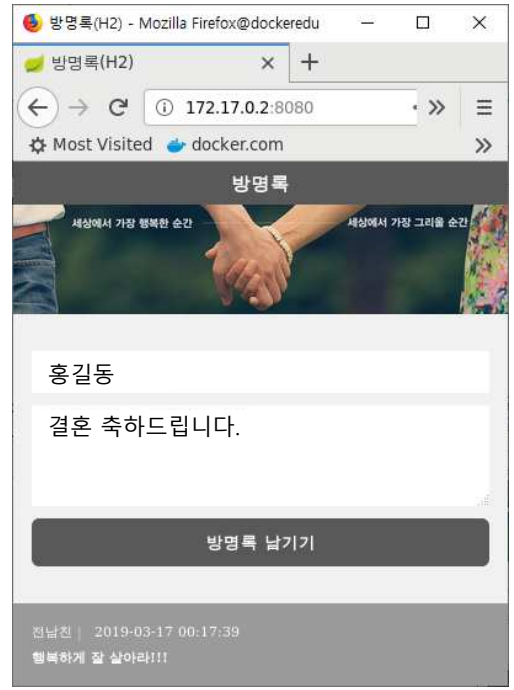
```
[root@dockeredu ~]# docker container commit -m "SpringBoot Guestbook(H2) created" guestbookH2_c01 yu3papa/guestbook_h2:3.0
sha256:8bdc605707b8a033bf79e29c57aa06c4944c4065e376b7bb0f834ccca828e31f
```

② 생성된 이미지 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    3.0                 17074d982872        18 seconds ago      648MB
yu3papa/guestbook_h2    2.0                 a2056ac008f7        12 minutes ago      640MB
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        26 minutes ago      640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
```

③ 생성된 guestbook_h2:3.0 이미지를 이용하여 방명록 컨테이너 시작 및 방명록 서비스 확인

```
[root@dockeredu ~]# docker container run yu3papa/guestbook_h2:3.0 java -jar /guestbook_H2.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-19 04:32:58.189  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : Starting GuestbookH2Application
v0.0.1-SNAPSHOT on c74e85307436 with PID 1 (/guestbook_H2.jar started by root in /)
```

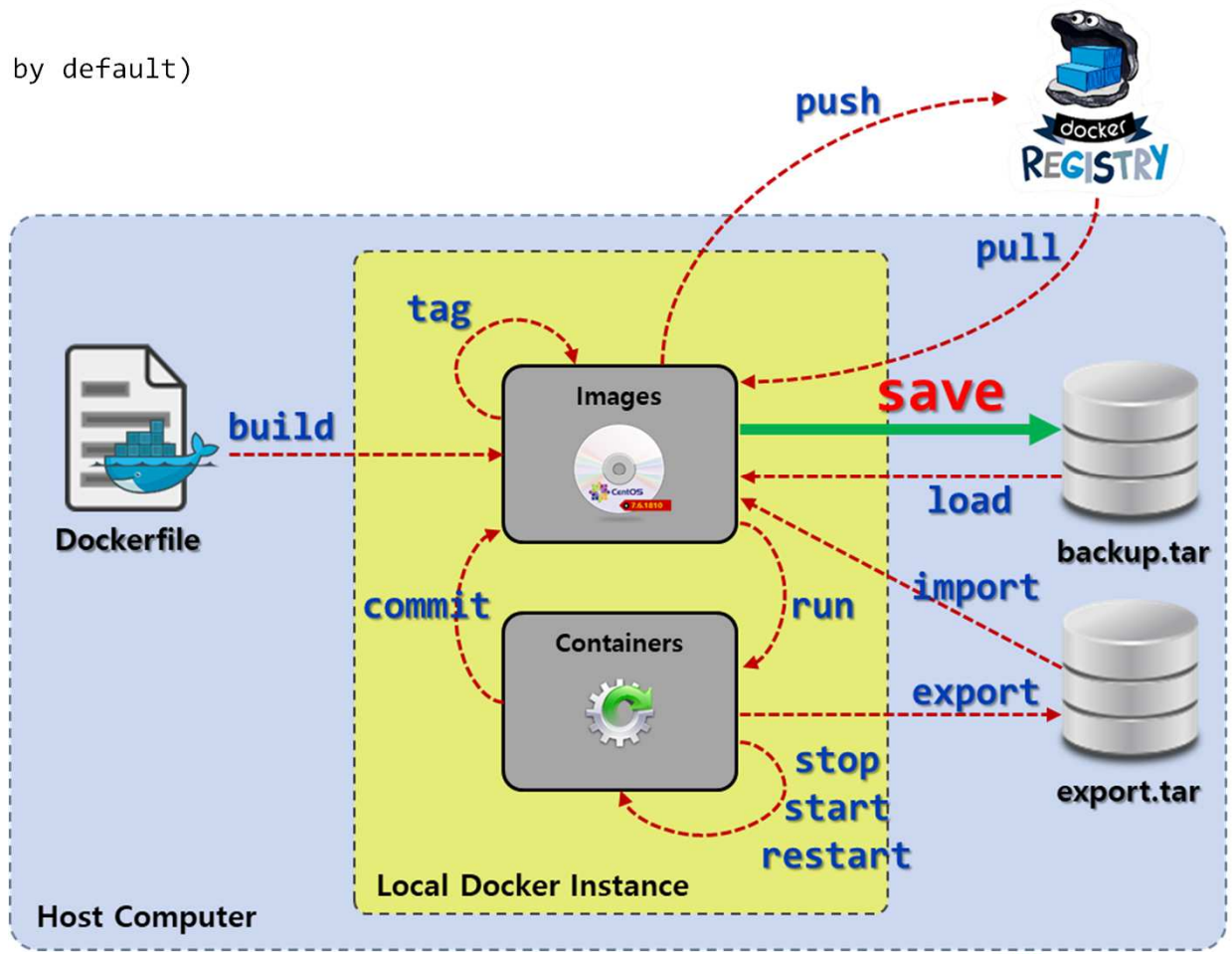


docker image save (1/2)

- 이미지 저장

```
[root@dockeredu ~]# docker image save --help

Usage:  docker image save [OPTIONS] IMAGE [IMAGE...]

Save one or more images to a tar archive (streamed to STDOUT by default)

Options:
  -o, --output string   Write to a file, instead of STDOUT
```

save 명령은 로컬에 있는
레파지토리의
전체 Layer를 tar파일로 저장



docker image save (2/2)

- **LAB 4. yu3papa/guestbook 레파지토리의 전체 이미지(1.0, 2.0, 3.0)를 tar파일로 저장**

① 지금까지 생성된 이미지 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    3.0                 17074d982872        7 minutes ago       648MB
yu3papa/guestbook_h2    2.0                 a2056ac008f7        19 minutes ago      640MB
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        33 minutes ago      640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
```

② LAB1~3에서 생성한 yu3papa/guestbook 레파지토리를 tar 파일로 저장

```
[root@dockeredu ~]# docker image save -o save.tar yu3papa/guestbook_h2
```

③ 저장된 tar 파일 내용 보기

UFS(Union File System) 형식의 OverlayFS 로 저장되어 있음

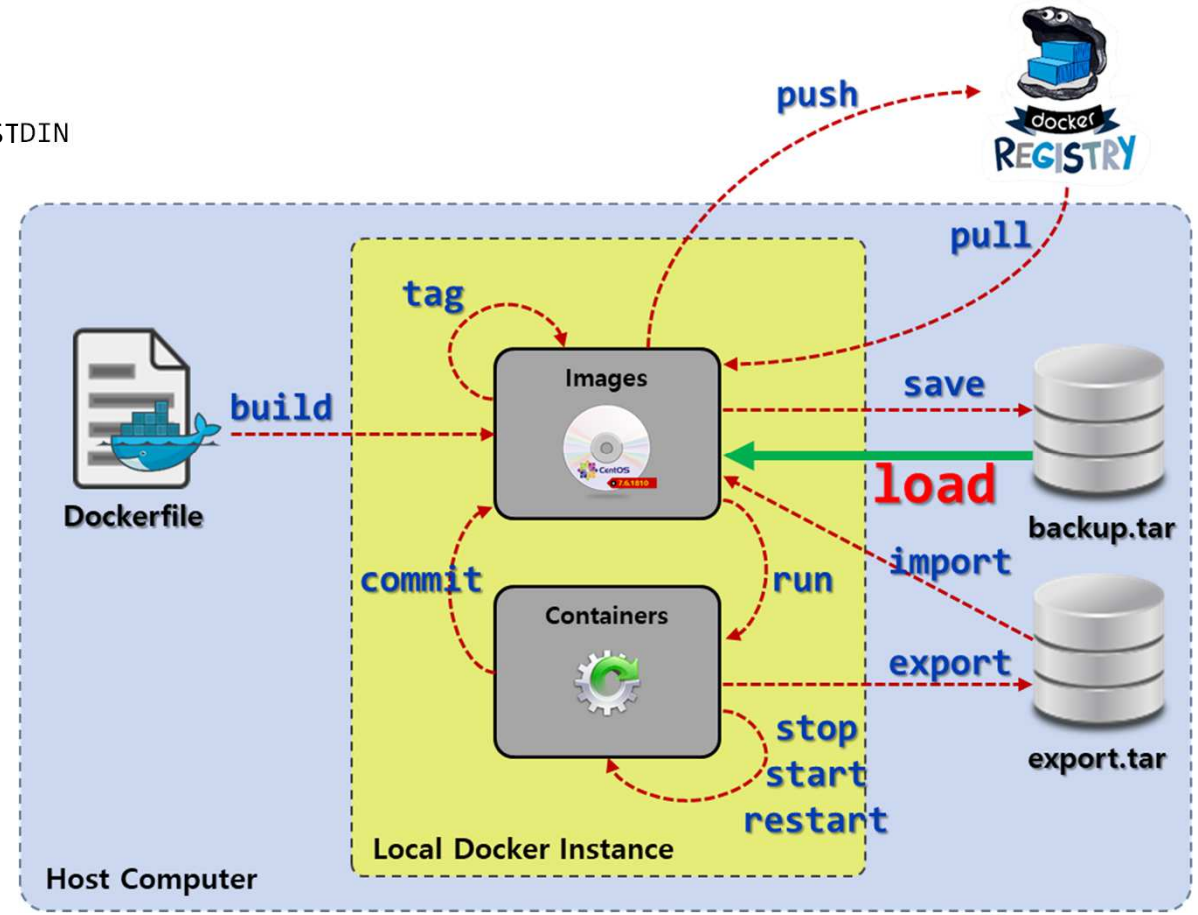
```
[root@dockeredu ~]# tar tvf save.tar | more
drwxr-xr-x 0/0               0 2019-04-19 18:12 098384f22c0305e668bda11b660469f3b11bf27ef9a8fe6a69cf725df3882b9a/
-rw-r--r-- 0/0               3 2019-04-19 18:12 098384f22c0305e668bda11b660469f3b11bf27ef9a8fe6a69cf725df3882b9a/VERSION
-rw-r--r-- 0/0             482 2019-04-19 18:12 098384f22c0305e668bda11b660469f3b11bf27ef9a8fe6a69cf725df3882b9a/json
-rw-r--r-- 0/0       357656064 2019-04-19 18:12 098384f22c0305e668bda11b660469f3b11bf27ef9a8fe6a69cf725df3882b9a/layer.tar
-rw-r--r-- 0/0            4828 2019-04-19 18:12 17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8.json
drwxr-xr-x 0/0               0 2019-04-19 17:46 225506dd5cc805b0d5c1973d34800a894e61f905f905ef2b7c50eaf9ecbad1f4/
-rw-r--r-- 0/0               3 2019-04-19 17:46 225506dd5cc805b0d5c1973d34800a894e61f905f905ef2b7c50eaf9ecbad1f4/VERSION
-rw-r--r-- 0/0             764 2019-04-19 17:46 225506dd5cc805b0d5c1973d34800a894e61f905f905ef2b7c50eaf9ecbad1f4/json
-rw-r--r-- 0/0       658262016 2019-04-19 17:46 225506dd5cc805b0d5c1973d34800a894e61f905f905ef2b7c50eaf9ecbad1f4/layer.tar
drwxr-xr-x 0/0               0 2019-04-19 18:12 26959e3d3d0f03f6289263c5b1aaa1721364eab29128d29f932ee592afbe9ab5/
-rw-r--r-- 0/0               3 2019-04-19 18:12 26959e3d3d0f03f6289263c5b1aaa1721364eab29128d29f932ee592afbe9ab5/VERSION
-rw-r--r-- 0/0            1307 2019-04-19 18:12 26959e3d3d0f03f6289263c5b1aaa1721364eab29128d29f932ee592afbe9ab5/json
-rw-r--r-- 0/0        22934528 2019-04-19 18:12 26959e3d3d0f03f6289263c5b1aaa1721364eab29128d29f932ee592afbe9ab5/layer.tar
```

docker image load (1/3)

- **이미지 저장**

```
[root@dockeredu ~]# docker image load --help

Usage:  docker image load [OPTIONS]

Load an image from a tar archive or STDIN

Options:
  -i, --input string   Read from tar archive file, instead of STDIN
  -q, --quiet          Suppress the load output
```

load 명령은 도커가 관리하는 overlay 방식으로 이미지의 전체 Layer를 포함하는 UFS 형식으로 이미지를 가져옴



docker image load (2/3)

- **LAB 5. save로 저장한 tar 파일로부터 로컬에 이미지를 로드 (1/2)**

① 지금까지 생성된 이미지 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    3.0                 17074d982872        7 minutes ago       648MB
yu3papa/guestbook_h2    2.0                 a2056ac008f7        19 minutes ago      640MB
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        33 minutes ago      640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
```

② yu3papa/guestbook_h2 레파지토리에 해당하는 모든 이미지(1.0, 2.0, 3.0)를 삭제 후 이미지 목록 조회

```
[root@dockeredu ~]# docker image rm yu3papa/guestbook_h2:1.0 yu3papa/guestbook_h2:2.0 yu3papa/guestbook_h2:3.0
Untagged: yu3papa/guestbook_h2:1.0
Deleted: sha256:5551f4cbcde61f95ec9f4b5b57dd3f8f21b1431d66a98cf1f881fdd9ea1bb445
Untagged: yu3papa/guestbook_h2:2.0
Deleted: sha256:a2056ac008f783fc8823283647a7790d05b1d0fc524c0de63c240db6f9c8b27f
Deleted: sha256:52b032dbf4bbdb5d1fd759bc59497289ea0df0410a963844f32b0efbfc8f622b
Untagged: yu3papa/guestbook_h2:3.0
Deleted: sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8
Deleted: sha256:90a8eea9c5bfe5f2f9159590841e92f9df59c27036bec626f0c1e5f87a208110

[root@dockeredu ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
openjdk             8                   b8d3f94869bb        3 weeks ago         625MB
```

해당 이미지의
모든 레이어가 삭제 됨

docker image load (3/3)

- **LAB 5. save로 저장한 tar 파일로부터 로컬에 이미지를 로드 (2/2)**

③ LAB4에서 save한 save.tar 파일로부터 yu3papa/guestbook_h2 이미지 로드

```
[root@dockeredu ~]# docker image load -i save.tar
52b032dbf4bb: Loading layer [==================================================>]  658.3MB/658.3MB
Loaded image: yu3papa/guestbook_h2:1.0
Loaded image: yu3papa/guestbook_h2:2.0
b8a659d8f838: Loading layer [==================================================>]  22.93MB/22.93MB
Loaded image: yu3papa/guestbook_h2:3.0
```

④ 로드된 이미지 목록 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY             TAG                 IMAGE_ID            CREATED             SIZE
yu3papa/guestbook_h2   3.0                 17074d982872        About an hour ago   648MB
yu3papa/guestbook_h2   2.0                 a2056ac008f7        2 hours ago         640MB
yu3papa/guestbook_h2   1.0                 5551f4cbcde6        2 hours ago         640MB
openjdk                8                   b8d3f94869bb        3 weeks ago         625MB
```

Image ID가 원본과 동일함

docker login

- **Docker Hub 개인 repository에 업로드 하려면 로그인이 필요함**

```
[root@dockeredu ~]# docker login --help

Usage:  docker login [OPTIONS] [SERVER]

Log in to a Docker registry

Options:
  -p, --password string   Password
      --password-stdin    Take the password from stdin
  -u, --username string   Username
```

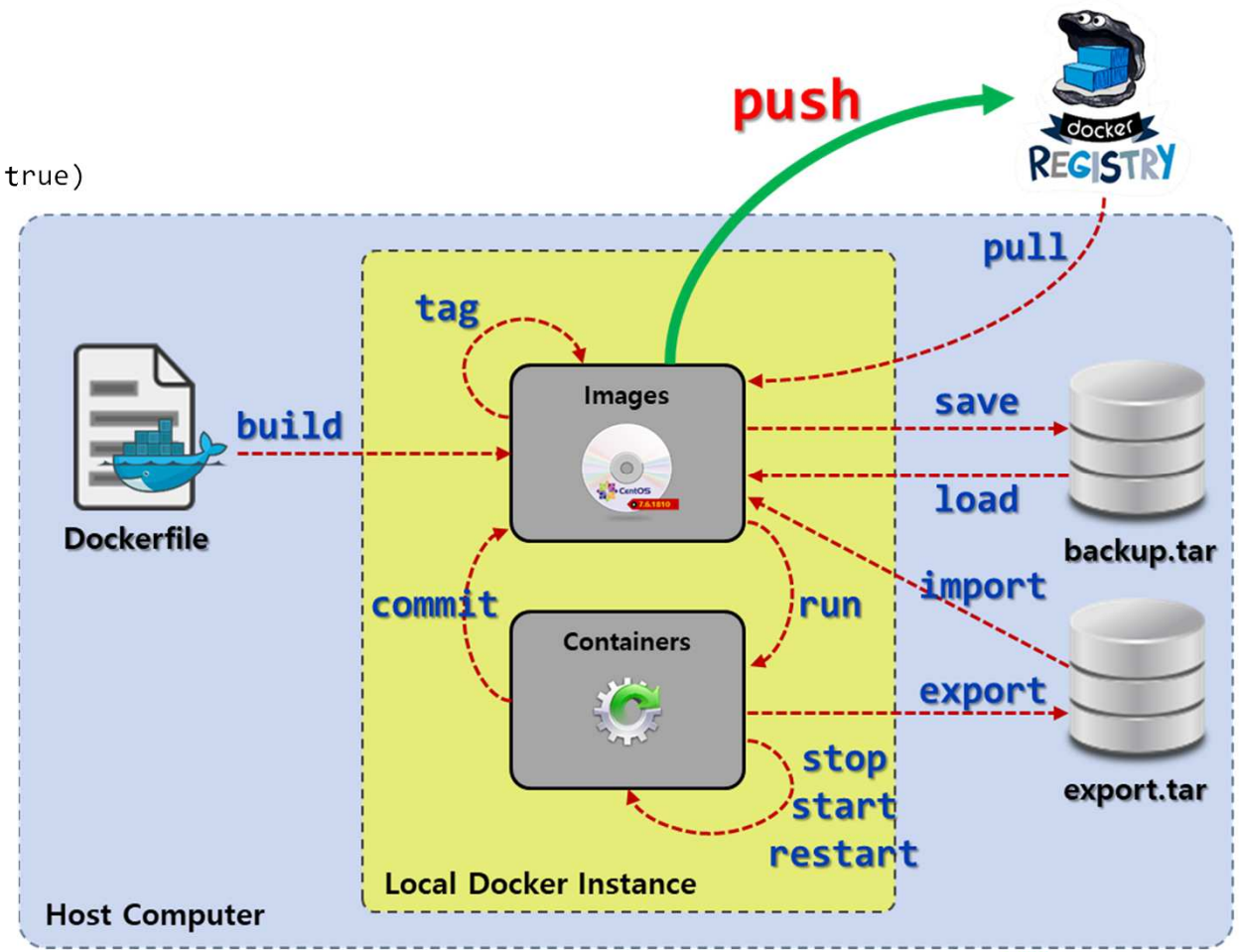
docker image push (1/3)

- **Docker Hub 레지스트리에 이미지 업로드**

```
[root@dockeredu ~]# docker image push --help

Usage:  docker image push [OPTIONS] NAME[:TAG]

Push an image or a repository to a registry

Options:
      --disable-content-trust   Skip image signing (default true)
```

- **Docker Hub 이미지명 규칙**
  - Docker Hub에 이미지를 push하기 위해서는 이미지명에 Docker Hub ID 를 넣어야 함

사용자ID/레파지토리:태그



docker image push (2/3)

- **LAB 6. 사용자ID/guestbook_h2 이미지를 Docker Hub에 업로드 (1/2)**

① Docker Hub에 로그인

```
[root@dockeredu ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to
https://hub.docker.com to create one.
Username: yu3papa
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

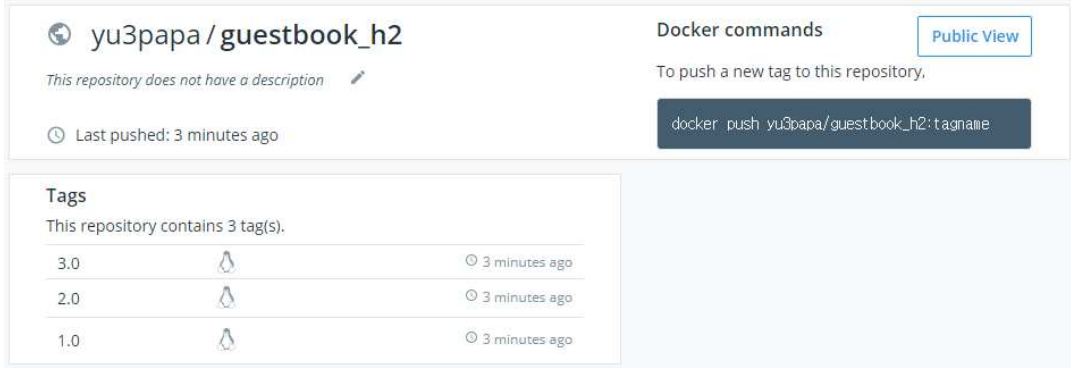
② 지금까지 생성한 yu3papa/guestbook_h2 이미지를 Docker Hub에 업로드(push)

```
[root@dockeredu ~]# docker image push yu3papa/guestbook_h2:3.0
The push refers to repository [docker.io/yu3papa/guestbook_h2]
52b032dbf4bb: Pushed
1.0: digest: sha256:3d4bceaa0505ea941aba0b404b3e4401dc8ed8effa3b5da896cd8e03c773d174 size: 529
52b032dbf4bb: Layer already exists
2.0: digest: sha256:d315d4591c33e8198234c0632353e8deb4255d43227670fbf1d093da09827c8c size: 529
b8a659d8f838: Pushed
f7d12d471667: Layer already exists
f350d0146bb3: Layer already exists
e38df31d449c: Layer already exists
af5ae4841776: Layer already exists
b17cc31e431b: Layer already exists
12cb127eee44: Layer already exists
604829a174eb: Layer already exists
fbb641a8b943: Layer already exists
3.0: digest: sha256:945a908956f29061ac2eca84fc1b309e0696b5d26569fa01a9d55b5dc5513cd4 size: 2213
```

docker image push (3/3)

- **LAB 6. 사용자ID/guestbook_h2 이미지를 Docker Hub에 업로드 (2/2)**

③ Docker Hub에서 push 된 이미지 확인



④ 특정 태그인 yu3papa/guestbook_h2:3.0 이미지를 Docker Hub에 업로드(push)

```
[root@dockeredu ~]# docker image push yu3papa/guestbook_h2:3.0
The push refers to repository [docker.io/yu3papa/guestbook_h2]
b8a659d8f838: Layer already exists
f7d12d471667: Layer already exists
f350d0146bb3: Layer already exists
e38df31d449c: Layer already exists
af5ae4841776: Layer already exists
b17cc31e431b: Layer already exists
12cb127eee44: Layer already exists
604829a174eb: Layer already exists
fbb641a8b943: Layer already exists
3.0: digest: sha256:945a908956f29061ac2eca84fc1b309e0696b5d26569fa01a9d55b5dc5513cd4 size: 2213
```

모든 Layer가 Docker Hub에 존재하여 업로드 하지 않음

docker image pull (1/2)

- **이미지 다운로드**
  - docker container run 명령 시 자동으로 실행됨
  - 이미지는 계층 구조를 가지며 Read Only
  - docker hub 사이트에서 기본 OS 이미지와 미리 구성된 애플리케이션 이미지를 다운로드

- **docker image pull –help**

```
[root@dockeredu ~]# docker image pull --help

Usage:  docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull an image or a repository from a registry

Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
```

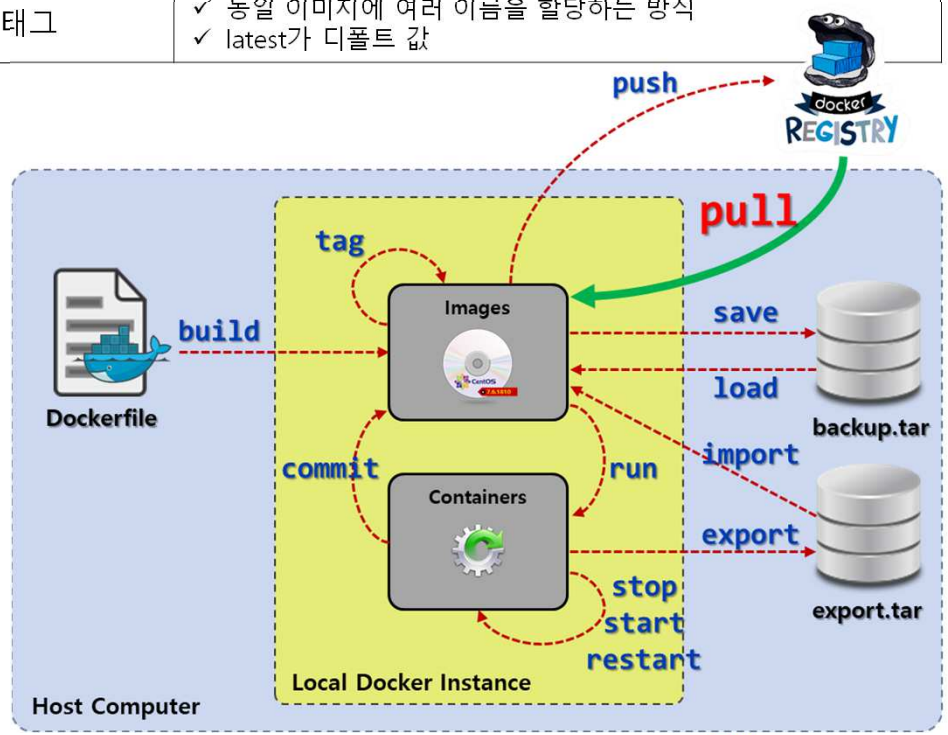
- **명령 예**
  - docker image pull centos ◀— *CentOS 최신 이미지 다운로드*
  - docker image pull centos:7 ◀— *CentOS 7 태그 이미지 다운로드*
  - docker image pull -a centos ◀— *CentOS의 모든 태그 이미지 다운로드*
  - docker image pull coordinatorj/debian-cowsay

coordinatoj 사용자의 debian-cowsay 이미지

- **이미지 이름 구성요소**

레지스트리주소/사용자ID/레파지토리:태그

구성요소	설명
레지스트리 주소	✓ 레지스트리 이름과 포트 ✓ 생략하면 디폴트로 docker hub 지칭
사용자 이름	✓ 레지스트리의 사용자 이름 ✓ 생략하면 docker hub에서 제공하는 공식 이미지
태그	✓ 동일 이미지에 여러 이름을 할당하는 방식 ✓ latest가 디폴트 값



docker image pull (2/2)

- **LAB 7. Docker Hub에 업로드 <사용자ID>/guestbook_h2:3.0 이미지를 로컬에 다운로드**

① LAB6 에서 Docker Hub에 push한 yu3papa/guestbook_h2:3.0 이미지를 로컬로 다운로드

```
[root@dockeredu ~]# docker image pull yu3papa/guestbook_h2:3.0
3.0: Pulling from yu3papa/guestbook_h2
Digest: sha256:945a908956f29061ac2eca84fc1b309e0696b5d26569fa01a9d55b5dc5513cd4
Status: Image is up to date for yu3papa/guestbook_h2:3.0
```

② Docker Hub에 push한 다른 사용자의 guestbook_h2:3.0 이미지를 로컬로 다운로드

```
[root@dockeredu ~]# docker image pull <someuser>/guestbook_h2:3.0
```

③ MySQL latest 이미지 다운로드

```
[root@dockeredu ~]# docker image pull mysql
Using default tag: latest
latest: Pulling from library/mysql
27833a3ba0a5: Pull complete
864c283b3c4b: Pull complete
cea281b2278b: Pull complete
8f856c14f5af: Pull complete
9c4f38c23b6f: Pull complete
1b810e1751b3: Pull complete
5479aaef3d30: Pull complete
ded8fa2e1614: Pull complete
636033ba4d2e: Pull complete
902e6010661d: Pull complete
dbe44d2bf055: Pull complete
e906385f419d: Pull complete
Digest: sha256:a7cf659a764732a27963429a87eccc8457e6d4af0ee9d5140a3b56e74986eed7
Status: Downloaded newer image for mysql:latest
```

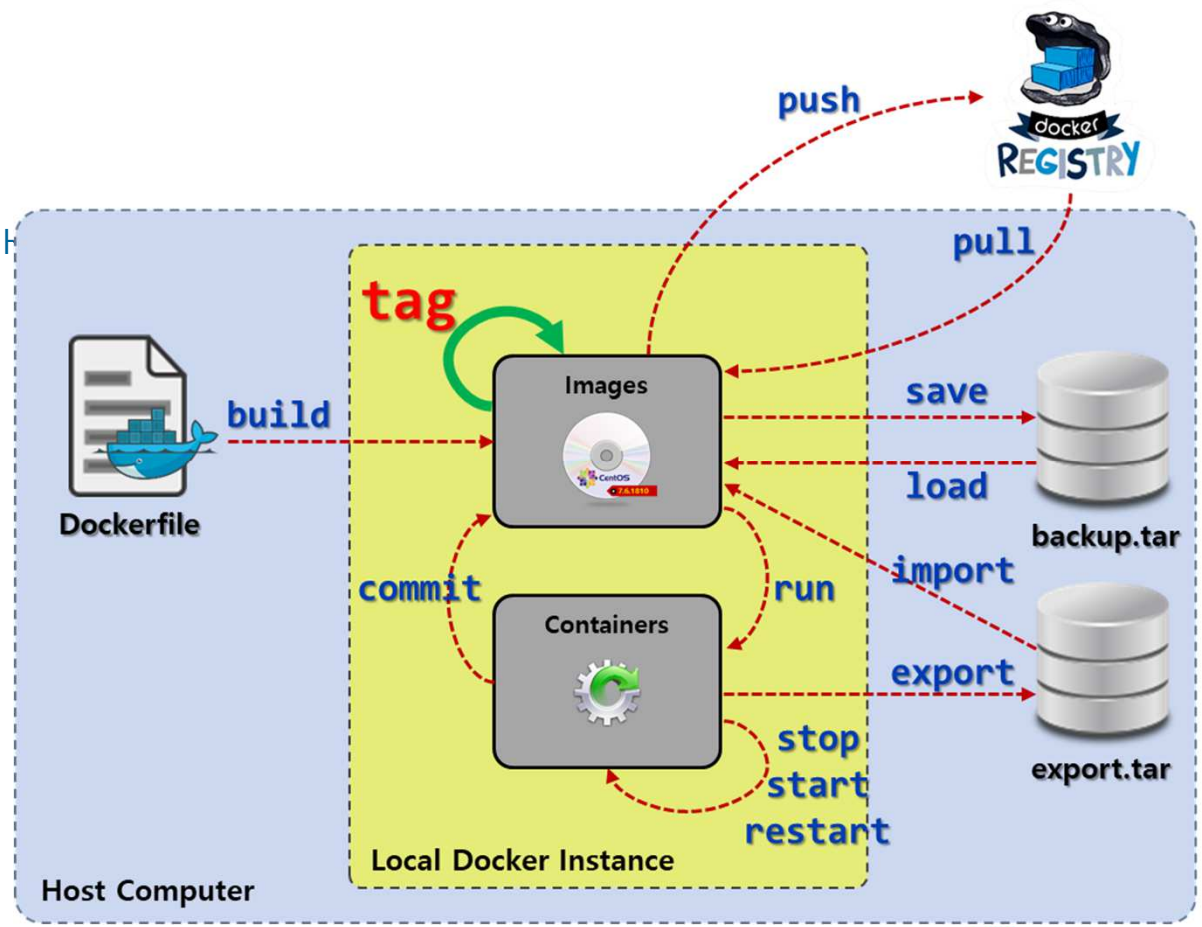
docker image tag (1/2)

- 이미지의 표식이 되는 태그를 붙여 새로운 이미지 버전 생성

```
[root@dockeredu ~]# docker image tag --help

Usage:  docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

- Docker Hub 이미지명 규칙
  - Docker Hub에 이미지를 push하기 위해서는 이미지명에 Docker H

사용자ID/레파지토리:태그



docker image tag (2/2)

- **LAB 8. yu3papa/guestbook_h2:3.0 이미지에서 4.0 태그를 갖는 이미지 생성**

① 로컬에 있는 이미지 목록 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2    3.0                 17074d982872        2 hours ago         648MB
yu3papa/guestbook_h2    2.0                 a2056ac008f7        3 hours ago         640MB
yu3papa/guestbook_h2    1.0                 5551f4cbcde6        3 hours ago         640MB
openjdk                 8                   b8d3f94869bb        3 weeks ago         625MB
mysql                   latest              7bb2586065cd        3 weeks ago         477MB
```

② yu3papa/guestbook_h2:3.0 이미지에서 4.0 태그를 갖는 이미지 생성

```
[root@dockeredu ~]# docker image tag yu3papa/guestbook_h2:3.0 yu3papa/guestbook_h2:4.0
```

③ 이미지 목록 조회

3.0과 4.0은 변경된 부분이 없기 때문에 IMAGE ID가 동일함

```
[root@dockeredu ~]# docker image ls --no-trunc
REPOSITORY              TAG                 IMAGE ID                                                                  CREATED             SIZE
yu3papa/guestbook_h2    3.0                 sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8   2 hours ago         648MB
yu3papa/guestbook_h2    4.0                 sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8   2 hours ago         648MB
yu3papa/guestbook_h2    2.0                 sha256:a2056ac008f783fc8823283647a7790d05b1d0fc524c0de63c240db6f9c8b27f   3 hours ago         640MB
yu3papa/guestbook_h2    1.0                 sha256:5551f4cbcde61f95ec9f4b5b57dd3f8f21b1431d66a98cf1f881fdd9ea1bb445   3 hours ago         640MB
openjdk                 8                   sha256:b8d3f94869bbbcf89afa5f3d4cb18b9e53dff94fb7200729796abc026600087f   3 weeks ago         625MB
mysql                   latest              sha256:7bb2586065cd50457e315a5dab0732a87c45c5fad619c017732f5a13e58b51dd   3 weeks ago         477MB
```

docker image ls

- **이미지 목록 조회**

```
[root@dockeredu ~]# docker image ls --help

Usage:  docker image ls [OPTIONS] [REPOSITORY[:TAG]]

List images

Aliases:
  ls, images, list

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

① 이미지 목록 조회 (Full Image ID)

```
[root@dockeredu ~]# docker image ls --no-trunc
REPOSITORY            TAG                 IMAGE ID                                                                  CREATED             SIZE
yu3papa/guestbook_h2  3.0                 sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8   2 hours ago         648MB
yu3papa/guestbook_h2  4.0                 sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8   2 hours ago         648MB
yu3papa/guestbook_h2  2.0                 sha256:a2056ac008f783fc8823283647a7790d05b1d0fc524c0de63c240db6f9c8b27f   3 hours ago         640MB
yu3papa/guestbook_h2  1.0                 sha256:5551f4cbcde61f95ec9f4b5b57dd3f8f21b1431d66a98cf1f881fdd9ea1bb445   3 hours ago         640MB
openjdk               8                   sha256:b8d3f94869bbbcf89afa5f3d4cb18b9e53dff94fb7200729796abc026600087f   3 weeks ago         625MB
mysql                 latest              sha256:7bb2586065cd50457e315a5dab0732a87c45c5fad619c017732f5a13e58b51dd   3 weeks ago         477MB
```

docker image history

- **이미지를 생성할 때 어떤 명령이 실행되는지를 확인**

```
[root@dockeredu ~]# docker image history --help

Usage:  docker image history [OPTIONS] IMAGE

Show the history of an image

Options:
      --format string   Pretty-print images using a Go template
  -H, --human           Print sizes and dates in human readable format (default true)
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
```

① yu3papa/guestbook_h2:3.0 이미지 히스토리 조회

```
[root@dockeredu ~]# docker image history yu3papa/guestbook_h2:3.0
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
17074d982872        3 hours ago         bash                                            22.9MB              SpringBoot Guestbook(H2) created
<missing>           3 weeks ago         /bin/sh -c set -ex;   if [ ! -d /usr/share/m…   349MB
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV JAVA_DEBIAN_VERSION=8…   0B
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV JAVA_VERSION=8u212       0B
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV JAVA_HOME=/docker-jav…   0B
<missing>           3 weeks ago         /bin/sh -c ln -svT "/usr/lib/jvm/java-8-open…   33B
<missing>           3 weeks ago         /bin/sh -c {   echo '#!/bin/sh';   echo 'set…   87B
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>           3 weeks ago         /bin/sh -c apt-get update && apt-get install…   2.21MB
<missing>           3 weeks ago         /bin/sh -c apt-get update && apt-get install…   142MB
<missing>           3 weeks ago         /bin/sh -c set -ex;  if ! command -v gpg > /…   7.81MB
<missing>           3 weeks ago         /bin/sh -c apt-get update && apt-get install…   23.2MB
<missing>           3 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           3 weeks ago         /bin/sh -c #(nop) ADD file:843b8a2a9df1a0730…   101MB
```

docker image inspect

- **이미지를 상세 정보 조회**

```
[root@dockeredu ~]# docker image inspect --help

Usage:  docker image inspect [OPTIONS] IMAGE [IMAGE...]

Display detailed information on one or more images

Options:
  -f, --format string   Format the output using the given Go template
```

① yu3papa/guestbook_h2:3.0 이미지 상세 정보 조회

```
[root@dockeredu ~]# docker image inspect yu3papa/guestbook_h2:3.0
[
    {
        "Id": "sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf8c8e945e8",
        "RepoTags": [
            "yu3papa/guestbook_h2:3.0",
            "yu3papa/guestbook_h2:4.0"
        ],
        "RepoDigests": [
            "yu3papa/guestbook_h2@sha256:945a908956f~~~01a9d55b5dc5513cd4"
        ],
        "Parent": "",
        "Comment": "SpringBoot Guestbook(H2) created",
        "Created": "2019-04-19T09:12:44.466224458Z",
        "Container": "ec8001694bb14de4451ed46250b5139ab8ecfabe0b62c227a89090a7cd2b7ffb",
        "ContainerConfig": {
            "Hostname": "ec8001694bb1",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
"AttachStdout": true,
            "AttachStderr": true,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "LANG=C.UTF-8",
                "JAVA_HOME=/docker-java-home",
                "JAVA_VERSION=8u212",
                "JAVA_DEBIAN_VERSION=8u212-b01-1~deb9u1"
            ],
            "Cmd": [
                "bash"
            ],
            "ArgsEscaped": true,
            "Image": "openjdk:8",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "DockerVersion": "18.06.2-ce",
```

컨테이너 1번 프로세스의 이해 (1/2)

docker image inspect yu3papa/guestbook_h2:3.0


```
[
    {
        "Id": "sha256:17074d9828722ccecd3e361d43e467c42f85133575ddc3e833395bf",
        "RepoTags": [
            "yu3papa/guestbook_h2:3.0",
            "yu3papa/guestbook_h2:4.0"
        ],
        "RepoDigests": [

"yu3papa/guestbook_h2@sha256:945a908956f29061ac2eca84fc1b26569fa01a9dc5513cd4"
        ],
        "Parent": "",
        "Comment": "SpringBoot Guestbook(H2) created",
        "Created": "2019-04-19T09:12:44.466224458Z",
        "Container": "ec8001694bb14de4451ed46250b5139ab8ecfabe0b62c227a890b7ffb",
        "ContainerConfig": {
            "Hostname": "ec8001694bb1",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
            "AttachStdout": true,
            "AttachStderr": true,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "LANG=C.UTF-8",
                "JAVA_HOME=/docker-java-home",
                "JAVA_VERSION=8u212",
                "JAVA_DEBIAN_VERSION=8u212-b01-1~deb9u1"
            ],
            "Cmd": [
                "bash"
            ],
            "ArgsEscaped": true,
            "Image": "openjdk:8",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "DockerVersion": "18.06.2-ce",
```


docker image inspect mysql:latest


```
[root@dockeredu ~]# docker image inspect mysql:latest
[
    {
        "Id": "sha256:7bb2586065cd50457e315a5dab0732a87c45c5fad619c5a13e58b51dd",
        "RepoTags": [
            "mysql:latest"
        ],
        "RepoDigests": [

"mysql@sha256:a7cf659a764732a27963429a87eccc8457e6d4af0ee9d5140a3b56e74986eed7"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2019-03-26T23:42:03.939173665Z",
        "Container": "24807a9771244f84a11db395fdd9c3331e4f8328855ef7a4fc369441",
        "ContainerConfig": {
            "Hostname": "24807a977124",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "3306/tcp": {},
                "33060/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "GOSU_VERSION=1.7",
                "MYSQL_MAJOR=8.0",
                "MYSQL_VERSION=8.0.15-1debian9"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"mysqld\"]"
            ],
            "ArgsEscaped": true,
            "Image": "sha256:fb28e3c8be89ca675d11ac80ea54d57ecb268e75e7a2010db",
            "Volumes": {
                "/var/lib/mysql": {}
            },
            "WorkingDir": "",
            "Entrypoint": [
                "docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {}
        },
        "DockerVersion": "18.06.1-ce",
```

컨테이너 1번 프로세스의 이해 (2/2)

- **shell form -vs- exec form**
  - 프로세스의 주체가 누구인가?

		shell form	exec form
형식		command param1 param2	[”executable”, ”param1”, ”param2”,]
CMD	형식	CMD command param1 param2	CMD [”executable”, ”param1”, ”param2”,]
	예	CMD nginx -g 'daemon off;'	CMD ["nginx", "-g", "daemon off;"]
	프로세스	/bin/sh을 통해 command가 실행됨	/bin/sh을 통해 executable이 프로세스로 실행됨
	특징	컨테이너 실행 시 command를 override 할 수 있음	
ENTRYPOINT	형식	ENTRYPOINT command param1 param2	ENTRYPOINT [”executable”, ”param1”, ”param2”,]
	예	ENTRYPOINT nginx -g 'daemon off;'	ENTRYPOINT ["nginx", "-g", "daemon off;"]
	프로세스	command가 프로세스로 실행됨	executable이 프로세스로 실행됨
	특징	컨테이너 실행 시 command를 override 할 수 없음 executable 위치에 shell-script를 사용할때는 exec 명령을 이용하여 1번 PID가 되도록 해야 함	

- **CMD와 ENTRYPOINT의 차이점**
  - CMD 명령의 경우는 컨테이너 시작 시에 실행하고 싶은 명령을 정의해도 docker container run 명령 실행 시에 인수로 새로운 명령을 지정한 경우 이것을 우선 실행하는 Override가 가능함
  - ENTRYPOINT 명령에서 지정한 명령은 반드시 컨테이너에서 실행되는데, 실행 시에 명령 인수를 지정하고 싶을 때는 CMD 명령과 조합하여 시용합니다. ENTRYPOINT 명령으로는 실행하고 싶은 명령 자체를 지정하고 CMD 명령으로는 그 명령의 인수를 지정하면 컨테이너를 실행했을 때의 기본 작동을 결정할 수 있습니다

docker image rm

- **이미지를 삭제**

```
[root@dockeredu ~]# docker image rm --help

Usage:  docker image rm [OPTIONS] IMAGE [IMAGE...]

Remove one or more images

Aliases:
  rm, rmi, remove

Options:
  -f, --force      Force removal of the image
      --no-prune   Do not delete untagged parents
```

① yu3papa/guestbook_h2:1.0 이미지 를 삭제한다.

```
[root@dockeredu ~]# docker image rm yu3papa/guestbook_h2:1.0
Untagged: yu3papa/guestbook_h2:1.0
Untagged: yu3papa/guestbook_h2@sha256:3d4bceaa0505ea941aba0b404b3e4401dc8ed8effa3b5da896cd8e03c773d174
Deleted: sha256:5551f4cbcde61f95ec9f4b5b57dd3f8f21b1431d66a98cf1f881fdd9ea1bb445
```

② 이미지 목록 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
yu3papa/guestbook_h2  3.0                 17074d982872        3 hours ago         648MB
yu3papa/guestbook_h2  4.0                 17074d982872        3 hours ago         648MB
yu3papa/guestbook_h2  2.0                 a2056ac008f7        3 hours ago         640MB
openjdk               8                   b8d3f94869bb        3 weeks ago         625MB
mysql                 latest              7bb2586065cd        3 weeks ago         477MB
```

docker image prune

- **컨테이너에서 사용되지 않는 이미지를 일괄 삭제**

```
[root@dockeredu ~]# docker image prune --help

Usage:  docker image prune [OPTIONS]

Remove unused images

Options:
  -a, --all             Remove all unused images, not just dangling ones
      --filter filter   Provide filter values (e.g. 'until=<timestamp>')
  -f, --force           Do not prompt for confirmation
```

① 이미지 중에서 컨테이너에서 사용되지 않는 이미지를 전부 삭제

```
[root@dockeredu ~]# docker image prune -a
WARNING! This will remove all images without at least one container associated to them.
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: openjdk:8
untagged: openjdk@sha256:96331221311fc8db41436b3ec3f7606e44ba1d28e2a7910f9f8883e8afce9fa2
deleted: sha256:b8d3f94869bbbcf89afa5f3d4cb18b9e53dff94fb7200729796abc026600087f
untagged: yu3papa/guestbook_h2:2.0
untagged: yu3papa/guestbook_h2@sha256:d315d4591c33e8198234c0632353e8deb4255d43227670fbf1d093da09827c8c
deleted: sha256:a2056ac008f783fc8823283647a7790d05b1d0fc524c0de63c240db6f9c8b27f
deleted: sha256:52b032dbf4bbdb5d1fd759bc59497289ea0df0410a963844f32b0efbfc8f622b
~~~

Total reclaimed space: 1.765GB
```

② 이미지 목록 조회

```
[root@dockeredu ~]# docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
```

docker search

- **docker hub에서 이미지 검색**
  - Usage: docker search [OPTIONS] TERM

옵션 예시	설명
--filter=stars=10	✓ 별점을 10개 이상 받은 이미지 검색
--filter=is-automated=true	✓ 자동으로 재구성되는 이미지만 검색
--no-trunc=true	✓ 이미지 설명을 자르지 않고 전체 출력

- **명령 예시**

```
# docker search tomcat
# docker search --filter=stars=10 tomcat
# docker serach --filter=is-automated=true tomcat
# docker search --no-trunc=true tomcat
```

```
[root@dockeredu ~]# docker search --filter=stars=10 tomcat
NAME                                DESCRIPTION                                     STARS     OFFICIAL    AUTOMATED
tomcat                              Apache Tomcat is an open source implementati…   2349      [OK]
tomee                               Apache TomEE is an all-Apache Java EE certif…   65        [OK]
dordoka/tomcat                      Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 base…   53                    [OK]
davidcaste/alpine-tomcat            Apache Tomcat 7/8 using Oracle Java 7/8 with…   34                    [OK]
bitnami/tomcat                      Bitnami Tomcat Docker Image                     28                    [OK]
cloudesire/tomcat                   Tomcat server, 6/7/8                            14                    [OK]
meirwa/spring-boot-tomcat-mysql-app a sample spring-boot app using tomcat and My…   12                    [OK]
aallam/tomcat-mysql                 Debian, Oracle JDK, Tomcat & MySQL              11                    [OK]
tutum/tomcat                        Base docker image to run a Tomcat applicatio…   11
```

Docker Image를 이용한 프라이빗 레지스트리 구성 (1/2)

- **registry**
  - https://hub.docker.com/_/registry
  - https://docs.docker.com/registry/

registry ☆

Docker Official Images

The Docker Registry 2.0 implementation for storing and distributing Docker images

① Dokcer Hub에서 registry:2.7 이미지를 이용하여 registry 컨테이너 실행

```
[root@dockeredu ~]# docker container run -d --name registry --restart always -p 5000:5000 registry:2.7
Unable to find image 'registry:2.7' locally
2.7: Pulling from library/registry
c87736221ed0: Pull complete
1cc8e0bb44df: Pull complete
54d33bcb37f5: Pull complete
e8afc091c171: Pull complete
b4541f6d3db6: Pull complete
Digest: sha256:3b00e5438ebd8835bcfa7bf5246445a6b57b9a50473e89c02ecc8e575be3ebb5
Status: Downloaded newer image for registry:2.7
d5eec830bdab8799aeed4dbd5c907d5814894fbc4ee878648c012bb47093b8f0
```

② Docker Hub에서 haproxy:1.9 이미지를 pull

```
[root@dockeredu ~]# docker image pull haproxy:1.9
1.9: Pulling from library/haproxy
27833a3ba0a5: Pull complete
7efa30c54e1d: Pull complete
c31a2866b5f9: Pull complete
Digest: sha256:6dae9c8674e2e5f418c3dd040041a05f6b490597315139c0bcacadf65a46cfd5
Status: Downloaded newer image for haproxy:1.9
```

Docker Image를 이용한 프라이빗 레지스트리 구성 (2/2)

③ localhost:5000/<사용자ID>/myhaproxy:1.9 로 태그 생성하고 image 목록 조회

```
[root@dockeredu ~]# docker image tag haproxy:1.9 localhost:5000/yu3papa/myhaproxy:1.9
[root@dockeredu ~]# docker image ls
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
localhost:5000/yu3papa/myhaproxy    1.9                 11fa4d7ff427        5 days ago          72.2MB
haproxy                             1.9                 11fa4d7ff427        5 days ago          72.2MB
registry                            2.7                 f32a97de94e1        7 weeks ago         25.8MB
```

④ localhost:5000 프라이빗 레지스트리에 myhaproxy:1.9 이미지 push

```
[root@dockeredu ~]# docker image push localhost:5000/yu3papa/myhaproxy:1.9
~~~
5dacd731af1b: Pushed
1.9: digest: sha256:45e78073f43b255b1d77442195f7bd4f13c2cce3e653159f7a3afdde9773b505 size: 947
```

⑤ image push가 정상적인지 registry catalog 조회

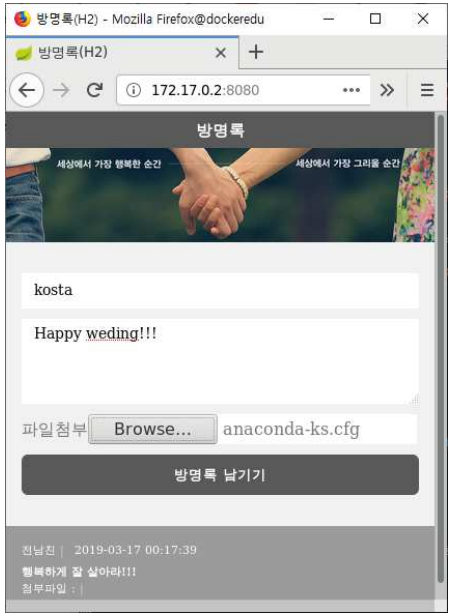
```
[root@dockeredu ~]# curl localhost:5000/v2/_catalog
{"repositories":["yu3papa/myhaproxy"]}
```

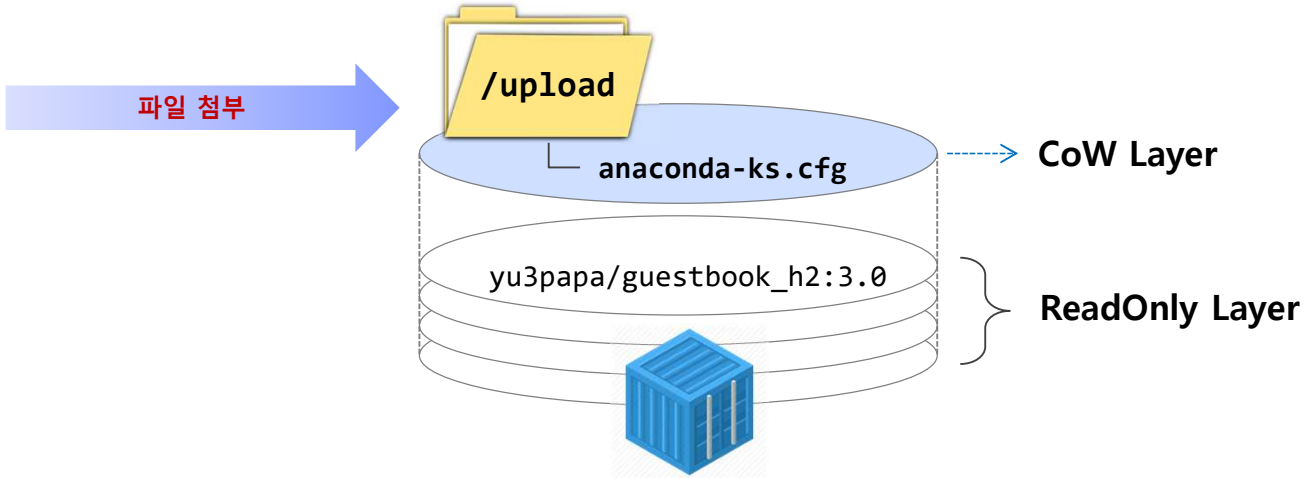
⑥ 로컬에 저장된 localhost:5000/yu3papa/myhaproxy:1.9 이미지를 삭제하고 프라이빗 레지스트리에서 다시 pull

```
[root@dockeredu ~]# docker image rm localhost:5000/yu3papa/myhaproxy:1.9
Untagged: localhost:5000/yu3papa/myhaproxy:1.9
Untagged: localhost:5000/yu3papa/myhaproxy@sha256:45e78073f43b255b1d77442195f7bd4f13c2cce3e653159f7a3afdde9773b505
[ [root@dockeredu ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
haproxy             1.9                 11fa4d7ff427        5 days ago          72.2MB
registry            2.7                 f32a97de94e1        7 weeks ago         25.8MB
[root@dockeredu ~]# docker image pull localhost:5000/yu3papa/myhaproxy:1.9
1.9: Pulling from yu3papa/myhaproxy
Digest: sha256:45e78073f43b255b1d77442195f7bd4f13c2cce3e653159f7a3afdde9773b505
Status: Downloaded newer image for localhost:5000/yu3papa/myhaproxy:1.9
[root@dockeredu ~]# docker image ls
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
haproxy                             1.9                 11fa4d7ff427        5 days ago          72.2MB
localhost:5000/yu3papa/myhaproxy    1.9                 11fa4d7ff427        5 days ago          72.2MB
registry                            2.7                 f32a97de94e1        7 weeks ago         25.8MB
```

도커 볼륨

컨테이너에서 생성한 파일의 영속성을 보장할 수 없을까?



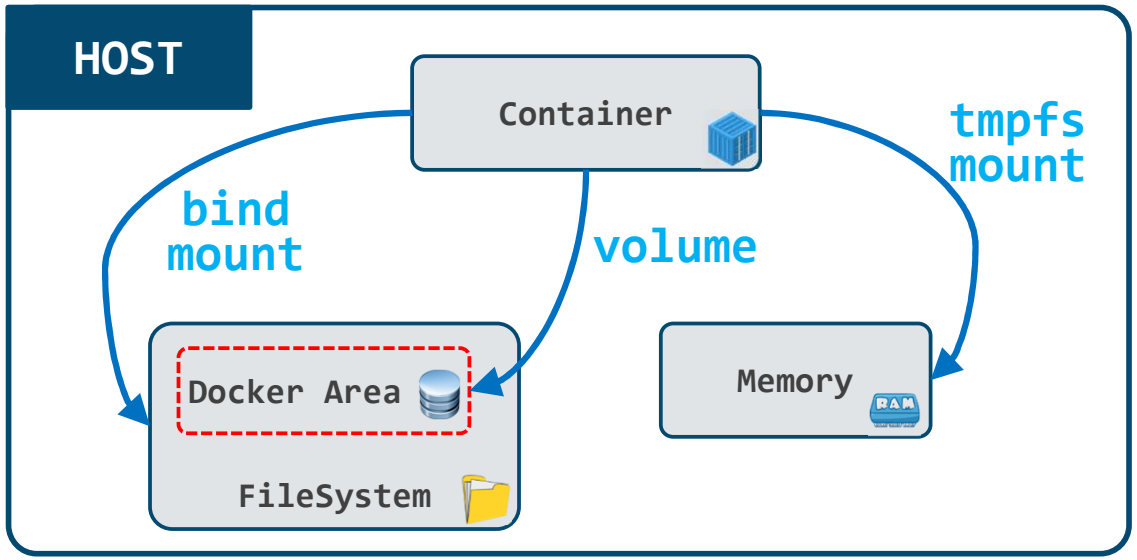


컨테이너가 삭제되거나 비정상일 때도 업로드 한 첨부파일을 안전하게 영구히 보존하는 방법은?

컨테이너 CoW Layer에 저장된 파일의 특성

- **컨테이너안에서 생성한 파일들은 디폴트로 CoW(Copy-on-Write)레이어에 저장됨**
  - 해당 컨테이너가 더 이상 존재하지 않으면 데이터가 지속되지 않으며 다른 프로세스에서 필요하면 컨테이너에서 데이터를 가져 오는 것이 어려울 수 있습니다.
  - 컨테이너의 쓰기 가능 계층은 컨테이너가 실행중인 호스트 시스템과 밀접하게 연결됩니다. 데이터를 다른 곳으로 쉽게 이동할 수 없습니다.
  - 컨테이너의 쓰기 가능 계층에 쓰기 작업을 수행하려면 파일 시스템을 관리하는 저장소 드라이버가 필요합니다. 스토리지 드라이버는 Linux 커널을 사용하여 공용 파일 시스템(Union File System)을 제공합니다. 이 여분의 추상화는 호스트 파일 시스템에 직접 쓰는 데이터 볼륨을 사용하는 것과 비교하여 성능을 저하시킵니다.

- **Docker가 지원하는 데이터 영속성 방법**

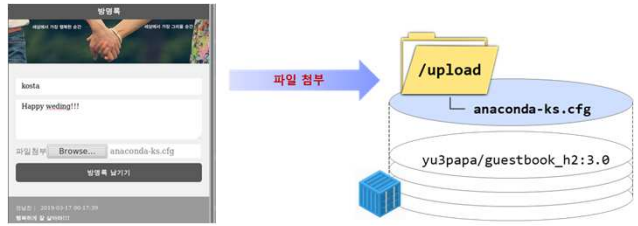


종류	설명
bind mount	✓ 호스트의 디렉토리에 마운트 ✓ 컨테이너 런타임때 호스트의 볼륨을 컨테이너에 마운트 ✓ HOST 파일시스템에서 파일 수정 가능
volume	✓ Docker 가 관리하는 호스트 파일 시스템(/var/lib/docker/volumes/)에 저장 ✓ Non-Docker 프로세스는 파일 시스템의 이 부분을 수정해서는 안 됩니다. ✓ 볼륨은 Docker에서 데이터를 유지하는 가장 좋은 방법입니다.
tmpfs mount	✓ 호스트 시스템의 메모리에만 저장 ✓ 호스트 시스템의 파일 시스템에는 저장되지 않음

HOST시스템에 저장되는 컨테이너의 디폴트 스토리지 위치

① yu3papa/guestbook_h2:3.0 이미지로부터 c01 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run --name=c01 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
```



② docker inspect 명령을 이용하여 c01 컨테이너의 파일시스템 레이어 확인

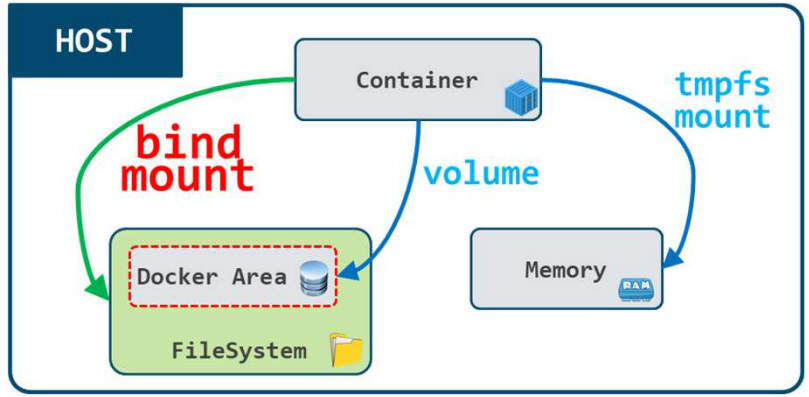
```
[root@dockeredu ~]# docker container inspect c01 | grep -A 8 GraphDriver
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/6ba2c3d4f1e77bc5478675c2442ba26f57189d6ca1e8dee748a9291b9aae9954-
init/diff:/var/lib/docker/overlay2/6e117233f16c55c2b94d13a28cd339f1c39b64484afc694a94246df8c7684f95/diff:/var/lib/docker/overlay2/7d0d009eede8b
66f64f1059ab03cb1ebf249c42c453b9d4005d941c85f6e5c42/diff:/var/lib/docker/overlay2/bd337cde307dc36d2739ce4fed86208bd4278c60d6ad96656f2aded94c5c5
239/diff:/var/lib/docker/overlay2/c366849cde34d909399615b77b26fb367b3565c73d12c61a9f6eafaa6355e9c8/diff:/var/lib/docker/overlay2/a11050294fd6e5
b3f583395d9aeff4b0f2020f2895ac224dcec152bec5400f78/diff:/var/lib/docker/overlay2/44d15ac7a78f0ed8a83f37d01b6ea2537b24e2b2bda3045a86cc7e50f1f191
13/diff:/var/lib/docker/overlay2/945401bebe28b40ff9fb6b8c1224b5c1b07a913571406d14b24f9f2754ab4497/diff:/var/lib/docker/overlay2/d4ecad8aa50f2d8
9649a4c553b776a6fb6b73a5db7f96a03f3ec1a7454f5cc40/diff:/var/lib/docker/overlay2/46de1a13d891bb2aecde95dad8e54ace5654641ffa05d0639d216a430082524
3/diff",
                "MergedDir": "/var/lib/docker/overlay2/6ba2c3d4f1e77bc5478675c2442ba26f57189d6ca1e8dee748a9291b9aae9954/merged",
                "UpperDir": "/var/lib/docker/overlay2/6ba2c3d4f1e77bc5478675c2442ba26f57189d6ca1e8dee748a9291b9aae9954/diff",
                "WorkDir": "/var/lib/docker/overlay2/6ba2c3d4f1e77bc5478675c2442ba26f57189d6ca1e8dee748a9291b9aae9954/work"
            },
            "Name": "overlay2"
        },
```

③ "UpperDir" 값 하위의 upload 폴더 파일 조회

```
[root@dockeredu ~]# ls -l /var/lib/docker/overlay2/6ba2c3d4f1e77bc5478675c2442ba26f57189d6ca1e8dee748a9291b9aae9954/diff/upload
total 4
-rw-r--r-- 1 root root 1443 Apr 21 14:05 20190421050539468_edu_anaconda-ks.cfg
```

bind mount (1/7)

- **호스트의 볼륨 공유**
  - docker container run에 -v 옵션 사용
    - **-v <호스트 경로>:<컨테이너 마운트 경로>**
    - -v <호스트 경로>:<컨테이너 마운트 경로>:<읽기 쓰기 모드>

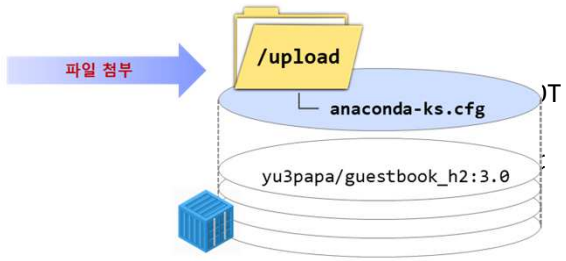


① HOST 시스템에서 /tmp/updir 폴더 생성

```
[root@dockeredu ~]# mkdir /tmp/updir
```

② yu3papa/guestbook_h2:3.0 이미지로부터 -v /tmp/updir:/upload 옵션을 적용하여 bindmount01컨테이너를 시작하고 HOST시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run -v /tmp/updir:/upload --name=bindmount01 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar

.   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-21 05:39:06.207  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Applicati
on 6d0c6fb91fe1 with PID 1 (/guestbook_H2.jar started by root in /)
2019-04-21 05:39:06.211  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Applicati
profiles: default
2019-04-21 05:39:07.850  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebS
```



bind mount (2/7)

③ docker inspect 명령을 이용하여 bindmount01 컨테이너의 파일시스템 레이어 + 마운트 확인

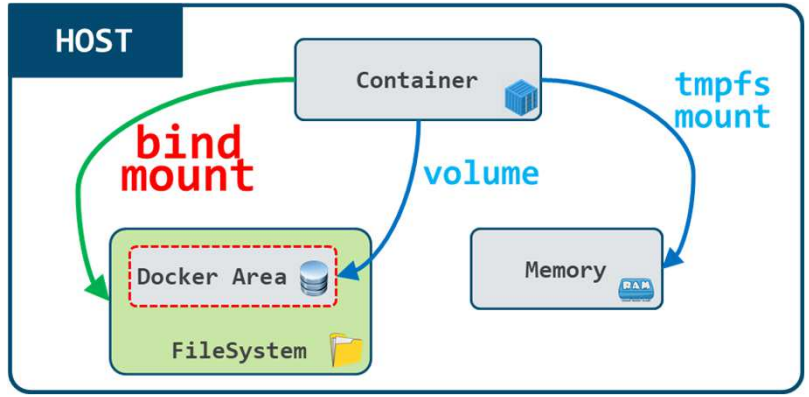
```
[root@dockeredu ~]# docker container inspect bindmount01 | grep -A 10 \"Mounts\"
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/tmp/updir",
                "Destination": "/upload",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

④ HOST 시스템에서 "Mount.Source" 값에 해당하는 /tmp/updir 폴더 파일 조회

```
[root@dockeredu ~]# ls -l /tmp/updir
total 4
-rw-r--r-- 1 root root 1443 Apr 21 15:16 20190421061620520_edu_anaconda-ks.cfg
```

bind mount (3/7)

- **호스트의 볼륨 공유**
  - docker container run에 -v 옵션 사용
    - -v <호스트 경로>:<컨테이너 마운트 경로>
    - **-v <호스트 경로>:<컨테이너 마운트 경로>:<읽기 쓰기 모드>**



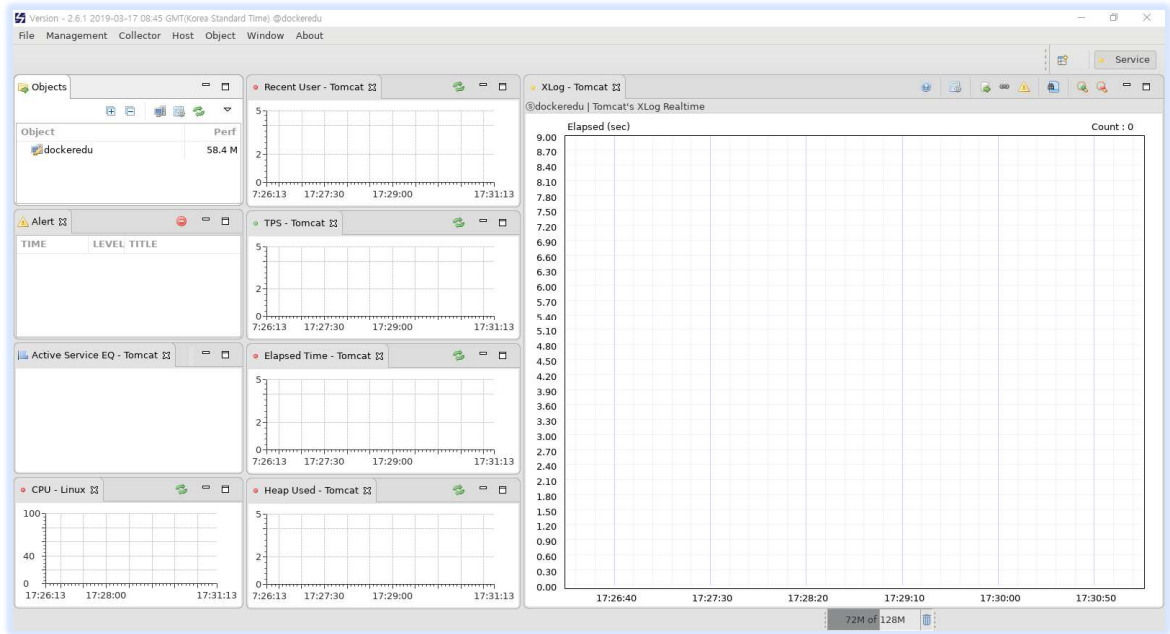
*** HOST 시스템의 2개 폴더 마운트 LAB**

① 스카우터 서버 시작

```
[root@dockeredu ~]# scouter.server.boot
nohup: redirecting stderr to stdout

  ____                  _
 / ___|  ___ ___  _   _| |_ ___ _ __
 \___ \ / __/   \| | | | __/ _ \ '__|
  ___) | (_| (+) | |_| | ||  __/ |
 |____/ \___\___/ \__,_|\__\___|_|
 Open Source S/W Performance Monitoring
 Scouter version 2.6.1
```

② 스카우터 클라이언트 시작

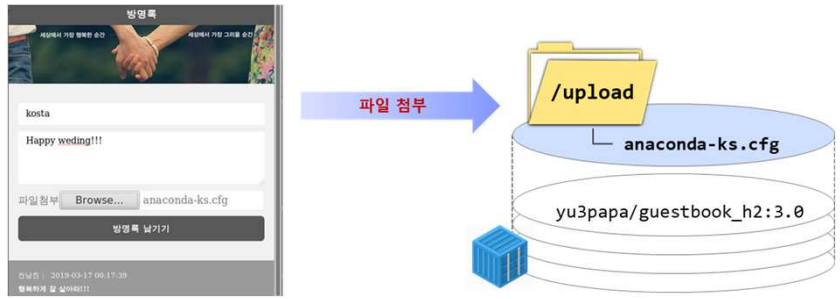
```
[root@dockeredu server]# scouter.client
[1] 11929
```



bind mount (4/7)

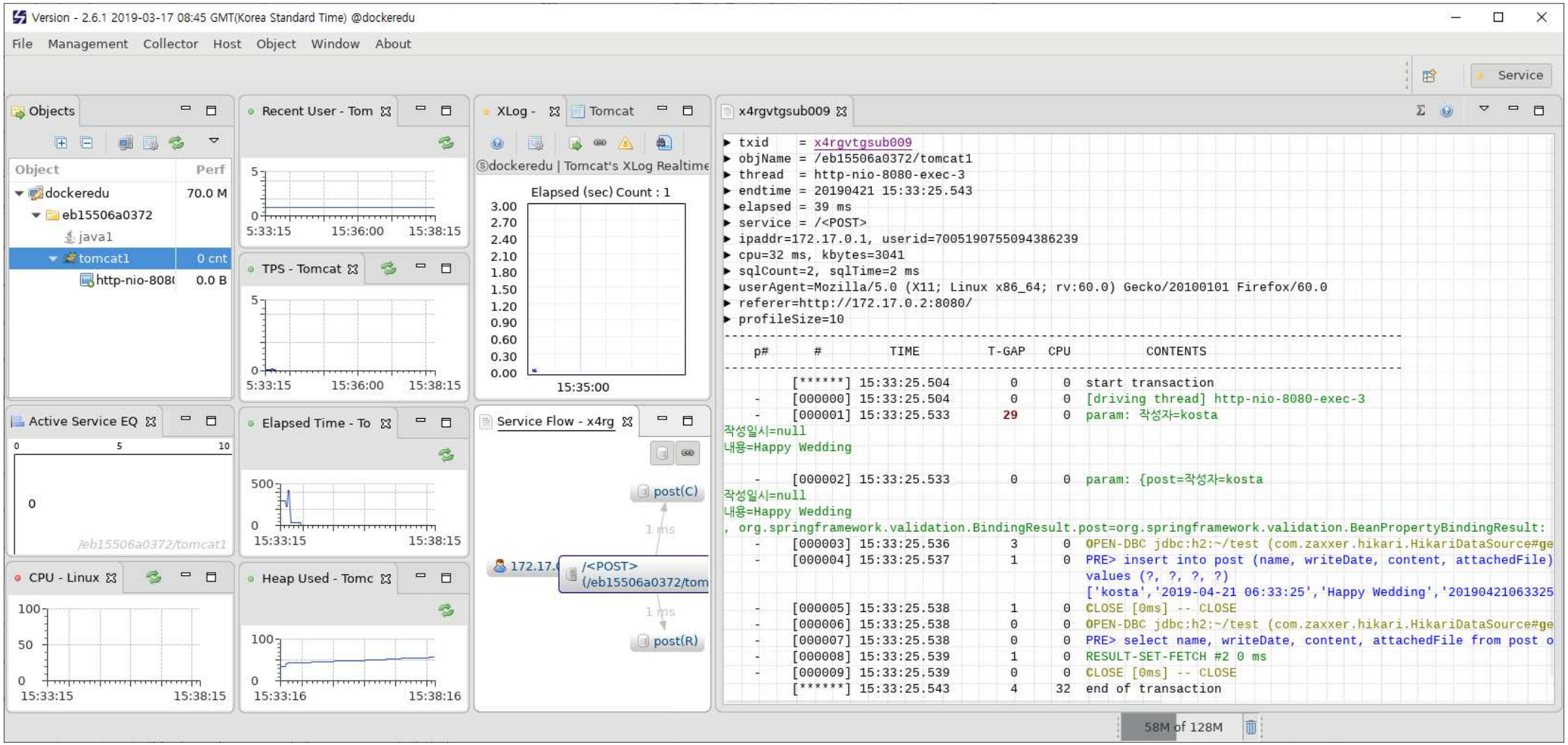
③ yu3papa/guestbook_h2:3.0 이미지로부터 -v /app/scouter/agent.java:/scouter:ro -v /tmp/updir:/upload 옵션을 적용하여 bindmount02 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run -v /app/scouter/agent.java:/scouter:ro -v /tmp/updir:/upload --name=bindmount02
yu3papa/guestbook_h2:3.0 java -javaagent:/scouter/scouter.agent.jar -jar guestbook_H2.jar

 / ___|  ___ ___  _   _| |_ ___ _ __
 \___ \ / __/   \| | | | __/ _ \ '__|
  ___) | (_| (+) | |_| | ||  __/ |
 |____/ \___\___/ \__,_|\__\___|_|
 Open Source S/W Performance Monitoring
 Scouter version 2.6.1

20190421 06:32:22 [SCOUTER] Version 2.6.1 2019-03-17 08:45 GMT_ENV_java8plus
20190421 06:32:22 [SCOUTER] loaded by system classloader
20190421 06:32:22 [SCOUTER] jar:file:/scouter/scouter.agent.jar
20190421 06:32:22 [SCOUTER] objType:java
~~~

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

20190421 06:32:24 [A119] Agent UDP local.port=0
2019-04-21 06:32:24.122  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : Starting GuestbookH2Application v0.0.1-
SNAPSHOT on eb15506a0372 with PID 1 (/guestbook_H2.jar started by root in /)
2019-04-21 06:32:24.132  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : No active profile set, falling back to default
profiles: default
20190421 06:32:24 [A103] HTTP javax/servlet/http/HttpServlet
```



bind mount (5/7)

④ 스카우터 클라이언트에서 해당 트랜잭션의 프로파일 정보 확인



bind mount (6/7)

⑤ bindmount02 컨테이너의 bash 쉘을 실행시키고 mount 정보 확인폴더 하위에 test.txt 파일을 생성

```
[root@dockeredu ~]# docker container exec -it bindmount02 bash

root@eb15506a0372:~# mount | grep /dev/sda3
/dev/sda3 on /scouter type xfs (ro,relatime,attr2,inode64,noquota)
/dev/sda3 on /upload type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sda3 on /etc/resolv.conf type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sda3 on /etc/hostname type xfs (rw,relatime,attr2,inode64,noquota)
/dev/sda3 on /etc/hosts type xfs (rw,relatime,attr2,inode64,noquota)
```

→ /scouter 폴더가 ReadOnly 모드로 마운트 되어 있음

→ /upload 폴더가 ReadWrite 모드로 마운트 되어 있음

→ 네트워크 관련 파일도 컨테이너에 bind mount 되어 있음

⑥ /scouter 폴더 하위에 test.txt 파일을 생성

```
root@eb15506a0372:/# cd /scouter

root@eb15506a0372:/scouter# ls -al
total 2176
drwxr-xr-x 4 root root      57 Apr 19 13:04 .
drwxr-xr-x 1 root root      44 Apr 21 06:32 ..
drwxr-xr-x 2 root root      55 Apr 21 06:23 conf
drwxr-xr-x 2 root root     205 Apr 19 13:04 plugin
-rw-r--r-- 1 root root 2224554 Mar 17 08:47 scouter.agent.jar

root@eb15506a0372:/scouter# touch test.txt
touch: cannot touch 'test.txt': Read-only file system

root@eb15506a0372:/scouter# exit

[root@dockeredu ~]#
```

Read-only 모드로 마운트 되어 있어 파일을 생성할 수 없음

bind mount (7/7)

⑦ HOST 시스템에서 docker inspect 명령을 이용하여 c02-3 컨테이너의 파일시스템 마운트 정보 확인

```
[root@dockeredu ~]# docker container inspect bindmount02 | grep -A 17 \"Mounts\"
 "Mounts": [
            {
                "Type": "bind",
                "Source": "/app/scouter/agent.java",
                "Destination": "/scouter",
                "Mode": "ro",
                "RW": false,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/tmp/updir",
                "Destination": "/upload",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

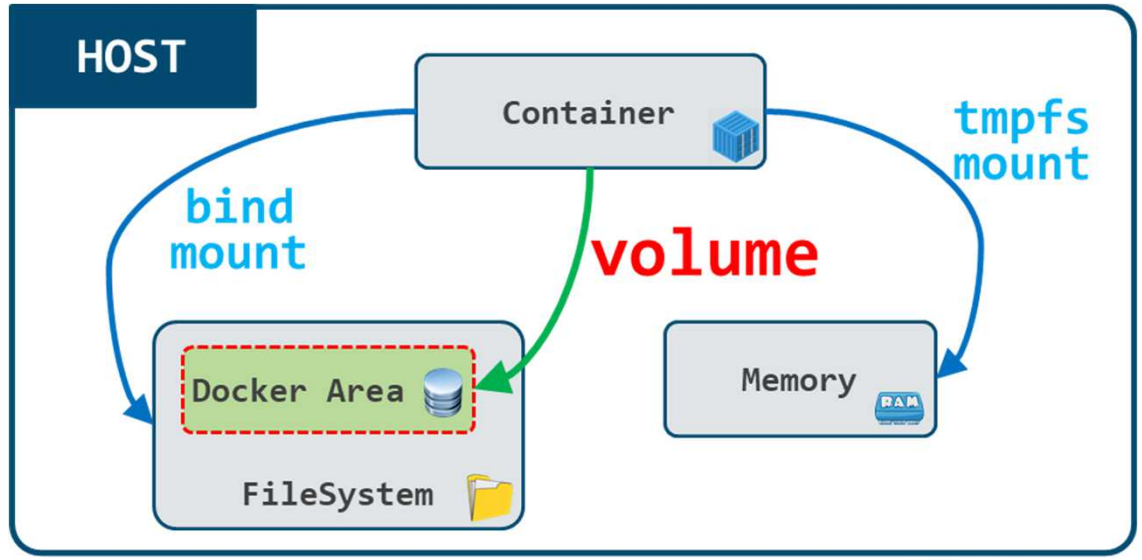
⑧ HOST 시스템에서 "Mount.Source" 값에 해당하는 /tmp/updir 폴더 파일 조회

```
[root@dockeredu ~]# ls -l /tmp/updir
total 4
-rw-r--r-- 1 root root 1443 Apr 21 15:16 20190421061620520_edu_anaconda-ks.cfg
```

도커 볼륨 개요

- **Docker Volume**
  - 도커 볼륨은 UFS(Union File System)의 일부가 아니며, HOST 파일시스템상의 일반적인 디렉토리나 파일로 존재
  - HOST 시스템상의 Native Storage Driver를 사용하므로 UFS(Union File System)보다 오버헤드가 적어 I/O 성능이 개선됨
  - 볼륨을 사용하면 컨테이너에서 수행한 쓰기 작업은 즉각적으로 commit 없이 스토리지에 영구적으로 반영됨
  - HOST 시스템의 /var/lib/docker/volumes 하위에 볼륨이 생성됨
- **볼륨 생성 방법**
  - ① 이름 없는 볼륨 생성
    - docker container run에 -v 옵션 사용 : -v <컨테이너 마운트 경로>
  - ② 이름 있는 볼륨 생성
    - docker volume create 명령을 사용하여 명시적으로 볼륨 생성
  - ③ 데이터 전용 컨테이너 생성
    - docker container run에 --volume-from 옵션 사용
- **Docker Volume 관련 명령어**

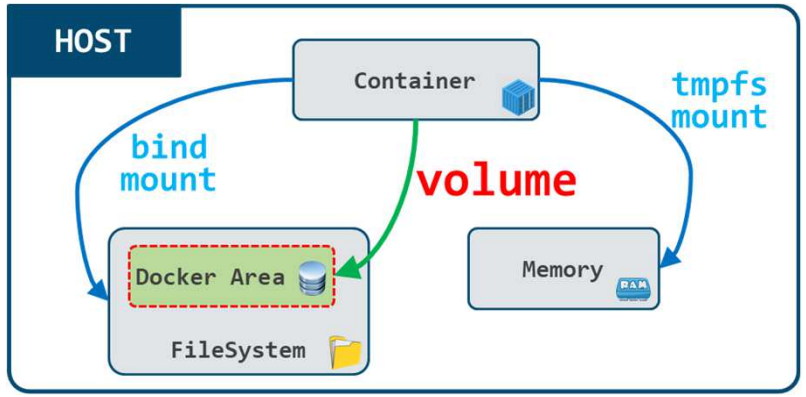
```
[root@dockeredu ~]# docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```



docker volume (1/7)

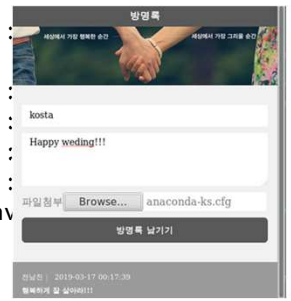
① 이름 없는 볼륨 사용

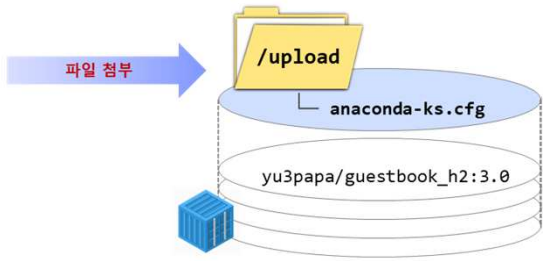
- docker container run에 -v 옵션 사용
  - -v <컨테이너 마운트 경로>



① yu3papa/guestbook_h2:3.0 이미지로부터 -v /upload 옵션을 적용하여 volume01 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run -v /upload --name=volume01 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-21 05:39:06.207  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : Starting GuestbookH2Application v0.0.1-SNAPSHOT on
6d0c6fb91fe1 with PID 1 (/guestbook_H2.jar started by root in /)
2019-04-21 05:39:06.211  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     :
default
2019-04-21 05:39:07.850  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  :
2019-04-21 05:39:07.901  INFO 1 --- [           main] o.apache.catalina.core.StandardService   :
2019-04-21 05:39:07.901  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  :
2019-04-21 05:39:07.919  INFO 1 --- [           main] o.a.catalina.core.AprLifecycleListener   :
optimal performance in production environments was not found on the java.library.path: [/usr/jav
gnu/jni:/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib]
```





docker volume (2/7)

② docker inspect 명령을 이용하여 volume01 컨테이너의 파일시스템 Mounts와 Volumes 확인

```
[root@dockeredu ~]# docker container inspect volume01 | grep -A 10 \"Mounts\"
        "Mounts": [
            {
                "Type": "volume",
                "Name": "5d1d8ff5388153c544b51341548e4b019263128bb92e2001ac40ef02195e62b4",
                "Source": "/var/lib/docker/volumes/55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b/_data",
                "Destination": "/upload",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
[root@dockeredu ~]# docker container inspect volume01 | grep -A 10 \"Volumes\"
 "Volumes": {
                "/upload": {}
            },
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
```

③ "Mount.Source" 값에 해당하는 폴더 파일 조회

```
[root@dockeredu ~]# ls -l /var/lib/docker/volumes/55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b/_data
total 4
-rw-r--r-- 1 root root 1443 Apr 21 14:39 20190421053945614_edu_anaconda-ks.cfg
```

④ docker volume ls 명령으로 생성된 volume 조회

```
[root@dockeredu ~]# docker volume ls
DRIVER              VOLUME NAME
local               55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b
```

이름없는 볼륨 생성

⑤ docker volume inspect 명령으로 생성된 volume 상세 조회

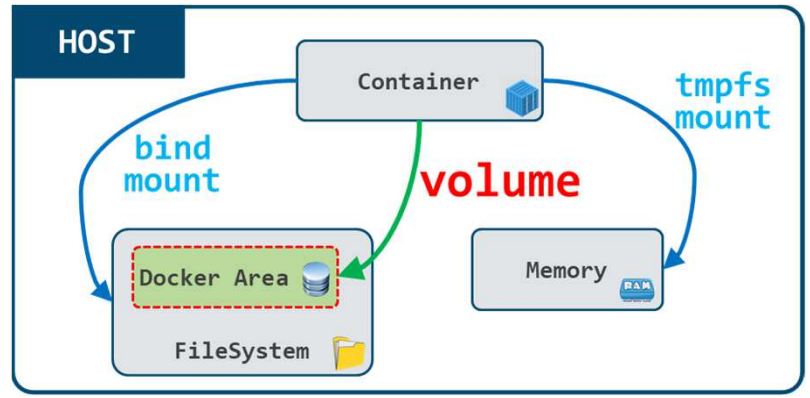
```
[root@dockeredu ~]# docker volume inspect 55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b
[
    {
        "CreatedAt": "2019-04-21T18:33:25+09:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b/_data",
        "Name": "55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b",
        "Options": null,
        "Scope": "local"
    }
]
```

docker volume (3/7)

② 이름 있는 볼륨 사용

- docker volume create 명령을 이용하여 명시적으로 볼륨 생성하고, 생성된 볼륨을 사용하여 영구적으로 데이터를 보존하는 방법

```
[root@dockeredu ~]# docker volume create --help

Usage:  docker volume create [OPTIONS] [VOLUME]

Create a volume

Options:
  -d, --driver string   Specify volume driver name (default "local")
      --label list      Set metadata for a volume
  -o, --opt map         Set driver specific options (default map[])
```



① HOST 시스템에서 uploadVol 이름을 갖는 볼륨을 생성

```
[root@dockeredu ~]# docker volume create uploadVol
uploadVol
```

② 생성된 볼륨 조회

```
[root@dockeredu ~]# docker volume ls
DRIVER              VOLUME NAME
local               55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b
local               uploadVol
```

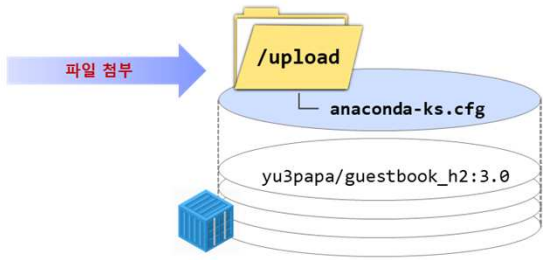
③ uploadVol 볼륨 상세 조회

```
[root@dockeredu ~]# docker volume inspect uploadVol
[
    {
        "CreatedAt": "2019-04-21T19:08:36+09:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/uploadVol/_data",
        "Name": "uploadVol",
        "Options": {},
        "Scope": "local"
    }
]
```

docker volume (4/7)

④ yu3papa/guestbook_h2:3.0 이미지로부터 -v uploadVol:/upload 옵션을 적용하여 volume02 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run -v uploadVol:/upload --name=volume02 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

20190421 06:32:24 [A119] Agent UDP local.port=0
2019-04-21 06:32:24.122  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Applicat
SNAPSHOT on eb15506a0372 with PID 1 (/guestbook_H2.jar started by root in /)
2019-04-21 06:32:24.132  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : No active profile set, falling back to default
profiles: default
```



⑤ HOST 시스템에서 uploadVol 볼륨의 MountPoints에 해당하는 /var/lib/docker/volumes/uploadVol/_data 폴더 조회

```
[root@dockeredu ~]# ls -l /var/lib/docker/volumes/uploadVol/_data
total 4
-rw-r--r-- 1 root root 1443 Apr 21 19:18 20190421101855600_edu_anaconda-ks.cfg
```

⑥ volume02 컨테이너를 종료한후 삭제하고 나서 /var/lib/docker/volumes/uploadVol/_data 조회한 후 웹에서 첨부한 파일이 유지 되는 지확인

```
[root@dockeredu ~]# docker container rm volume02
volume02

[root@dockeredu ~]# ls -l /var/lib/docker/volumes/uploadVol/_data
total 4
-rw-r--r-- 1 root root 1443 Apr 21 19:18 20190421101855600_edu_anaconda-ks.cfg
```

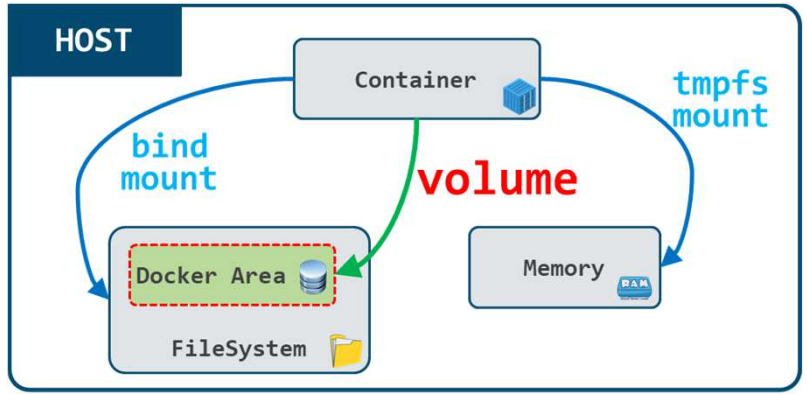
컨테이너 삭제된 후에도 사용자가 웹에서 업로드 한 파일은 HOST 시스템에 영속적으로 보존됨

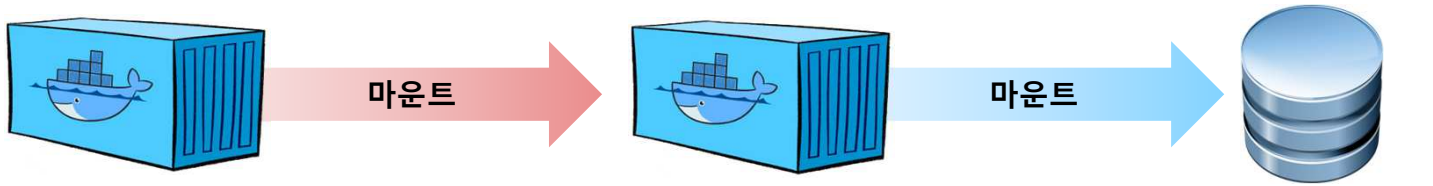
docker volume (5/7)

③ 데이터 전용 컨테이너 사용

- 볼륨 전용의 컨테이너를 생성하고, --volumes-from 옵션을 이용하여 다른 컨테이너에서 볼륨을 공유해서 사용하는 방법
- 데이터 전용 컨테이너는 실행하지 않고 생성만 되어 있어도 볼륨은 공유됨.

```
[root@dockeredu ~]# docker container run --help

Usage:  docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

Options:
 -v, --volume list                    Bind mount a volume
     --volume-driver string            Optional volume driver for the container
     --volumes-from list               Mount volumes from the specified container(s)
```





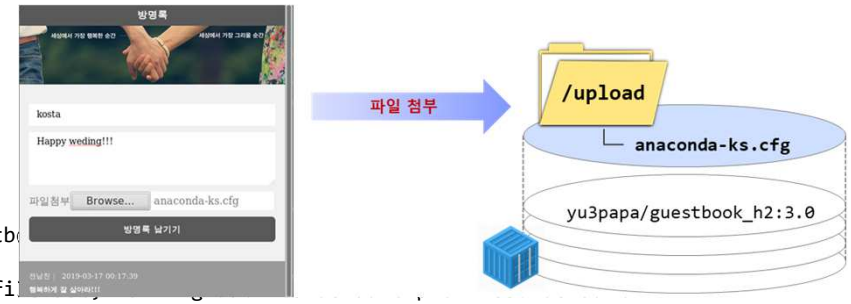
docker volume (6/7)

① 이름없는 볼륨을 사용하는 데이터전용 컨테이너를 dataOnlyCon 이름으로 생성

```
[root@dockeredu ~]# docker container create --name=dataOnlyCon -v /upload busybox
8aa6ca42b15afc0a73ef7bea3eebb64726d4dd3ba007125393ba43bf61afbd60
```

② dataOnlyCon 컨테이너를 --volumes-from 옵션으로 사용하는 volume03 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run --volumes-from dataOnlyCon --name=volume03 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

20190421 06:32:24 [A119] Agent UDP local.port=0
2019-04-21 06:32:24.122  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : Starting Guestb
(/guestbook_H2.jar started by root in /)
2019-04-21 06:32:24.132  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Application     : No active profi
```



③ volume03 컨테이너에 bash 쉘을 실행하고 /upload 폴더에 anaconda-ks.cfg 파일이 업로드 되었는지 확인하고 쉘을 빠져나오기

```
[root@dockeredu ~]# docker container exec -it volume03 bash

root@d5f9e10905bd:/# ls -l /upload
total 4
-rw-r--r-- 1 root root 1443 Apr 21 11:08 20190421110810718_edu_anaconda-ks.cfg
root@d5f9e10905bd:/# exit
[root@dockeredu ~]#
```

docker volume (7/7)

④ volume03 컨테이너의 Mounts 정보를 출력하고 어떤 볼륨이 마운트 되었는지 확인

```
[root@dockeredu ~]# docker container inspect volume03 | grep -A10 \"Mounts\"
        "Mounts": [
            {
                "Type": "volume",
                "Name": "86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8",
                "Source": "/var/lib/docker/volumes/86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8/_data",
                "Destination": "/upload",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
```

⑤ 볼륨 목록을 조회하고 위 스텝 결과의 Name에 해당하는 볼륨이 존재하는지 확인

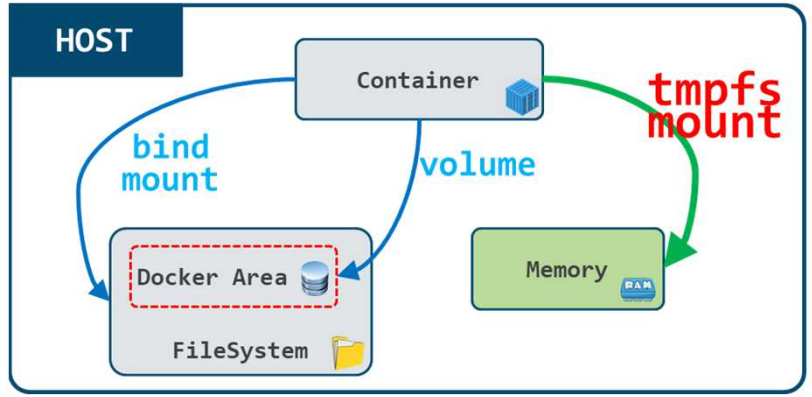
```
[root@dockeredu ~]# docker volume ls
DRIVER              VOLUME NAME
local               55432f278c1fea85bb8803b4c17aace9a3eda39ef86c124f89b30dbed2cac18b
local               86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8
local               uploadVol
```

⑥ 확인된 볼륨의 상세정보를 확인하고 MountPoint 경로의 파일 목록의 조회하여 anaconda-ks.cfg 파일이 존재하는지 확인

```
[root@dockeredu ~]# docker volume inspect 86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8
[
    {
        "CreatedAt": "2019-04-21T20:08:10+09:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8/_data",
        "Name": "86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8",
        "Options": null,
        "Scope": "local"
    }
]
[root@dockeredu ~]# ls -l /var/lib/docker/volumes/86d3480f7b702811b91de4927105366c33a5df5c957c39b22f19f46e9d9044c8/_data
total 4
-rw-r--r-- 1 root root 1443 Apr 21 20:08 20190421110810718_edu_anaconda-ks.cfg
```

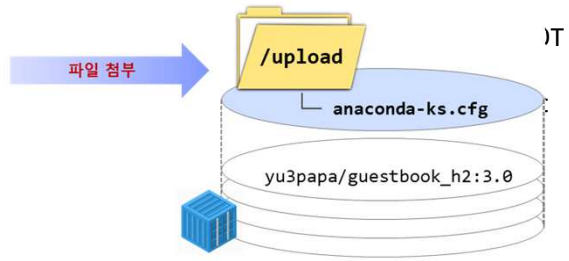
tmpfs mount (1/2)

- **tmpfs mount 사용**
  - docker container run에 --tmpfs 옵션 사용
    - --tmpfs list Mount a tmpfs directory
  - 컨테이너 실행시 HOST 머신의 메모리에 마운트 됨
  - 컨테이너가 정지되면 data도 삭제됨
  - bind mout와 volume과 달리 영속적인 데이터 저장이 안됨
  - 컨테이너간의 볼륨 공유가 되지 않음
  - HOST 시스템이 Linux 일 때만 사용 가능



① yu3papa/guestbook_h2:3.0 이미지로부터 --tmpfs /upload 옵션을 적용하여 tmpfs01 컨테이너를 시작하고 HOST 시스템의 "anaconda-ks.cfg" 파일 첨부

```
[root@dockeredu ~]# docker container run --tmpfs /upload --name=tmpfs01 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar

.   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-21 05:39:06.207  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Applicati
on 6d0c6fb91fe1 with PID 1 (/guestbook_H2.jar started by root in /)
2019-04-21 05:39:06.211  INFO 1 --- [           main] c.j.guestbook.GuestbookH2Applicati
profiles: default
2019-04-21 05:39:07.850  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebS
```



tmpfs mount (2/2)

② tmpfs 컨테이너에 bash 쉘을 실행하고 /upload 폴더의 목록으로 조회하여 업로드한 파일이 존재하는 지 확인

```
[root@dockeredu ~]# docker container exec -it tmpfs01 bash
root@58e72ac4a191:/# ls -l /upload
total 4
-rw-r--r-- 1 root root 1443 Apr 21 12:16 20190421121639604_edu_anaconda-ks.cfg
root@58e72ac4a191:/#
```

③ docker inspect 명령을 이용하여 tmpfs 컨테이너의 HostConfig 섹션 확인

```
[root@dockeredu ~]# docker container inspect tmpfs01 | grep -A 40 HostConfig
```


```
"HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "shareable",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "Tmpfs": {
                "/upload": ""
            },
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
```


Mounts 섹션이 아닌 HostConfig 섹션에 정보가 존재

Good use cases

- **Good use cases for volumes**
  - Volumes are the preferred way to persist data in Docker containers and services. Some use cases for volumes include:
    - Sharing data among multiple running containers. If you don't explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.
    - When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you decouple the configuration of the Docker host from the container runtime.
    - When you want to store your container's data on a remote host or a cloud provider, rather than locally.
    - When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice. You can stop containers using the volume, then back up the volume's directory (such as /var/lib/docker/volumes/<volume-name>).

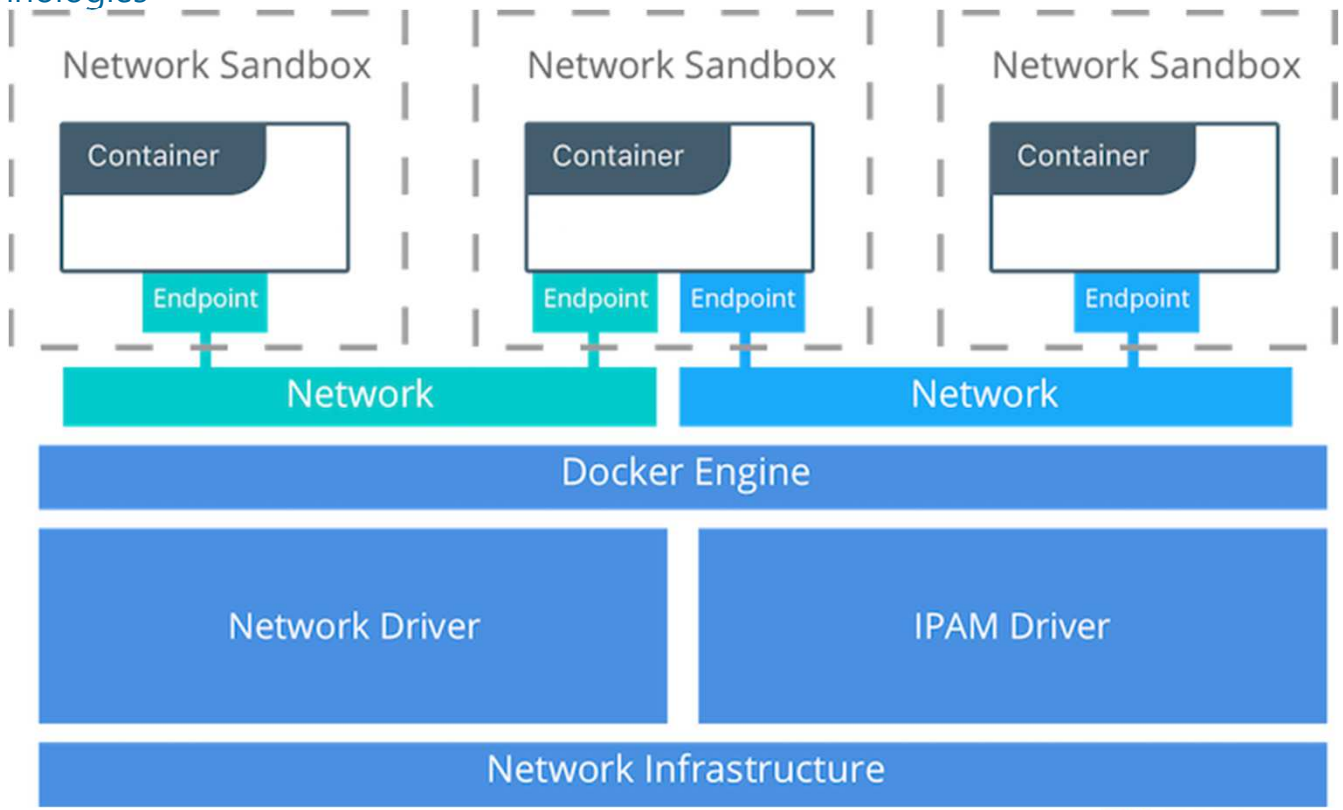
- **Good use cases for bind mounts**
  - In general, you should use volumes where possible. Bind mounts are appropriate for the following types of use case:
    - Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting /etc/resolv.conf from the host machine into each container.
    - Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.
    - If you use Docker for development this way, your production Dockerfile would copy the production-ready artifacts directly into the image, rather than relying on a bind mount.
    - When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

- **Good use cases for tmpfs mounts**
  - tmpfs mounts are best used for cases when you do not want the data to persist either on the host machine or within the container. This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.

도커 네트워크

CNM

- **Container Network Model**
  - The CNM is an open-source container networking specification contributed to the community by Docker, Inc
  - The CNM defines sandboxes, endpoints, and networks
  - Libnetwork is Docker's implementation of the CNM
  - Libnetwork is extensible via pluggable drivers
  - Drivers allow libnetwork to Support many network technologies
  - Libnetwork is cross-platform and open-source



docker network 개요

▪ 도커가 지원하는 네트워크

```
[root@dockeredu ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
585341fe1b62        bridge              bridge              local
1b61980ce83f        host                host                local
430302211c70        none                null                local
```

- bridge network(default)
  - HOST 시스템의 네트워크와 브릿지 연결을 통한 네트워킹
- host network
  - HOST 시스템의 네트워크 공유(network namespace 격리 해제와 동일)
- none network
  - 네트워크 사용안함

- 네트워크별 상세 정보

none

```
# docker network inspect none
[
    {
        "Name": "none",
        "Id": "430302211c70",
        "Created": "2019-04-21T17:03:11",
        "Scope": "local",
        "Driver": "null",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

host

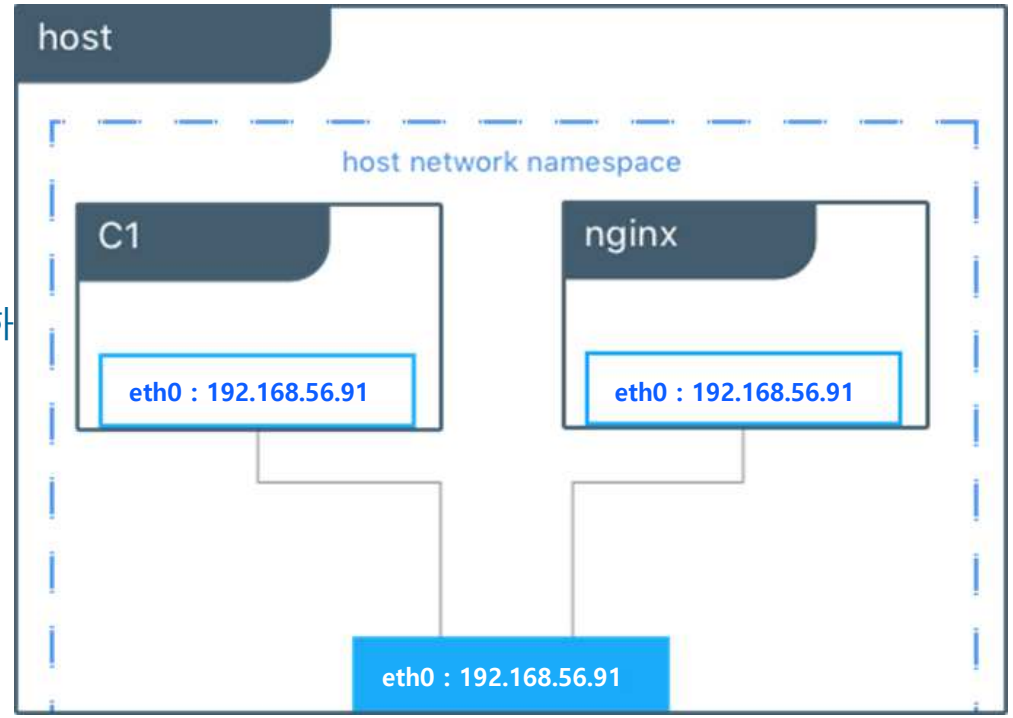
```
# docker network inspect host
[
    {
        "Name": "host",
        "Id": "1b61980ce83f",
        "Created": "2019-04-21T17:03:11",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

bridge

```
# docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "585341fe1b62",
    "Created": "2019-04-22T10:28:22",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

host network (1/2)

- **HOST 시스템의 네트워크 공유**
  - docker containr run시 --network=host 옵션으로 실행
  - 컨테이너의 network 격리를 해제
  - brige같은 네트워크를 거치지 않고 HOST 시스템의 네트워크를 그대로 사용하
  - 보안에 잠재적인 위험이 있음
    - 컨테이너에 root로 로그인되어 있기 때무에 HOST 네트워크를 파괴할 수 있음



① yu3papa/guestbook_h2:3.0 이미지로부터 --network=host 옵션을 적용하여 nethost01 컨테이너를 시작

```
[root@dockeredu ~]# docker container run -d --network=host --name=nethost01 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
```

② HOST 시스템에서 ifconfig 명령으로 추가적인 veth 인터페이스가 생성되지 않았음을 확인

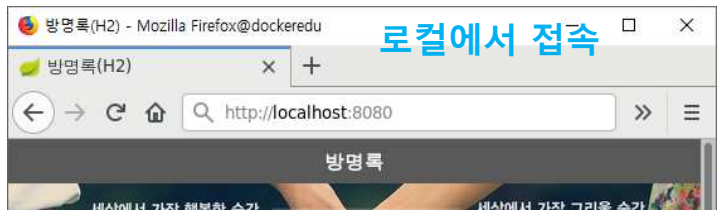
```
[root@dockeredu ~]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ~~~
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.91  netmask 255.255.255.0  broadcast 192.168.56.255
        ~~~
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        ~~~
```

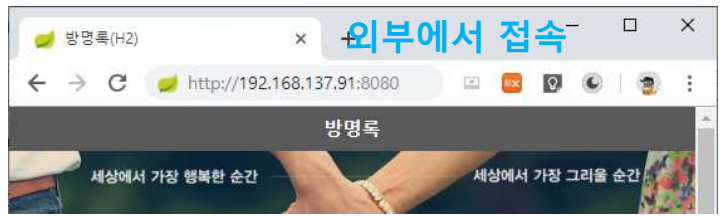
host network (2/2)

③ nethost01 컨테이너의 bash 쉘을 실행시키고 ifconfig 명령을 실행하여 IP주소가 HOST 시스템과 동일함을 확인하고 /etc/hosts 파일 내용도 동일한지 확인

```
[root@dockeredu ~]# docker exec -it nethost01 bash

root@dockeredu:/# apt update
root@dockeredu:/# apt install net-tools
root@dockeredu:/# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ~~~
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.91  netmask 255.255.255.0  broadcast 192.168.56.255
        ~~~
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        ~~~
root@dockeredu:/# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.56.91          dockeredu
192.168.56.92          registry
192.168.56.1           host-svr
```

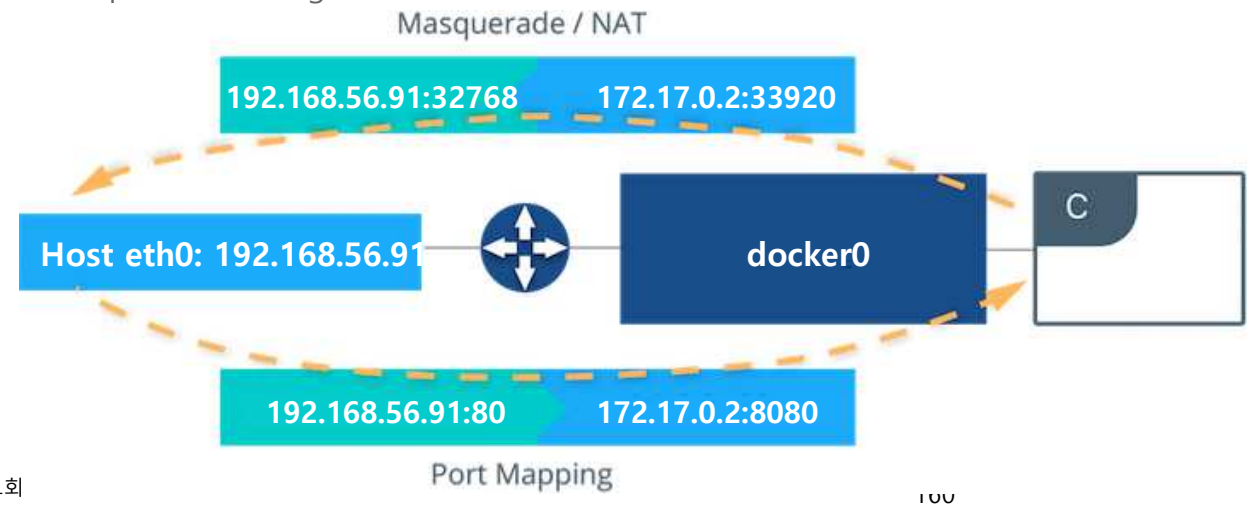
④ HOST 시스템에서 http://localhost:8080으로 접속이 되는지, 외부 시스템인 노트북에서 http://192.168.56.91:8080 으로 접속이 되는지 확인

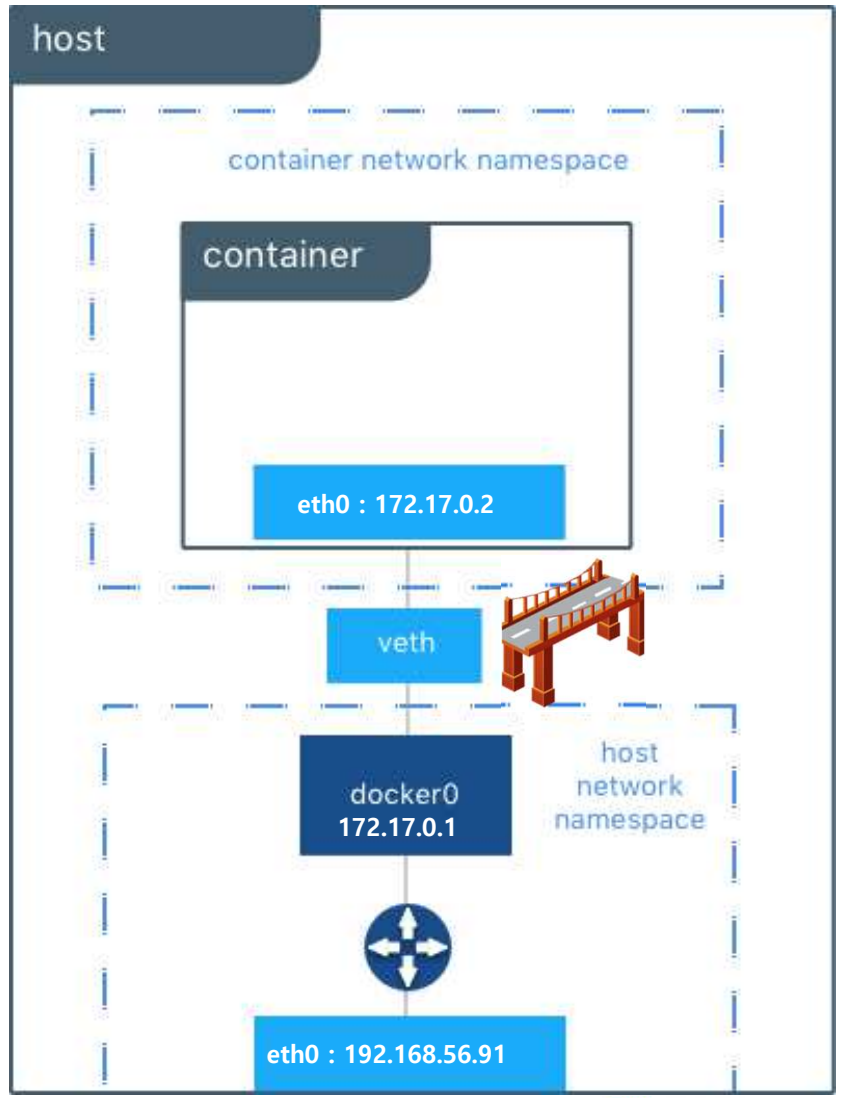




bridge network (1/4)

- **bridge network 개요**
  - docker containr run시 --network=bridge 옵션(default)으로 실행
  - Docker를 설치하면 서버의 물리 NIC가 docker0라는 가상 브리지 네트워크로 연결됨
  - 컨테이너가 실행되면 컨테이너에 172.17.0. 0/16이라는 서브넷 마스크를 가진 프라이빗 IP 주소가 eth0 인터페이스에 자동으로 할당
  - 컨테이너에 고정 IP 할당 불가
- **bridge network 와 port mapping**
  - 외부에서 접속하게 하려면 port mapping 옵션을 적용해야 함
    - ex) docker container run -p 80:8080
  - NAPT 기술과 iptables 를 이용하여 컨테이너가 외부와 통신
    - NAPT(Network Address Port Translation)는 하나의 IP 주소를 여러 컴퓨터가 공유하는 기술로, IP 주소와 포트 번호를 변환하는 기능으로 "IP 마스커레이드[1]"라고도 부름
    - iptables의 port-forwarding 기술을 이용하여 외부 패킷을 컨테이너에게 전달





1) mascarade : 가면무도회

bridge network (2/4)

① 지금까지 생성된 모든 컨테이너를 삭제

```
[root@dockeredu ~]# docker container rm -f $(docker container ls -aq)
```

② ip addr 명령으로 현재 HOST의 NIC 정보를 확인

```
[root@dockeredu ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.91/24 brd 192.168.56.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe1b:adc4/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:d1:0f:45:ee brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:d1ff:fe0f:45ee/64 scope link
       valid_lft forever preferred_lft forever
```

③ iptables 명령을 이용하여 현재 nat정보와 mascarade정보 확인

```
[root@dockeredu ~]# iptables -t nat -vnL
Chain PREROUTING (policy ACCEPT 4 packets, 934 bytes)
 pkts bytes target     prot opt in     out     source               destination
   58  3554 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 4 packets, 934 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 241 packets, 14566 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1088 65280 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 241 packets, 14566 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    3   826 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0
```

bridge network (3/4)

④ yu3papa/guestbook_h2:3.0 이미지로부터 --network=bridge 옵션과 -p 80:8080 옵션을 적용하여 netbridge01 컨테이너를 시작

```
# docker container run -d --name=netbridge01 --network=bridge -p 80:8080 yu3papa/guestbook_h2:3.0 java -jar guestbook_H2.jar
```

⑤ ip addr 명령으로 현재 HOST의 NIC 정보를 확인

```
[root@dockeredu ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1b:ad:c4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.91/24 brd 192.168.56.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe1b:adc4/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:d1:0f:45:ee brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:d1ff:fe0f:45ee/64 scope link
       valid_lft forever preferred_lft forever
34: veth75fd51e@if33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 32:67:c3:8c:ce:a6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::3067:c3ff:fe8c:cea6/64 scope link
       valid_lft forever preferred_lft forever
```

veth 추가

⑥ iptables 명령을 이용하여 현재 NAT정보와 MASQUERADE 정보 확인

```
[root@dockeredu ~]# iptables -t nat -vnL
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   58  3554 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 18 packets, 1132 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1107 66420 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 18 packets, 1132 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0
    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2           tcp dpt:8080

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    3   826 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0
    0     0 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0            tcp dpt:80 to:172.17.0.2:8080
```

bridge network (4/4)

⑦ netbridge01 컨테이너에 bash 쉘로 접속

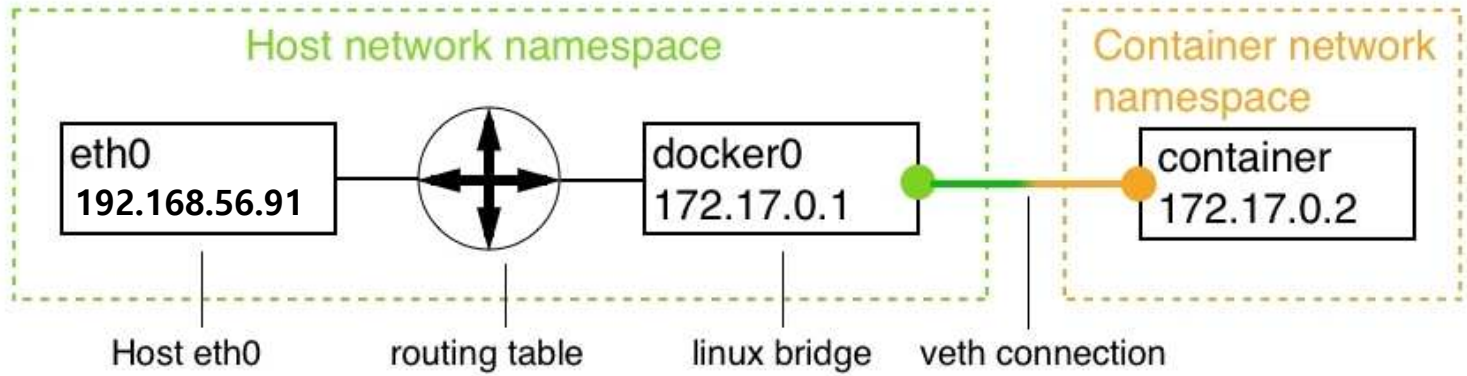
```
[[root@dockeredu ~]# docker container exec -it netbridge01 bash
root@08ae56df2806:/#
```

⑧ 컨테이너안에서 ip addr 명령으로 네트워크 정보 확인

```
root@9c30009e7615:/# apt update
root@9c30009e7615:/# apt install net-tools
root@9c30009e7615:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ~~~

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        ~~~
```

⑨ hosts파일과 resolve.conf 파일 내용 확인

```
root@08ae56df2806:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      08ae56df2806

root@08ae56df2806:/# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.56.1
```



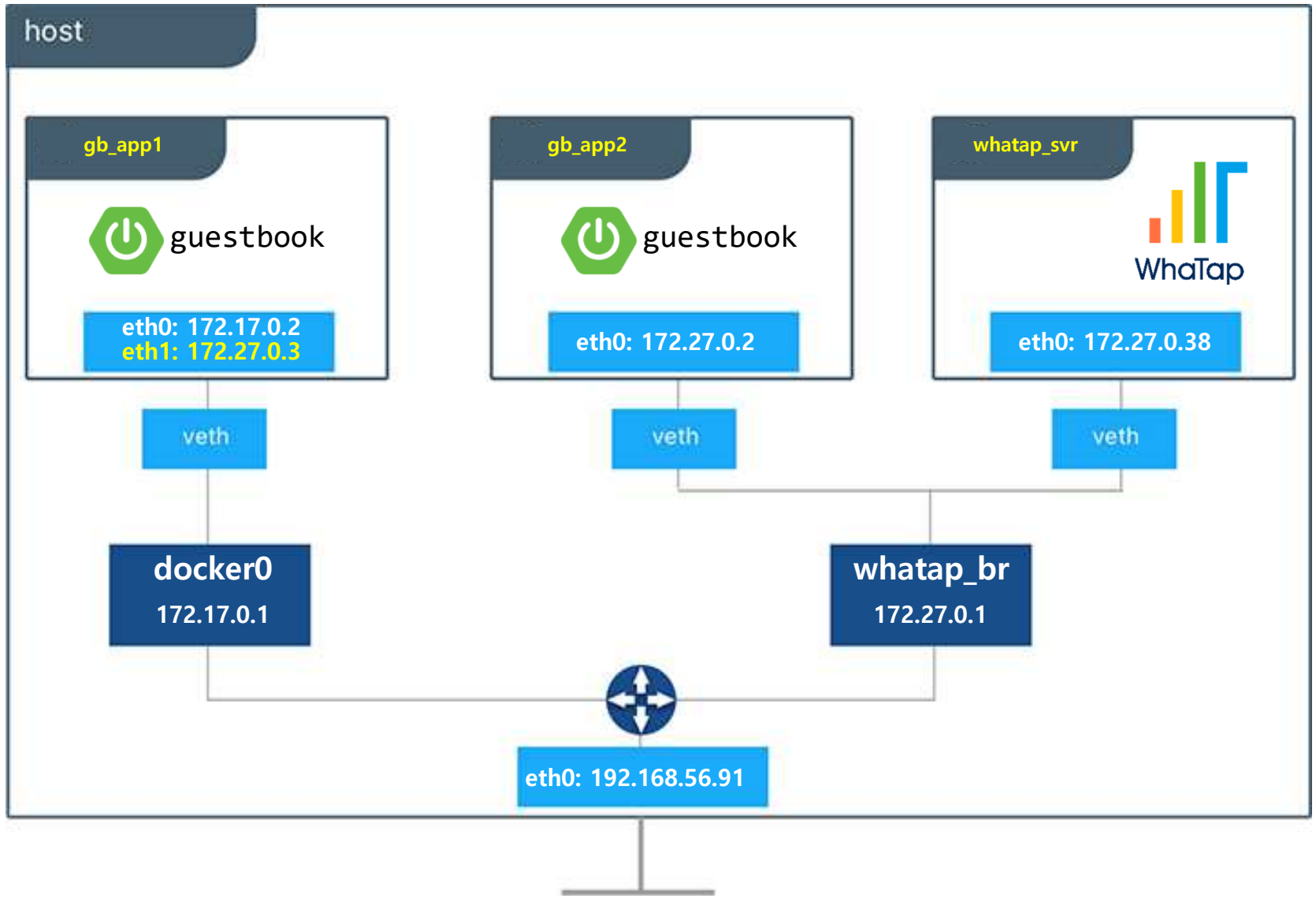
custom bridge network (1/6)

- **사용자 정의 bridge network**
  - Linux bridge가 HOST에 할당됨
  - 수동 IP 주소 및 서브넷 지정이 가능
  - 서브넷을 지정하지 않으면 IPAM 드라이버가 172.18.0.0/16 서브넷부터 순차적으로 할당
  - NAT 및 라우팅을 지원하므로 외부 네트워크 연결 가능
  - docker network create 명령으로 사용자 정의 네트워크 생성
- **docker network create**

```
[root@dockeredu ~]# docker network create --help

Usage:  docker network create [OPTIONS] NETWORK

Create a network

Options:
      --attachable           Enable manual container attachment
      --aux-address map      Auxiliary IPv4 or IPv6 addresses used by Network driver (default map[])
      --config-from string   The network from which copying the configuration
      --config-only          Create a configuration only network
  -d, --driver string        Driver to manage the Network (default "bridge")
      --gateway strings      IPv4 or IPv6 Gateway for the master subnet
      --ingress              Create swarm routing-mesh network
      --internal             Restrict external access to the network
      --ip-range strings     Allocate container ip from a sub-range
      --ipam-driver string   IP Address Management Driver (default "default")
      --ipam-opt map         Set IPAM driver specific options (default map[])
      --ipv6                 Enable IPv6 networking
      --label list           Set metadata on a network
  -o, --opt map              Set driver specific options (default map[])
      --scope string         Control the network's scope
      --subnet strings       Subnet in CIDR format that represents a network segment
```

custom bridge network (2/6)

- 실습 개요



custom bridge network (3/6)

① 172.27.0.0 서브넷 whatap-bridge 네트워크 생성

```
[root@dockeredu ~]# docker network create --driver=bridge --subnet=172.27.0.0/16 whatap-bridge
9beb5a7baba491c83aa1322d789072fa0ac0a202a7a4b2d90ada94e6e78cbc69
```

② whatap-bridge 사용자 정의 브리지 네트워크가 정상적으로 생성되었는지 확인

```
[root@dockeredu ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
585341fe1b62        bridge              bridge              local
1b61980ce83f        host                host                local
430302211c70        none                null                local
9beb5a7baba4        whatap-bridge       bridge              local
[root@dockeredu ~]#
```

```
[root@dockeredu ~]# ifconfig
br-9beb5a7baba4: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.27.0.1  netmask 255.255.0.0  broadcast 172.27.255.255
        ether 02:42:83:9d:d2:00  txqueuelen 0  (Ethernet)
        RX packets 2693934  bytes 2535933292 (2.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2428018  bytes 2383251380 (2.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:d1ff:fe0f:45ee  prefixlen 64  scopeid 0x20<link>
        ether 02:42:d1:0f:45:ee  txqueuelen 0  (Ethernet)
        RX packets 805  bytes 1317399 (1.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 727  bytes 86950 (84.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.91  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe1b:adc4  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:1b:ad:c4  txqueuelen 1000  (Ethernet)
        RX packets 2693934  bytes 2535933292 (2.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2428018  bytes 2383251380 (2.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 1312726  bytes 11060031045 (10.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1312726  bytes 11060031045 (10.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

custom bridge network (4/6)

③ 와탭 APM 수집서버 실행 → 특정 IP에 임시라이센스가 적용되어 있음

```
[root@dockeredu ~]# docker container run -it --name=whatapsvr --network=whatap-bridge --ip=172.27.0.38 -p 8888:8888
yu3papa/whatap_collector:2.0
Unable to find image 'yu3papa/whatap_collector:2.0' locally
2.0: Pulling from yu3papa/whatap_collector
62daa894dfde: Pull complete
Digest: sha256:dba9a5800226bd89579fa35b601bb6209942a3ba2ce2f81de07f556b172a9be0
Status: Downloaded newer image for yu3papa/whatap_collector:2.0

root@5b88cfae31a3:/# cd /whatap/bin; ./start.sh
logs directory exists.
```

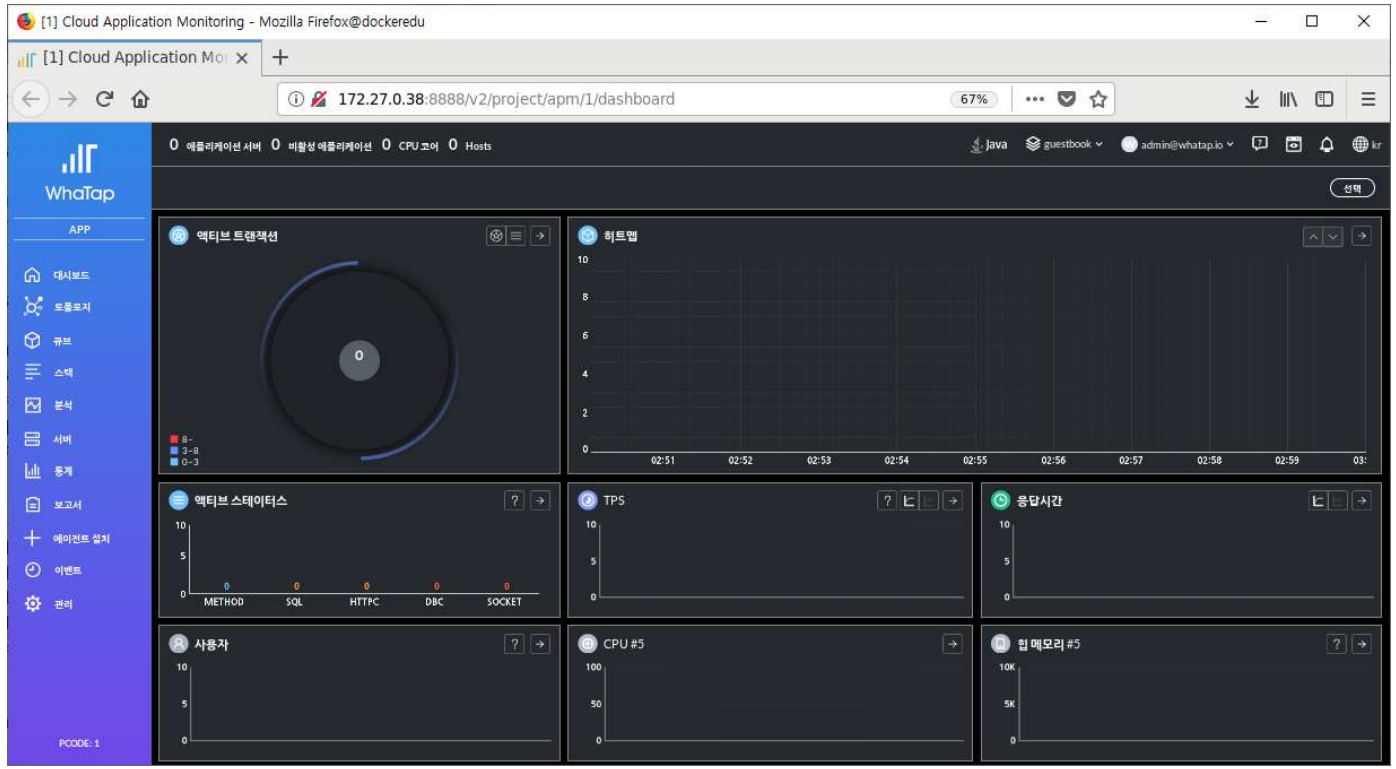
→ *ctrl + pq 를 입력하여 컨테이너 쉘 빠져나오기*

```
root@5b88cfae31a3:/#
```

ctrl + pq

④ 웹브라우저를 열고 아래 주소로 이동

- 주소 : http://172.27.0.38:8888
- 계정/암호 : admin@whatap.io / admin



custom bridge network (5/6)

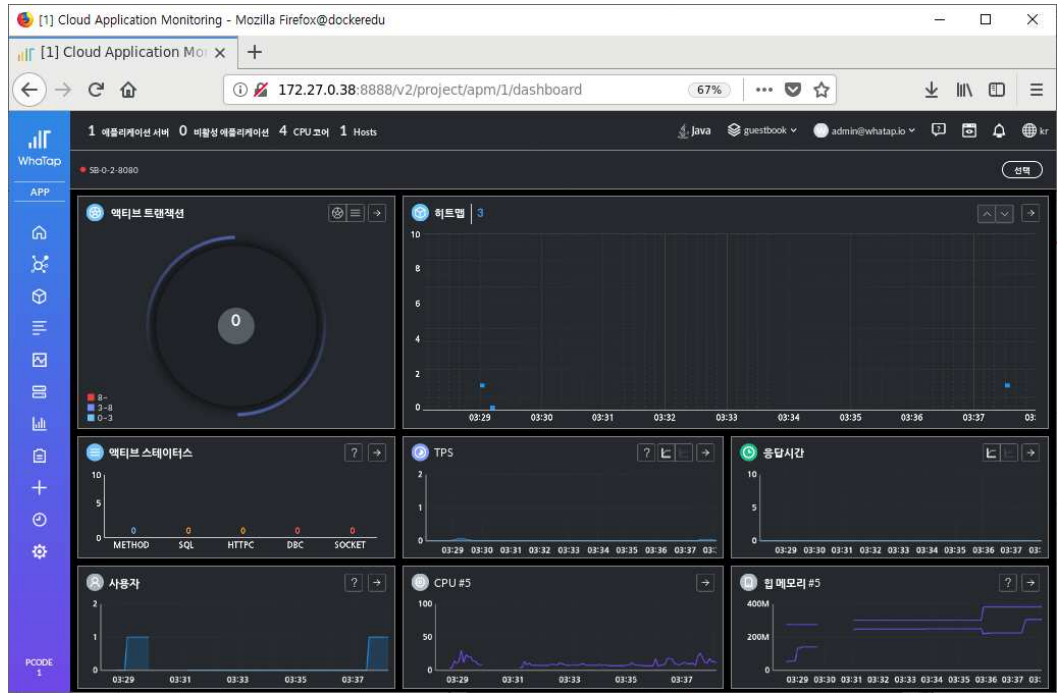
⑤ Docker Hub에서 yu3papa/whatap_guestbook:2.0 이미지로 부터 whatap-bridge 네트워크를 사용하는 whatapguestbook1 컨테이너 생성

```
[root@dockeredu ~]# docker container run -d --name=whatapguestbook1 --network=whatap-bridge yu3papa/whatap_guestbook:2.0
a258fac52506ac5c9d36ae728360def3e9237eea782fa36b1b5c619d52899cbc
[root@dockeredu ~]#
```

⑥ whatapguestbook1 컨테이너의 IP를 확인 후 브라우져로 방명록 어플리케이션 접속하여, 게시글 작성

```
[root@dockeredu ~]# docker container inspect whatapguestbook1 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.27.0.2",
```

`http://172.27.0.2:8080` 으로 접속하여 게시글을 작성하고 `Whatap APM`에서 서비스 모니터링이 되는지 확인



custom bridge network (6/6)

⑦ Docker Hub에서 yu3papa/whatap_guestbook:2.0 이미지로 부터 default bridge 네트워크를 사용하는 whatapguestbook2 컨테이너 생성

```
[root@dockeredu ~]# docker container run -d --name=whatapguestbook2 yu3papa/whatap_guestbook:2.0
dc420701c0ad1e24a5467c30f05c856fa51e72121901d5f1ebc23e9b7870150e
[root@dockeredu ~]#
```

⑧ whatapguestbook2 컨테이너의 IP를 확인 후 브라우져로 방명록 어플리케이션 접속하여, 게시글 작성

```
[root@dockeredu ~]# docker container inspect whatapguestbook2 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",
```

http://172.17.0.2:8080 으로 접속하여 게시글을 작성하고 Whatap APM에서 서비스 모니터링이 되는지 확인
→ 모니터링이 되지 않는 이유는 디폴트 bridge 네트워크와 whatap-bridge 커스텀 네트워크간에 라우팅이 되지 않기 때문임

⑨ docker network connect를 이용하여 whatapguestbook2 컨테이너에거 whatap-bridge 네트워크 연결

```
[root@dockeredu ~]# docker network connect whatap-bridge whatapguestbook2
```

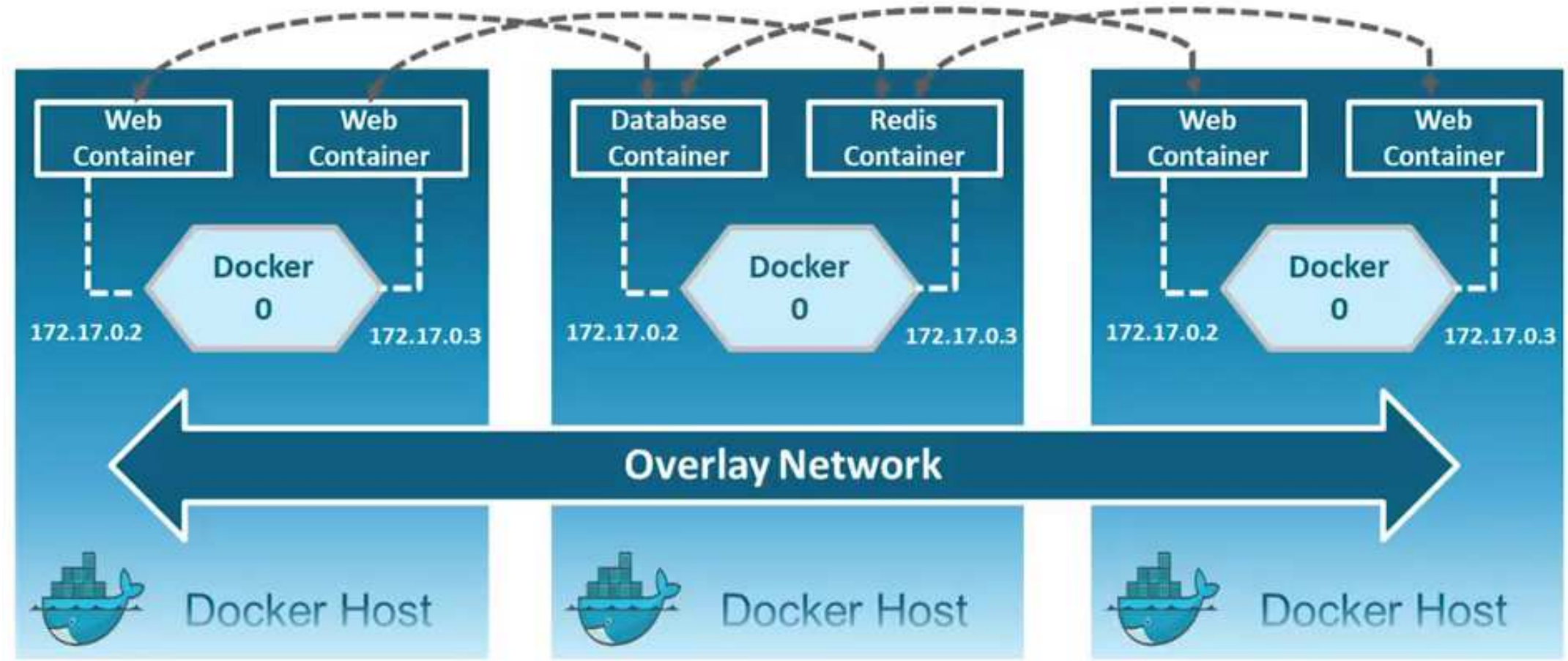
⑩ http://172.17.0.2:8080 으로 접속하여 게시글을 작성하고 Whatap APM에서 서비스 모니터링이 되는지 확인하고 모니터링이 잘 된다면 whatapguestbook2 컨테이너의 IP 정보 조회

```
[root@dockeredu ~]# docker container exec -it whatapguestbook2 bash
root@dc420701c0ad:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
109: eth0@if110: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
111: eth1@if112: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:1b:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.27.0.3/16 brd 172.27.255.255 scope global eth1
       valid_lft forever preferred_lft forever
```

런타임에 동적으로 IP가 추가되었음

overlay network

- **멀티 도커 호스트간 네트워크 연결**
  - swarm이나 kubernetes같은 컨테이너 오케스트레이션에서 사용



컨테이너 모니터링 및 자원 관리

컨테이너 모니터링 (1/3)

▪ 컨테이너 상태 통계 표시

▸ Usage: docker container stats [OPTIONS] [CONTAINER...]

옵션		설명
-a	--all	✓ Show all containers (default shows just running)
	--no-stream	✓ Disable streaming stats and only pull the first result
	--no-trunc	✓ Don't truncate output

```
[root@dockeredu ~]# docker container run -d --name T01 -p 7070:8080 tomcat:6
ee947812fb5e3dce2ca78928d633786427f4d4313eeeb1e0d179dfc0f7b44fa5

[root@dockeredu ~]# docker container stats --no-stream T01
CONTAINER ID   NAME   CPU %   MEM USAGE / LIMIT     MEM %   NET I/O           BLOCK I/O   PIDS
95f97d8f13b3   T01    0.26%   144.6MiB / 3.858GiB   3.66%   73.5kB / 1.35MB   0B / 0B     48
```

▪ 명령결과 설명

항목	설명
CONTAINER ID	컨테이너 ID
NAME	컨테이너 이름
CPU %	CPU 사용률
MEM USAGE/LIMIT	메모리 사용량 / 컨테이너에서 사용할 수 있는 메모리 제한

항목	설명
MEM %	메모리 사용률
NET I/O	네트워크 I/O
BLOCK I/O	블록 I/O
PIDS	컨테이너안의 프로세스 개수

컨테이너 모니터링 (2/3)

- **docker container top**
  - 컨테이너안에 실행되는 process 목록을 출력
  - Usage: docker container top CONTAINER [ps OPTIONS]

```
[root@dockeredu ~]# docker container run -d --name=web httpd:2.4
8b6aa7d83c0c10ae939ec6942cb4fffdd363cd5184f8e2d07a7d7a5f23df589d

[root@dockeredu ~]# docker container top web
UID                 PID                 PPID                C                   STIME               TTY
root                17168               17149               0                   00:01               ?
bin                 17213               17168               0                   00:01               ?
bin                 17214               17168               0                   00:01               ?
bin                 17215               17168               0                   00:01               ?
[root@dockeredu ~]#
```

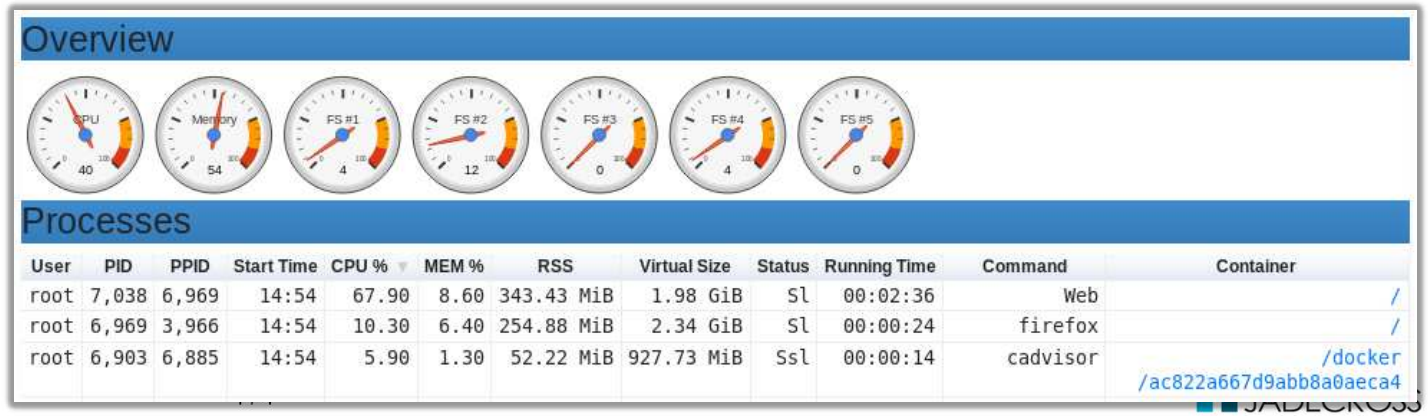
컨테이너 모니터링 (3/3)

- **cadvisor**
  - https://github.com/google/cadvisor
  - 실행중인 컨테이너들의 자원 사용량과 각종 성능 지표를 수집하여 웹으로 모니터링할수 있는 컨테이너 이미지

① 아래 명령으로 cadvisor 컨테이너를 실행

```
[root@dockeredu ~]# docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8090:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:v0.33.0
ac822a667d9abb8a0aeca489852c0a353a9385d67cefae0d99a654ac6d1820a7
[root@dockeredu ~]#
```

② http://locahost:8090 또는 http://192.168.56.91:8090 으로 접속하여 웹에서 컨테이너 자원 모니터링



컨테이너 자원 제한 설정

▪ 메모리 자원 제한 설정 옵션

Option	Description
-m or --memory=	컨테이너가 사용할 수 있는 최대 메모리
--memory-reservation	--memory 값보다 적은 값으로 구성하는 소프트 제한 값 HOST의 메모리 경합이나 메모리가 부족할 때 작동(Swapper가 동작할 때 컨테이너가 최소한 유지할 메모리)
--kernel-memory	컨테이너가 사용할 수 있는 커널 메모리 컨테이너에서 새로운 프로세스가 계속 생성될 때, HOST 커널을 보호하기 위해 컨테이너어의 커널 메모리를 제한하는 효과 -m과 함께 사용시 제한하는 메모리 값에 커널 메모리가 포함됨
--memory-swap*	컨테이너가 디스크로 스왑할 수 있도록 허용된 메모리
--memory-swappiness	컨테이너가 사용하는 어나니모스 페이지 퍼센트로 0-100 값
--oom-kill-disable	OOM 에러가 발생하면 커널이 컨테이너의 프로세스를 죽이지 않는다 이 옵션을 사용하면 OOM killer를 해제한다 . -m 옵션을 설정한 컨테이너에만 이 옵션을 사용한다. -m 옵션을 사용하지 않으면, 호스트는 메모리 부족 문제가 발생할 수 있고 커널은 호스트 시스템의 프로세스를 죽여야 할 수도 있다

▪ CPU 자원 제한 설정 옵션

- 기본적으로 CFS(Completely Fair Scheduler) 스케쥴러에 대한 설정

Option	Description
--cpus=<value>	컨테이너가 사용할 수 있는 CPU 리소스 예) 호스트가 2개의I CPU를 갖는다면 --cpus="1.5" 설정은 최대 CPU 하나 반의 리소스 를 컨테이너에 보장.
--cpuset-cpus	컨테이너가 사용할 수 있는 CPU 나 코어를 제한 예) 0-3: 첫 번째부터 네번째 CPU, 1,3: 두번째와 네번째 CPU
--cpu-shares	컨테이너가 사용하는 CPU 비중을 1024 값을 기반으로 설정 CPU 사용에 병목이 있을 경우에만 soft limit으로 적용

CPU 사용률 제한 데모 (1/3)

① 방명록 어플리케이션 데모 환경

```
@RequestMapping(method = RequestMethod.GET)
public String index(Model model) {
    // 01. 방명록 조회
    model.addAttribute("posts", postService.getAll());
    // 02. 의미없이 CPU 사용량 증가
    this.useCPU(1000);
    return "index";
}

private double useCPU(int loopCount) {
    double result = 0;
    for (int i = 1; i < loopCount; i++) {
        result = i * Math.random() / loopCount;
        for (int j = 1; j < loopCount; j++) {
            result = i * j * Math.random() / loopCount;
            for (int k = 1; k < loopCount; k++) {
                // CPU 만 소모
            }
        }
    }
    return result;
}
```

방명록 웹 어플리케이션의index 페이지는
의미 없이 CPU를 사용하는 코드가 구현되어 있음

② 아래 명령을 이용하여 H2 메모리 DB를 사용하는 방명록 컨테이너를 실행

```
[root@dockeredu ~]# docker container run -d --name=cpulimit01 -p 8080:8080 yu3papa/guestbook_h2:3.0 java -jar /guestbook_H2.jar
cb7f2b5da0924f765e94eb51bf98267148c89956fee9deae2b05a91e453036f9
[root@dockeredu ~]#
```

CPU 사용률 제한 데모 (2/3)

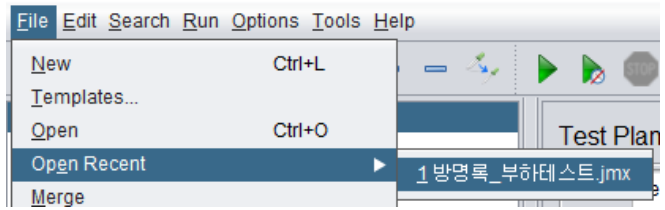
③ docker container stats 명령을 실행하여 cplulimit01 컨테이너의 CPU 사용률 모니터링

```
[root@dockeredu ~]# docker container stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
cb7f2b5da092	cpulimit01	0.28%	268.1MiB / 3.858GiB	6.79%	656B / 0B	0B / 66kB	31

④ jmeter 실행 후 File → Open (/app/jmeter/jmx/방명록_부하테스트.jmx) 를 오픈하고 ▶ 클릭하여 index.html 페이지 부하를 발생시키고 cpulimit01 컨테이너의 CPU 사용률이 70%까지 사용되는것을 모니터링

```
[root@dockeredu ~]# jmeter
```



컨테이너 CPU 사용률이 70% 까지 증가

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
cb7f2b5da092	cpulimit01	68.18%	379.6MiB / 3.858GiB	9.61%	94.5MB / 405MB	1.41MB / 324kB	63

CPU 사용률 제한 데모 (3/3)

⑤ 실행중인 cpulimit01 컨테이너에 --cpus="0.2" 옵션을 적용하여 컨테이너의 CPU 사용률을 동적으로 제한하여 cpulimit01 컨테이너의 CPU 사용률이 20%로 유지되는것을 모니터링

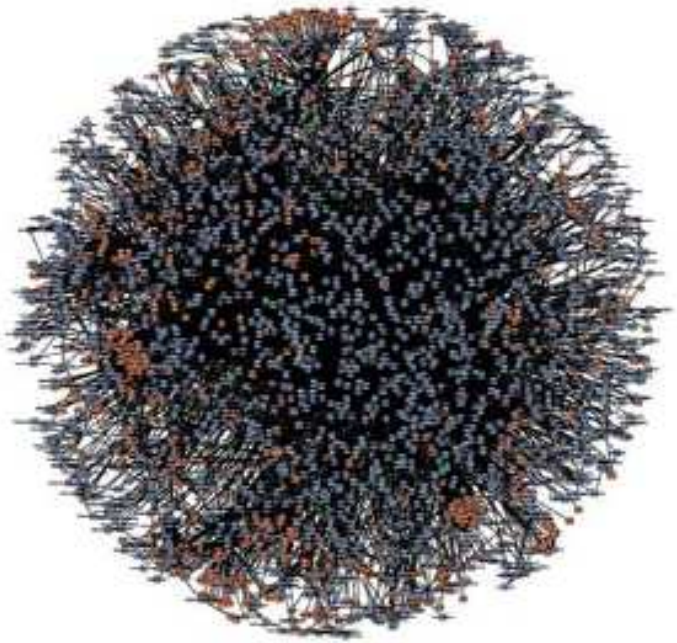
```
[root@dockeredu ~]# docker container update --cpus="0.2" cpulimit01
```

```
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
cb7f2b5da092        cpulimit01          20.25%              379.9MiB / 3.858GiB   9.62%               108MB / 468MB       1.41MB / 340kB      64
```

컨테이너 CPU 사용률이 20% 로 제한되었음

컨테이너 상호 운영

"Death Star" Architecture Diagrams

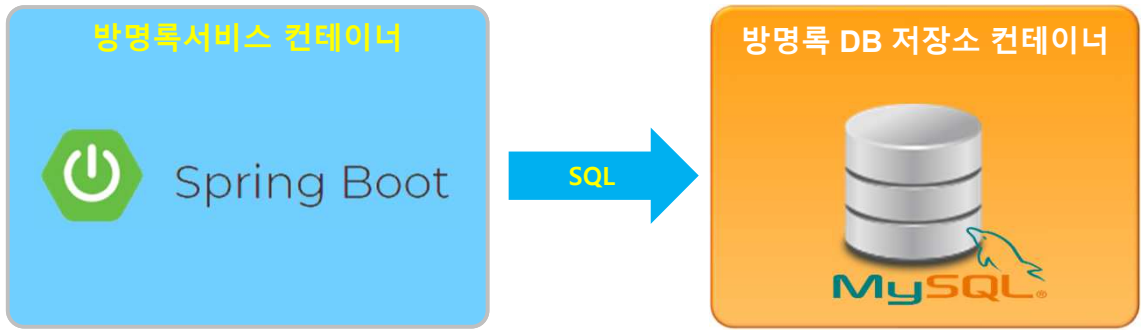


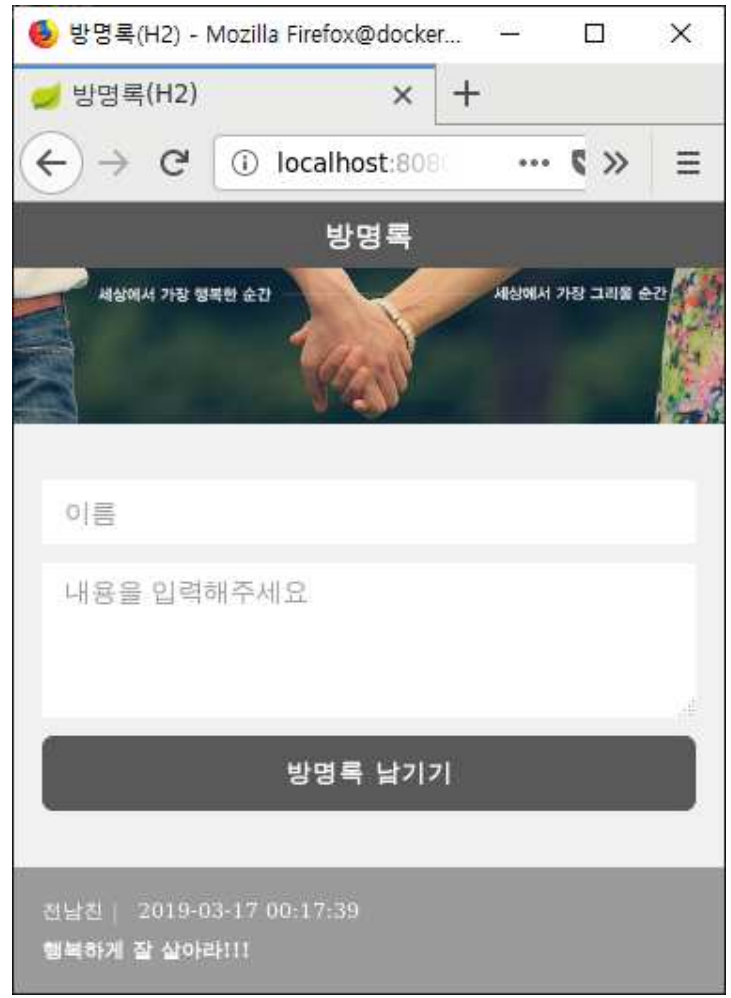
amazon.com

NETFLIX

방명록 샘플 어플리케이션 : WAS와 DB 컨테이너 분리

- **어플리케이션 구조**
  - Spring Boot + MySQL





MySQL 도커 이미지

- https://hub.docker.com/_/mysql

- **How to use this image**
  - $ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag

- **Environment Variables**

none	host
MYSQL_ROOT_PASSWORD	This variable is mandatory and specifies the password that will be set for the MySQL root superuser account. In the above example, it was set to my-secret-pw.
MYSQL_DATABASE	This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to GRANT ALL) to this database.
MYSQL_USER, MYSQL_PASSWORD	These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the MYSQL_DATABASE variable. Both variables are required for a user to be created. Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the MYSQL_ROOT_PASSWORD variable.
MYSQL_ALLOW_EMPTY_PASSWORD	This is an optional variable. Set to yes to allow the container to be started with a blank password for the root user. NOTE: Setting this variable to yes is not recommended unless you really know what you are doing, since this will leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access.
MYSQL_RANDOM_ROOT_PASSWORD	This is an optional variable. Set to yes to generate a random initial password for the root user (using pwgen). The generated root password will be printed to stdout (GENERATED ROOT PASSWORD: .....).
MYSQL_ONETIME_PASSWORD	Sets root (not the user specified in MYSQL_USER!) user as expired once init is complete, forcing a password change on first login. NOTE: This feature is supported on MySQL 5.6+ only. Using this option on MySQL 5.5 will throw an appropriate error during initialization.

환경변수를 이용한 컨테이너 상호작용 (1/5)

- **MySQL 컨테이너 생성**

① mysql:5.7.8 이미지로 mysqldb 컨테이너 생성 - myslq root 사용자 암호는 edu로 설정

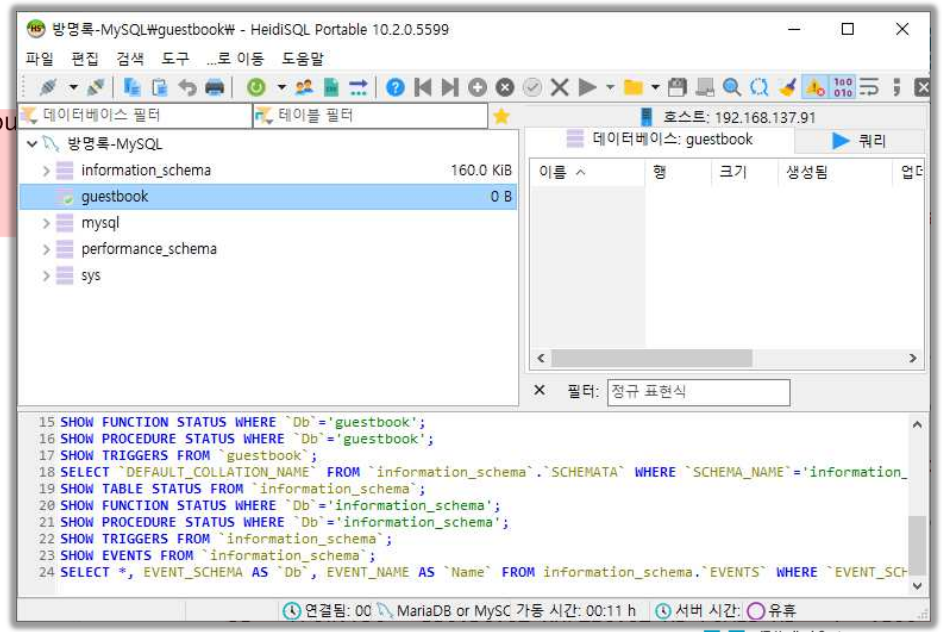
```
[[root@dockeredu ~]# docker container run -d --name=mysqldb -e MYSQL_ROOT_PASSWORD=edu -e MYSQL_DATABASE=guestbook -p 3306:3306
mysql:5.7.8
Unable to find image 'mysql:5.7.8' locally
5.7.8: Pulling from library/mysql
6fa32ff2e443: Pull complete
Digest: sha256:590a950939c4a65ee9146ae23ee5eb2516ec5016a99a8a5308dd39306e67b8e5
Status: Downloaded newer image for mysql:5.7.8
6515ac47ff515a8148ad7ae4c2f7abc46bdb9c0b54bf88ba7d638f38f86badf8
```

② mysqldb 컨테이너에 bash 쉘로 연결 후 컨테이너 IP 확인

```
[root@dockeredu ~]# docker container exec -it mysqldb bash
root@6515ac47ff51:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
35: eth0@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP grou
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

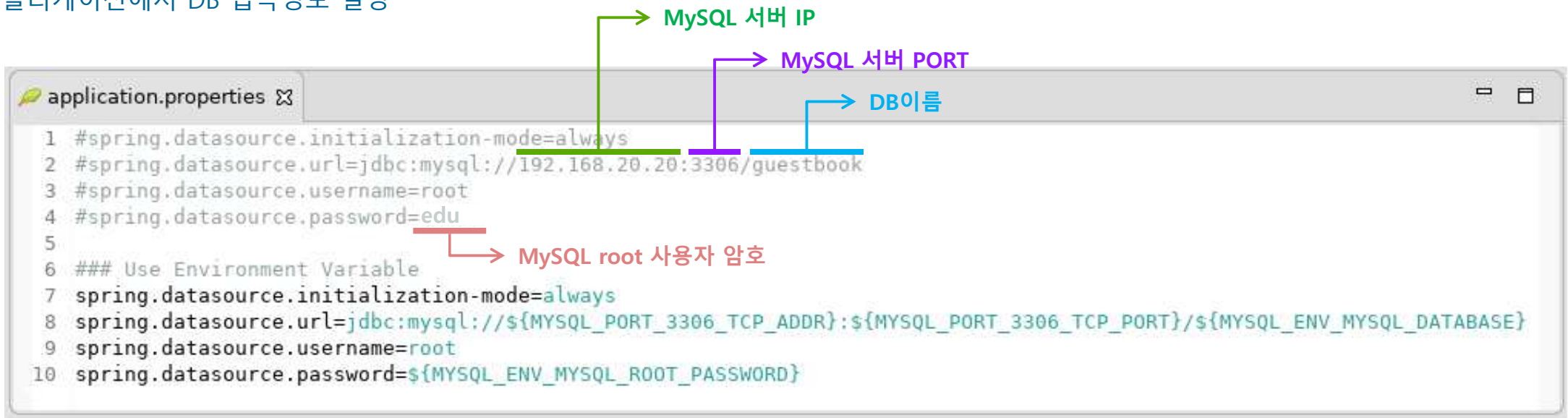
③ mysql 클라이언트를 실행하여 guestbook DB 생성 확인

```
root@6515ac47ff51:/# mysql -uroot -pedu
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| guestbook          |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```



환경변수를 이용한 컨테이너 상호작용 (2/5)

- **mysqldb를 사용하는 guestbook 어플리케이션**
  - 어플리케이션에서 DB 접속정보 설정



  - guestbook 어플리케이션 컨테이너 실행 시 전달해야 하는 환경변수 목록

환경변수	값	비고
MYSQL_PORT_3306_TCP_ADDR	172.17.0.2	MySQL 컨테이너 IP
MYSQL_PORT_3306_TCP_PORT	3306	MySQL 컨테이너에서 DB 서비스 하는 PORT
MYSQL_ENV_MYSQL_DATABASE	guestbook	방명록 DB이름
MYSQL_ENV_MYSQL_ROOT_PASSWORD	edu	MySQL root 사용자 암호

환경변수를 이용한 컨테이너 상호작용 (3/5)

- **mysqldb를 사용하는 guestbook 서비스 컨테이너 생성 후 commit 하여 도커 이미지 생성**

① openjdk:8 이미지로 guestbookMySQL_c01 컨테이너 생성하고 시작

```
[root@dockeredu ~]# docker container run -it --name guestbookMySQL_c01 openjdk:8
6556aa1cdaa1e378245fb1f81ff224bf9a4ed21f3651e663b0c964b1993d1994
[root@dockeredu ~]#
```

ctrl + pq

② guestbook_MYSQL.jar Spring Boot 웹어플리케이션을 guestbookMySQL_c01 컨테이너의 / 폴더에 복사

```
[root@dockeredu ~]# docker container cp ~/guestbook/guestbook_MYSQL.jar guestbookMySQL_c01:/
[root@dockeredu ~]#
```

③ guestbookMySQL_c01 컨테이너 종료 후 commit 하여 yu3papa/guestbook_mysql:1.0 이미지 작성

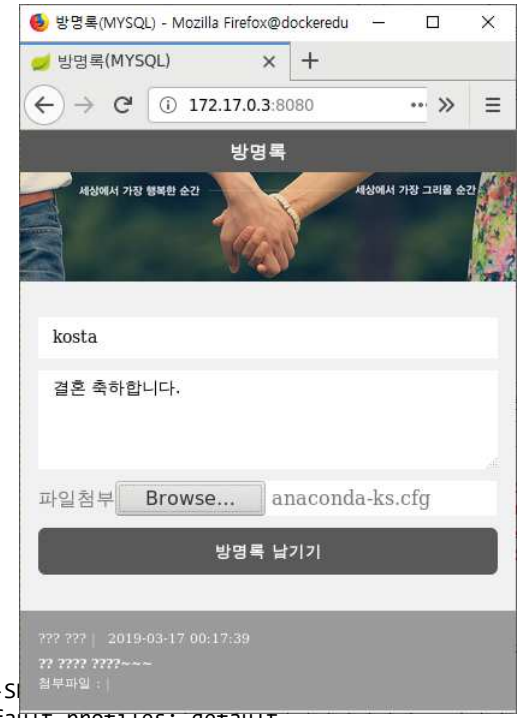
```
[root@dockeredu ~]# docker container stop guestbookMySQL_c01
guestbookMySQL_c01

[root@dockeredu ~]# docker container commit -m "SpringBoot Guestbook(MySQL) created" guestbookMySQL_c01
yu3papa/guestbook_mysql:1.0
sha256:b115754c9f9b5a62c6b7b43c406aa15d6c0212f04099b863fa20ca88bd981076

[root@dockeredu ~]# docker image ls yu3papa/guestbook_mysql
REPOSITORY                TAG                 IMAGE ID            CREATED              SIZE
yu3papa/guestbook_mysql   1.0                 b115754c9f9b        About a minute ago   649MB
```

환경변수를 이용한 컨테이너 상호작용 (4/5)

- **yu3papa/guestbook_mysql:1.0 이미지를 사용하여 guestbook 서비스 컨테이너 생성**
  - 컨테이너 생성시 mysqldb 컨테이너에 접속할 수 있는 환경변수 정보 추가

④ yu3papa/guestbook_mysql:1.0 이미지를 사용하여 guestbook 서비스 컨테이너 생성
→ 컨테이너 생성시 mysqldb 컨테이너에 접속할 수 있는 환경변수 정보 추가

```
# docker container run \
      -e MYSQL_PORT_3306_TCP_ADDR=172.17.0.2 \
      -e MYSQL_PORT_3306_TCP_PORT=3306 \
      -e MYSQL_ENV_MYSQL_DATABASE=guestbook \
      -e MYSQL_ENV_MYSQL_ROOT_PASSWORD=edu \
      --name=guestbook_mysql_c01 \
      yu3papa/guestbook_mysql:1.0 \
      java -jar /guestbook_MYSQL.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)
2019-04-26 06:14:26.703  INFO 1 --- [     main] c.j.guestbook.GuestbookMysqlApplication  : Starting GuestbookMysqlApplication v0.0.1-S
2019-04-26 06:14:26.707  INFO 1 --- [     main] c.j.guestbook.GuestbookMysqlApplication  : No active profile set, falling back to default profiles: default
2019-04-26 06:14:28.942  INFO 1 --- [     main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2019-04-26 06:14:28.989  INFO 1 --- [     main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
```



⑤ guestbook 서비스 컨테이너의 IP를 확인

```
[root@dockeredu ~]# docker container inspect --format="{{.NetworkSettings.Networks.bridge.IPAddress}}" guestbook_mysql_c01
172.17.0.3
```

⑥ 확인된 IP를 통해 웹 브라우저로 접속 후 방명록 글 게시

```
http://172.17.0.3:8080
```

환경변수를 이용한 컨테이너 상호작용 (5/5)

- **게시글이 MySQL DB에 정상적으로 저장되었는지 확인**

⑦ mysqldb 컨테이너에 bash 쉘로 연결 후 mysql 클라이언트를 실행

```
[root@dockeredu ~]# docker container exec -it mysqldb bash

root@6515ac47ff51:/# mysql -uroot -pedu
table information for
mysql> use guestbook
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_guestbook |
+---------------------+
| post                |
+---------------------+
1 row in set (0.00 sec)

mysql> select * from post;
+----+---------+---------------------+-----------------+----------------------------------------+
| id | name    | writeDate           | content         | attachedFile                           |
+----+---------+---------------------+-----------------+----------------------------------------+
|  1 | ??? ??? | 2019-03-17 00:17:39 | ?? ???? ????~~~ | NULL                                   |
|  2 | edu     | 2019-04-26 06:24:47 | ?? ?????.       | 20190426062447844_edu_anaconda-ks.cfg  |
+----+---------+---------------------+-----------------+----------------------------------------+
2 rows in set (0.00 sec)
```

한글 저장이 비정상

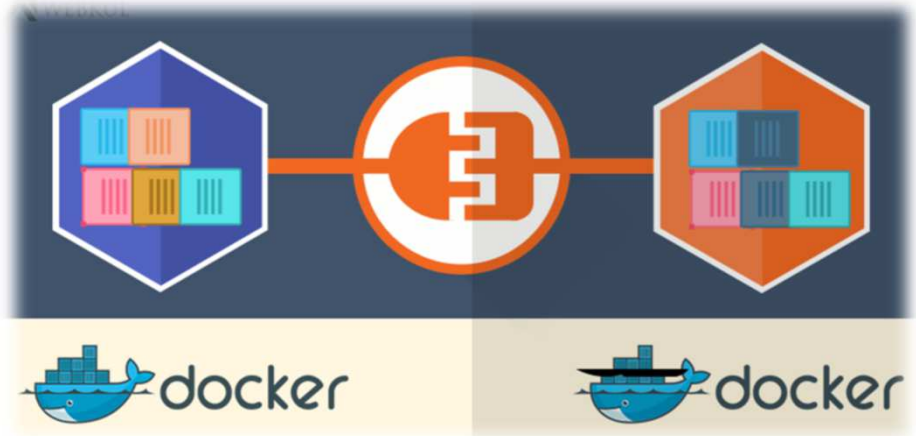
Legacy container links

- **컨테이너 링크**
  - docker container run --link <원본컨테이너>:<앨리어스> 형태로 사용

- **컨테이너 링크시 전달 되는 정보 (원본컨테이너 → 대상컨테이너)**
  - 환경변수 목록
    - <앨리어스>_ENV_환경변수
  - 오픈한 포트 정보
    - <앨리어스>_PORT_<포트번호>_<프로토콜> → tcp://172.17.0.2:3306
    - <앨리어스>_PORT_<포트번호>_<프로토콜>_ADDR → 172.17.0.2
    - <앨리어스>_PORT_<포트번호>_<프로토콜>_PORT → 3306
    - <앨리어스>_PORT_<포트번호>_<프로토콜>_PROTO → tcp
  - /etc/hosts 파일의 IP와 HOST명

- **주의 사항**
  - 원본 컨테이너 재 시작 시 환경변수 목록과, /etc/hosts 파일 내용은 자동으로 대상 컨테이너에서 갱신되지 않는다

- **--link는 legacy 기술로 언젠가는 사라질 예정**
  - 쿠버네티스, 스웜 같은 오케스트레인션 툴에서는 DNS 기능을 이용



컨테이너 링크를 이용한 컨테이너 상호작용 (1/3)

- **yu3papa/guestbook_mysql:1.0 이미지를 사용하여 guestbook 서비스 컨테이너 생성**
  - 컨테이너 생성시 mysqldb 컨테이너에 대한 링크 정보 추가

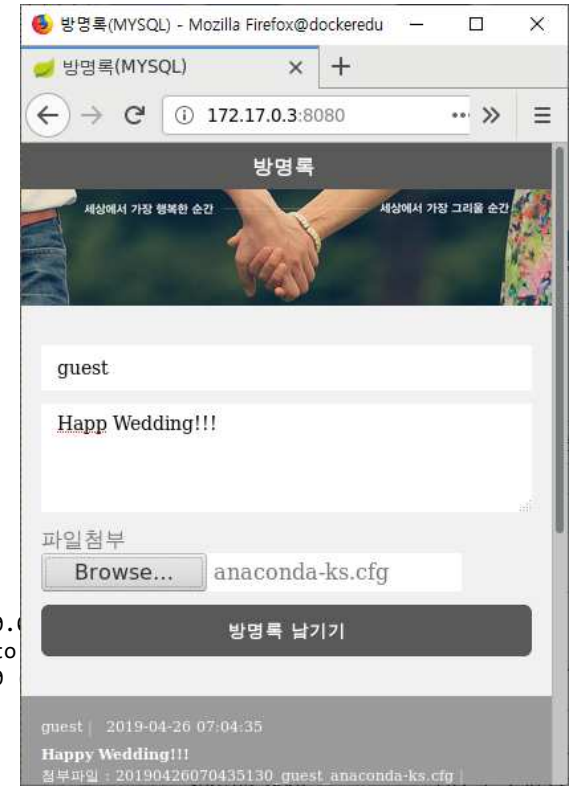
① yu3papa/guestbook_mysql:1.0 이미지를 사용하여 guestbook 서비스 컨테이너 생성
→ 컨테이너 생성시 mysqldb 컨테이너에 접속할 수 있는 환경변수 정보 추가

```
# docker container run \
      --link mysqldb:mysql \
      --name guestbook_mysql_c02 \
      yu3papa/guestbook_mysql:1.0 \
      java -jar /guestbook_MYSQL.jar
```

환경변수를 전달하는 방식에서 컨테이너 링크로 변경

```
docker container run \
    -e MYSQL_PORT_3306_TCP_ADDR=172.17.0.2 \
    -e MYSQL_PORT_3306_TCP_PORT=3306 \
    -e MYSQL_ENV_MYSQL_DATABASE=guestbook \
    -e MYSQL_ENV_MYSQL_ROOT_PASSWORD=kosta \
    --name=guestbook_mysql_c01 \
    yu3papa/guestbook_mysql:1.0 \
    java -jar /guestbook_MYSQL.jar
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2019-04-26 06:14:26.703  INFO 1 --- [     main] c.j.guestbook.GuestbookMysqlApplication  : Starting GuestbookMysqlApplication v0.(
2019-04-26 06:14:26.707  INFO 1 --- [     main] c.j.guestbook.GuestbookMysqlApplication  : No active profile set, falling back to
2019-04-26 06:14:28.942  INFO 1 --- [     main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080
2019-04-26 06:14:28.989  INFO 1 --- [     main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
```



② guestbook 서비스 컨테이너의 IP를 확인

```
[root@dockeredu ~]# docker container inspect --format="{{.NetworkSettings.Networks.bridge.IPAddress}}" guestbook_mysql_c02
172.17.0.3
```

③ 확인된 IP를 통해 웹 브라우저로 접속 후 방명록 글 게시

```
http://172.17.0.3:8080
```

컨테이너 링크를 이용한 컨테이너 상호작용 (2/3)

- **게시글이 MySQL DB에 정상적으로 저장되었는지 확인**

④ mysqldb 컨테이너에 bash 쉘로 연결 후 mysql 클라이언트를 실행

```
[root@dockeredu ~]# docker container exec -it mysqldb bash

root@6515ac47ff51:/# mysql -uroot -pedu
table information for
mysql> use guestbook
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_guestbook |
+---------------------+
| post                |
+---------------------+
1 row in set (0.00 sec)

mysql> select * from post;
+----+---------+---------------------+------------------+----------------------------------------+
| id | name    | writeDate           | content          | attachedFile                           |
+----+---------+---------------------+------------------+----------------------------------------+
|  1 | ??? ??? | 2019-03-17 00:17:39 | ?? ???? ????~~~  | NULL                                   |
|  2 | edu     | 2019-04-26 06:24:47 | ?? ?????.        | 20190426062447844_edu_anaconda-ks.cfg  |
|  3 | ??? ??? | 2019-03-17 00:17:39 | ?? ???? ????~~~  | NULL                                   |
|  4 | guest   | 2019-04-26 07:04:35 | Happy Wedding!!! | 20190426070435130_guest_anaconda-ks.cfg |
+----+---------+---------------------+------------------+----------------------------------------+
```

한글 저장이 비정상

컨테이너 링크를 이용한 컨테이너 상호작용 (3/3)

▪ guestbook_mysql_c02 컨테이너의 환경 변수 및 네트워크 정보 확인

⑤ guestbook_mysql_c02 컨테이너에 bash 쉘 실행

```
[root@dockeredu ~]# docker container exec -it guestbook_mysql_c02 bash
root@5eee6af1479a:/#
```

⑥ 컨테이너의 환경 변수 확인

```
root@5eee6af1479a:/# env | sort
HOME=/root
HOSTNAME=5eee6af1479a
JAVA_DEBIAN_VERSION=8u212-b01-1~deb9u1
JAVA_HOME=/docker-java-home
JAVA_VERSION=8u212
LANG=C.UTF-8
MYSQL_ENV_MYSQL_DATABASE=guestbook
MYSQL_ENV_MYSQL_MAJOR=5.7
MYSQL_ENV_MYSQL_ROOT_PASSWORD=edu
MYSQL_ENV_MYSQL_VERSION=5.7.8-rc
MYSQL_NAME=/guestbook_mysql_c02/mysql
MYSQL_PORT=tcp://172.17.0.2:3306
MYSQL_PORT_3306_TCP=tcp://172.17.0.2:3306
MYSQL_PORT_3306_TCP_ADDR=172.17.0.2
MYSQL_PORT_3306_TCP_PORT=3306
MYSQL_PORT_3306_TCP_PROTO=tcp
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
SHLVL=1
TERM=xterm
_=/usr/bin/env
root@5eee6af1479a:/#
```

링크된 mysqldb 컨테이너 환경변수가 링크한 guestbook_mysql_c02 컨테이너에게 전달됨

⑦ /etc/hosts 파일 내용 확인

```
root@5eee6af1479a:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      mysql 6515ac47ff51 mysqldb
172.17.0.3      5eee6af1479a
```

링크된 mysqldb 컨테이너의 IP와 HOST명이 링크한 guestbook_mysql_c02 컨테이너의 hosts 파일에 등록됨

대상컨테이너 실행 명령

```
# docker container run \
      --link mysqldb:mysql \
      --name guestbook_mysql_c02 \
      yu3papa/guestbook_mysql:1.0 \
      java -jar /guestbook_MYSQL.jar
```

Dockerfile

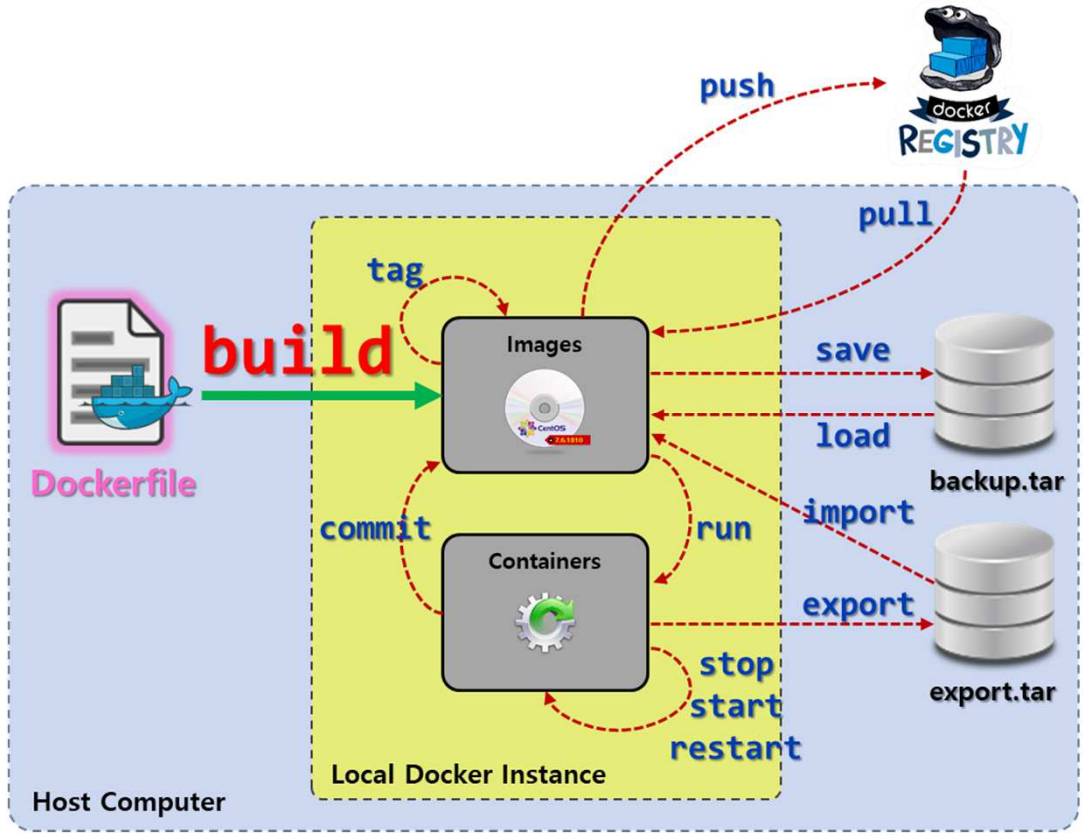
Dockerfile과 docker build

- **Dockerfile**
  - 도커 이미지의 인프라 구성을 기술한 파일

openjdk:8

```
FROM buildpack-deps:stretch-scm
# A few reasons for installing distribution-provided OpenJDK:
RUN apt-get update && apt-get install -y --no-install-recommends bzip2 unzip xz-utils \
              && rm -rf /var/lib/apt/lists/*

# Default to UTF-8 file.encoding
ENV LANG C.UTF-8
# add a simple script that can auto-detect the appropriate JAVA_HOME value
# based on whether the JDK or only the JRE is installed
RUN { \
    echo '#!/bin/sh'; \
    echo 'set -e'; \
    echo; \
    echo 'dirname "$(dirname "$(readlink -f "$(which javac || which java)")")"'; \
    } > /usr/local/bin/docker-java-home \
   && chmod +x /usr/local/bin/docker-java-home
# do some fancy footwork to create a JAVA_HOME that's cross-architecture-safe
RUN ln -svT "/usr/lib/jvm/java-8-openjdk-$(dpkg --print-architecture)" /docker-java-home
ENV JAVA_HOME /docker-java-home
ENV JAVA_VERSION 8u212
ENV JAVA_DEBIAN_VERSION 8u212-b01-1~deb9u1
RUN set -ex; \
# deal with slim variants not having man page directories (which causes "update-alternatives"
to fail)
    if [ ! -d /usr/share/man/man1 ]; then \
        mkdir -p /usr/share/man/man1; \
    fi; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        openjdk-8-jdk="$JAVA_DEBIAN_VERSION" \
    ; \
    rm -rf /var/lib/apt/lists/*; \
# verify that "docker-java-home" returns what we expect
    [ "$(readlink -f "$JAVA_HOME")" = "$(docker-java-home)" ]; \
# update-alternatives so that future installs of other OpenJDK versions don't change
/usr/bin/java
   update-alternatives --get-selections | awk -v home="$(readlink -f "$JAVA_HOME")" 'index($3,
home) == 1 { $2 = "manual"; print | "update-alternatives --set-selections" }'; \
# ... and verify that it actually worked for one of the alternatives we care about
    update-alternatives --query java | grep -q 'Status: manual'
```

- **Dockerfile reference**
  - https://docs.docker.com/engine/reference/builder/
- **docker build**
  - Dockerfile에 기술된 내용대로 도커 이미지 생성



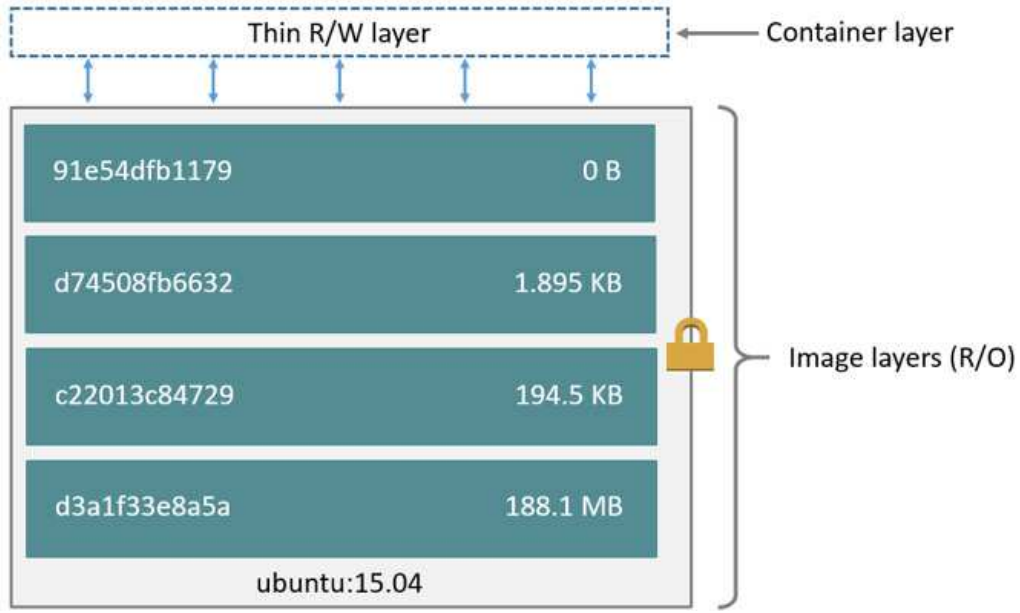
Dockerfile

- **기본 서식**

  명령 인수

  - 명령은 대/소문자 상관없지만 관례적으로 대문자로 사용

- **Best Practice**
  - 설치 구성의 최소화
  - 1 application per 1 container
  - Dockerfile 스크립트 최소화
  - cachinig 방식 유의(cache busting)
  - apt, yum 캐시 제거



이미지 레이어수를 최소화

Dockerfile FROM 명령과 docker image build (1/2)

- **FROM 명령**

```
FROM <image> [AS <name>]
FROM <image>[:<tag>] [AS <name>]
```

  - 베이스 이미지를 명시하는 명령으로 필수항목이며, 다른 명령보다 가장 먼저 나와야 함
  - 태그명을 생략하면 latest가 적용됨
  - 똑같은 [이미지명]:[태그명] 으로 몇번이든 이미지 build가 가능

- **dokcer image build**

```
docker image build -t [생성할 이미지명]:[태그명] [Dockerfile의 위치]
```

  - Dockerfile로부터 이미지를 생성하는 명령

Dockerfile FROM 명령과 docker image build (2/2)

① 이미지 작업할 폴더를 생성하고 이동한 후 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/from && cd $_
[root@dockeredu buildlab]# vi Dockerfile
```

```
# 베이스 이미지 지정
FROM openjdk:8
```

② docker build 명령을 실행하여 sample:1.0 이미지 작성

```
[root@dockeredu buildlab]# docker build -t from:1.0 .
Sending build context to Docker daemon  2.048kB
Step 1/1 : FROM openjdk:8
 ---> b8d3f94869bb
Successfully built b8d3f94869bb
Successfully tagged sample:1.0
```

③ from:1.0 이미지가 생성되었는지 확인

Base Image에서 변경된 부분이 없어 Image ID가 동일함

```
[root@dockeredu buildlab]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
openjdk             8                   b8d3f94869bb        4 weeks ago         625MB
from                1.0                 b8d3f94869bb        4 weeks ago         625MB
```

④ from:1.0 이미지로부터 컨테이너가 정상적으로 동작하는지 확인

```
[root@dockeredu buildlab]# docker container run -it from:1.0
root@8a39832dcfc7:/# date
Sun Apr 28 06:58:18 UTC 2019  ← TimeZone이 UTC로 되어 있음

root@8a39832dcfc7:/# ifconfig
bash: ifconfig: command not found
root@8a39832dcfc7:/#
```

Dockerfile : RUN (1/4)

▪ RUN 명령

```
RUN <command>
RUN ["executable", "param1", "param2"]
```

← *shell form*

← *exec from*

- 현재 이미지의 탑 레이어에서 명령을 실행하고 commit한다. → commit하기 때문에 새로운 레이어가 생성됨
- RUN 스텝마다 cache에 저장하여 빠른 수행
  - --no-cache 옵션으로 캐시 사용 불가 설정 가능

▪ shell form -vs- exec form

	shell form	exec form
예	RUN apt-get update RUN apt-get install -y net-tools	RUN ["/usr/bin/apt-get", "update"] RUN ["/usr/bin/apt-get", "install -y net-tools"]
설명	디폴트 쉘(/bin/sh -c)에서 명령 실행 - SHELL 명령을 사용하여 디폴트 쉘 변경 가능	쉘을 경유하지 않고 명령어 자체를 직접 실행 - JSON 타입으로 docker daemeon에게 명령 전달
변수 확장	변수 확장 가능	환경변수 확장 안됨(다른 방법이 있기는 함) → 문자열로 인식
레이어수 및 가독성	쉘을 이용하기 때문에 여러 명령을 한 줄에 쓰기 가능 `RUN apt-get update; apt-get install -y net-tools; apt-get clean` *==> 아래는 동일한 명령으로 가독성을 높여줌* `RUN apt-get update; \` `  apt-get install -y net-tools; \` `  apt-get clean`	여러 명령을 한 줄에 쓰기 가능하지만 가독성이 떨어짐 `RUN ["/bin/bash", "-c", "apt-get update; apt-get install -y net-tools; apt-get clean"]` *==> 아래는 동일한 명령으로 가독성을 높여줌* `RUN ["/bin/bash", "-c", "apt-get update;\` `  apt-get install -y net-tools;\` `  apt-get clean"]`

Dockerfile : RUN (2/4)

① 이미지 작업할 폴더를 생성하고 이동한 후 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/run && cd $_
[root@dockeredu run]# vi Dockerfile
```

```
FROM openjdk:8

# exec form
RUN ["/usr/bin/apt-get", "update"]

# shell from
RUN apt-get install -y net-tools; \
    apt-get clean; \
    rm -rf /var/lib/apt/lists/*
```

② docker build 명령을 실행하여 run:1.0 이미지 작성

```
[root@dockeredu run]# docker build --no-cache -t run:1.0 .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM openjdk:8
 ---> b8d3f94869bb
Step 2/4 : RUN ["/usr/bin/apt-get", "update"]
 ---> Running in 9b985b483084
Ign:1 http://deb.debian.org/debian stretch InRelease
~~~
Get:8 http://security.debian.org/debian-security stretch/updates/main amd64 Packages [485 kB]
Fetched 7884 kB in 1s (4255 kB/s)
Reading package lists...
Removing intermediate container 9b985b483084
 ---> 5376c212c217
Step 3/4 : RUN apt-get install -y net-tools;     apt-get clean
 ---> Running in 492195d4b280
Reading package lists...
~~~
Setting up net-tools (1.60+git20161116.90da8a0-1) ...
Removing intermediate container 492195d4b280
 ---> ef41989231ca
Successfully built 434defa4e392
Successfully tagged run:1.0
```

Dockerfile : RUN (3/4)

③ run:1.0 이미지가 생성되었는지 확인

```
[root@dockeredu run]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
run                 1.0                 ddd835aa82d4        3 seconds ago       644MB
<none>              <none>              ab5925c0ddc7        4 seconds ago       644MB
<none>              <none>              4f64f0484f98        6 seconds ago       641MB
openjdk             8                   b8d3f94869bb        4 weeks ago         625MB
```

중간 이미지가 생기는 이유는?

④ run:1.0 이미지로부터 컨테이너가 정상적으로 동작하는지 확인

```
[root@dockeredu run]# docker container run -it --rm run:1.0

root@e02178fef65c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 6  bytes 516 (516.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Dockerfile : RUN (4/4)

⑤ docker image history 명령을 사용하여 image 생성시 명령을 추적

```
[root@dockeredu run]# docker image history run:1.0
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
ddd835aa82d4	7 minutes ago	/bin/sh -c mv /etc/localtime /etc/localtime_…	57B	← shell form
ab5925c0ddc7	7 minutes ago	/bin/sh -c apt-get install -y net-tools; …	2.61MB	
4f64f0484f98	7 minutes ago	/usr/bin/apt-get update	16.3MB	← exec form
b8d3f94869bb	4 weeks ago	/bin/sh -c set -ex; if [! -d /usr/share/m…	349MB	쉘을 경유하지 않음
<missing>	4 weeks ago	/bin/sh -c #(nop) ENV JAVA_DEBIAN_VERSION=8…	0B	
<missing>	4 weeks ago	/bin/sh -c #(nop) ENV JAVA_VERSION=8u212	0B	
<missing>	4 weeks ago	/bin/sh -c #(nop) ENV JAVA_HOME=/docker-jav…	0B	
<missing>	4 weeks ago	/bin/sh -c ln -svT "/usr/lib/jvm/java-8-open…	33B	
<missing>	4 weeks ago	/bin/sh -c { echo '#!/bin/sh'; echo 'set…	87B	
<missing>	4 weeks ago	/bin/sh -c #(nop) ENV LANG=C.UTF-8	0B	
<missing>	4 weeks ago	/bin/sh -c apt-get update && apt-get install…	2.21MB	
<missing>	4 weeks ago	/bin/sh -c apt-get update && apt-get install…	142MB	
<missing>	4 weeks ago	/bin/sh -c set -ex; if ! command -v gpg > /…	7.81MB	
<missing>	4 weeks ago	/bin/sh -c apt-get update && apt-get install…	23.2MB	
<missing>	4 weeks ago	/bin/sh -c #(nop) CMD ["bash"]	0B	
<missing>	4 weeks ago	/bin/sh -c #(nop) ADD file:843b8a2a9df1a0730…	101MB	

Dockerfile : COPY 와 ADD (1/3)

- **COPY 명령**

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

- 파일 또는 디렉토리를 복사하는 명령
- src
  - 빌드 작업 디렉토리를 기준으로 상대경로
  - 와일드 카드(*,?)를 사용할 수 있음
- dest
  - 절대경로와 상대경로 사용
  - 상대경로 사용시 WORKDIR 명령 필요
  - 대상경로가 존재하지 않으면 생성 후 복사

- **ADD 명령**

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

- COPY 명령과 기본적으로 동일한 기능 수행
- COPY 명령과 차이점
  - 네트워크 (URL)를 통해 파일 복사 가능
  - 압축 파일은 대상 경로에서 압축 해제

Dockerfile : COPY 와 ADD (2/3)

① 이미지 작업할 폴더를 생성하고 이동한 후 guestbook_H2.jar 파일 복사하고 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/copyadd && cd $_
[root@dockeredu copyadd]# cp ~/guestbook/guestbook_H2.jar .
[root@dockeredu copyadd]# vi Dockerfile
```

```
FROM openjdk:8
RUN apt-get update; \
    apt-get install -y net-tools; \
    apt-get clean; \
    rm -rf /var/lib/apt/lists/*

COPY guestbook_H2.jar /app/
ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
RUN tar xfz scouter-min-2.6.1.tar.gz; \
    echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf; \
    rm -fr scouter-min-2.6.1.tar.gz
```

원격지에서 scouter agent 복사

② docker build 명령을 실행하여 copyadd:1.0 이미지 작성

```
[root@dockeredu copyadd]# docker build -t copyadd:1.0 .
Sending build context to Docker daemon  23.42MB
Step 1/5 : FROM openjdk:8
 ---> b8d3f94869bb
Step 2/5 : RUN apt-get update;     apt-get install -y net-tools;     apt-get clean;     mv /etc/localtime /etc/localtime_org;     ln -s
/usr/share/zoneinfo/Asia/Seoul /etc/localtime
 ---> Using cache
 ---> 64528ea8e8dd
Step 3/5 : COPY guestbook_H2.jar /app/
 ---> Using cache
 ---> 862b68ca2262
Step 4/5 : ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
Downloading [==================================================>]  22.05MB/22.05MB
 ---> Using cache
 ---> 82441e353895
Step 5/5 : RUN tar xfz scouter-min-2.6.1.tar.gz;     echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf;     rm -fr scouter-min-
2.6.1.tar.gz
 ---> Running in 768d292af7e3
Removing intermediate container 768d292af7e3
 ---> 7a42ba9f3a0e
Successfully built 7a42ba9f3a0e
Successfully tagged copyadd:1.0
```

Dockerfile : COPY 와 ADD (3/3)

③ copyadd:1.0 이미지로부터 컨테이너를 시작하고 파일들이 복사되었는지 확인

```
[root@dockeredu run]# docker container run -it --rm copyadd:1.0

root@e592f2a8c06d:/# ls -l /app/
total 22876
-rw-r--r-- 1 root root 23421760 Apr 28 16:53 guestbook_H2.jar

root@e592f2a8c06d:/# ls -l /scouter
total 0
drwxr-xr-x 4 root root 57 Mar 17 17:48 agent.java
root@e592f2a8c06d:/#
```

Dockerfile : ENV 와 LABEL (1/3)

ENV 명령

```
ENV <key> <value>
ENV <key>=<value> ...
```

- 환경변수를 설정하는 명령
- ENV 명령으로 지정한 환경변수는 컨테이너 실행시에 --env 옵션으로 변경 가능

	<key> <value> 형태로 지정하는 경우	<key>=<value> ... 지정하는 경우
예	ENV myName "HwanYeoul, Jeong" ENV myOrder Gin Whisky Calvados ENV myNickName yu3papa	ENV myName="HwanYeoul, Jeong" \ myOrder=Gin\ Whisky\ Calvados \ myNickName=yu3papa
특징	한번에 하나의 환경변수만 지정 가능	한번에 여러 개의 환경변수 지정 가능

LABEL 명령

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

- 이미지에 작성자 정보, 버전과 같은 부가 정보를 제공할 때 사용
- docker image inspect 명령으로 확인
- 예)
  - LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>"
  - LABEL title="guestbook App"
  - LABEL version="1.0"
  - LABEL description="This image is guestbook service"

Dockerfile : ENV 와 LABEL (2/3)

① 이미지 작업할 폴더를 생성하고 이동한 후 guestbook_H2.jar 파일 복사하고 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/envlabel && cd $_
[root@dockeredu envlabel]# cp ~/guestbook/guestbook_H2.jar .
[root@dockeredu envlabel]# vi Dockerfile
```

```
FROM openjdk:8
RUN apt-get update; \
    apt-get install -y net-tools; \
    apt-get clean; \
    rm -rf /var/lib/apt/lists/*
COPY guestbook_H2.jar /app/
ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
RUN tar xfz scouter-min-2.6.1.tar.gz; \
    echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf; \
    rm -fr scouter-min-2.6.1.tar.gz
LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>" \
      title="guestbook App" \
      version="1.0" \
      description="This image is guestbook service"

ENV APP_HOME /app
```

← *LABEL 설정*

← *ENV 설정*

② docker build 명령을 실행하여 **envlabel**:1.0 이미지 작성

```
[root@dockeredu copyadd]# docker build -t envlabel:1.0 .
Sending build context to Docker daemon  23.42MB
Step 1/5 : FROM openjdk:8
 ---> b8d3f94869bb
~~~
Step 6/7 : LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>"       title="guestbook App"       version="1.0"       description="This
image is guestbook service"
 ---> Running in 7b5faa61500e
Removing intermediate container 7b5faa61500e
 ---> a1f2fc994751
Step 7/7 : ENV APPHOME="/app"
 ---> Running in ca9a9c4857ae
Removing intermediate container ca9a9c4857ae
 ---> 6427121c4232
Successfully built 6427121c4232
Successfully tagged envlabel:1.0
```

Dockerfile : ENV 와 LABEL (3/3)

③ envlabel:1.0 이미지에 LABEL 정보가 저장되었는지 확인

```
[root@dockeredu envlabel]# docker image inspect envlabel:1.0 | grep -A4 Labels
            "Labels": {
                "description": "This image is guestbook service",
                "maintainer": "HwanYeoul Jeong<coordinatorj@jadecross.com>",
                "title": "guestbook App",
                "version": "1.0"
```

④ envlabel:1.0 이미지로부터 컨테이너를 시작하고 APP_HOME 환경변수가 적용되었는지 확인

```
[root@dockeredu envlabel]# docker container run -it --rm envlabel:1.0
root@946c61ac7b53:/# env
LANG=C.UTF-8
HOSTNAME=946c61ac7b53
APP_HOME=/app
JAVA_HOME=/docker-java-home
JAVA_VERSION=8u212
PWD=/
HOME=/root
JAVA_DEBIAN_VERSION=8u212-b01-1~deb9u1
TERM=xterm
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
```

Dockerfile : EXPOSE 와 VOLUME(1/3)

- **EXPOSE 명령**

```
EXPOSE <port> [<port>/<protocol>...]
```

  - 컨테이너의 공개 포트 번호를 지정할 때 사용
  - 도커 엔진에게 실행중인 컨테이너가 Listen하고 있는 PORT를 알려주며, -p 옵션을 사용하여 어떤 PORT를 호스트에 공개할지를 정의
  - 프로토콜을 명시하지 않으면 디폴트인 TCP 프로토콜이 적용됨
  - 예)
    - `EXPOSE 8080`

- **VOLUME 명령**

```
VOLUME ["/mount_point"]
```

  - 지정한 이름의 마운트 포인트를 작성하고, 호스트나 그 외 다른 컨테이로부터 볼륨의 외부 마운트를 수행
  - 예)
    - `VOLUME /upload`

Dockerfile : EXPOSE 와 VOLUME(2/3)

① 이미지 작업할 폴더를 생성하고 이동한 후 guestbook_H2.jar 파일 복사하고 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/exposevolume && cd $_
[root@dockeredu exposevolume]# cp ~/guestbook/guestbook_H2.jar .
[root@dockeredu exposevolume]# vi Dockerfile
```

```
FROM openjdk:8
RUN apt-get update; \
    apt-get install -y net-tools; \
    apt-get clean; \
    rm -rf /var/lib/apt/lists/*
COPY guestbook_H2.jar /app/
ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
RUN tar xfz scouter-min-2.6.1.tar.gz; \
    echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf; \
    rm -fr scouter-min-2.6.1.tar.gz
LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>" \
      title="guestbook App" \
      version="1.0" \
      description="This image is guestbook service"
ENV APP_HOME /app

EXPOSE 8080
VOLUME /upload
```

← *8080 포트 리슨*

← */upload 폴더 마운트*

② docker build 명령을 실행하여 envlabel:1.0 이미지 작성

```
[root@dockeredu exposevolume]# docker build -t exposevolume:1.0 .
Sending build context to Docker daemon  23.43MB
Step 1/9 : FROM openjdk:8
 ---> b8d3f94869bb
~~~
Step 8/9 : EXPOSE 8080
 ---> Using cache
 ---> a35632dd3c03
Step 9/9 : VOLUME /upload
 ---> Using cache
 ---> 04c36aa117f2
Successfully built 04c36aa117f2
Successfully tagged exposevolume:1.0
```

Dockerfile : EXPOSE 와 VOLUME(3/3)

③ exposevolume:1.0 이미지로부터 컨테이너를 시작하고 /upload 폴더가 마운트 되었는지 확인

```
[root@dockeredu exposevolume]# docker container run -it --rm -P exposevolume:1.0
root@9748e3379505:/# mount | grep upload
/dev/sda3 on /upload type xfs (rw,relatime,attr2,inode64,noquota)
root@9748e3379505:/#
root@9748e3379505:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          96G  4.6G   91G   5% /
tmpfs            64M     0   64M   0% /dev
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/sda3        96G  4.6G   91G   5% /upload
shm              64M     0   64M   0% /dev/shm
tmpfs           2.0G     0  2.0G   0% /proc/acpi
tmpfs           2.0G     0  2.0G   0% /proc/scsi
tmpfs           2.0G     0  2.0G   0% /sys/firmware
```

④ ctrl+p,q 를 입력하여 컨테이너에서 나온후 docker container ls 명령으로 8080 포트가 호스트에 매핑 되었는지 확인

```
[root@dockeredu exposevolume]# docker container ls
CONTAINER ID     IMAGE               COMMAND      CREATED             STATUS             PORTS                     NAMES
9748e3379505     exposevolume:1.0    "bash"       2 minutes ago       Up 2 minutes       0.0.0.0:32768->8080/tcp   compassionate_lamport
```

Dockerfile : CMD 와 ENTRYPOINT (1/4)

- **CMD 명령**

```
CMD ["executable","param1","param2"]
CMD ["param1","param2"]
CMD command param1 param2
```

← *exec form, this is the preferred form*
← *as default parameters to ENTRYPOINT*
← *shell form*

  - Dockerfile 내에 1번만 기술할 수 있고, 여러 번 기술되면 마지막 CMD 명령이 적용됨
  - ENTRYPOINT 명령과 함께 사용되면 ENTRYPOINT의 파라미터값으로 전달됨
  - 컨테이너 실행 시 오버라이드 할 수 있음
    - busybox - CMD ["sh"]
    - maven - CMD ["mvn"]
    - centos - CMD ["/bin/bash"]

✓ 컨테이너의 1번 프로세스가 될 명령을 지정
✓ 이미지 제작자가 1번 프로세스의 변경을 허용하고 싶지 않을 때 ENTRYPOINT 사용

- **ENTRYPOINT 명령**

```
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT command param1 param2
```

← *exec form, preferred*
← *shell form*

  - 컨테이너 시작 시 자동으로 실행되는 명령을 지정할 때 사용
  - CMD 명령과 함께 사용되면 CMD 명령의 값이 ENTRYPOINT 명령의 파라미터로 전달 됨
  - CMD 명령과 다르게 컨테이너 실행 시 오버라이드 할 수 없음(--entrypoint 옵션으로 변경 가능)
    - mysql : ENTRYPOINT ["docker-entrypoint.sh"]
    - postgress : ENTRYPOINT ["docker-entrypoint.sh"]

Dockerfile : CMD 와 ENTRYPOINT (2/4)

- **exec**
  - Bash Built-In Command

```
[root@dockeredu ~]# help -m exec
NAME
    exec - Replace the shell with the given command.

SYNOPSIS
    exec [-cl] [-a name] [command [arguments ...]] [redirection ...]

DESCRIPTION
    Replace the shell with the given command.

    Execute COMMAND, replacing this shell with the specified program.
    ARGUMENTS become the arguments to COMMAND.  If COMMAND is not specified, any redirections take effect in the current shell.

    Options:
      -a name   pass NAME as the zeroth argument to COMMAND
      -c                execute COMMAND with an empty environment
      -l                place a dash in the zeroth argument to COMMAND

    If the command cannot be executed, a non-interactive shell exits, unless the shell option `execfail' is set.

    Exit Status:
    Returns success unless COMMAND is not found or a redirection error occurs.

SEE ALSO
    bash(1)

IMPLEMENTATION
    GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu)
    Copyright (C) 2011 Free Software Foundation, Inc.
    License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

Dockerfile : CMD 와 ENTRYPOINT (3/4)

① 이미지 작업할 폴더를 생성하고 이동한 후 guestbook_H2.jar 파일 복사하고 Dockerfile 편집

```
[root@dockeredu ~]# mkdir -p ~/buildlab/cmdentrypoint && cd $_
[root@dockeredu cmdentrypoint]# cp ~/guestbook/guestbook_H2.jar .
[root@dockeredu cmdentrypoint]# vi Dockerfile
```

```
FROM openjdk:8
RUN apt-get update; \
    apt-get install -y net-tools; \
    apt-get clean; \
    rm -rf /var/lib/apt/lists/*
COPY guestbook_H2.jar /app/
ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
RUN tar xfz scouter-min-2.6.1.tar.gz; \
    echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf; \
    rm -fr scouter-min-2.6.1.tar.gz
LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>" \
      title="guestbook App" \
      version="1.0" \
      description="This image is guestbook service"
ENV APP_HOME /app
EXPOSE 8080
VOLUME /app/upload
WORKDIR $APP_HOME

ENTRYPOINT ["java"]
CMD ["-javaagent:/scouter/agent.java/scouter.agent.jar", "-jar", "guestbook_H2.jar"]
```

② docker build 명령을 실행하여 envlabel:1.0 이미지 작성

```
[root@dockeredu cmdentrypoint]# docker build -t cmdentrypoint:1.0 .
Sending build context to Docker daemon  23.43MB
Step 1/12 : FROM openjdk:8
 ~~~
Step 11/12 : ENTRYPOINT ["java"]
 ---> Using cache
 ---> 03938ad43032
Step 12/12 : CMD ["-javaagent:/scouter/agent.java/scouter.agent.jar", "-jar", "guestbook_H2.jar"]
 ---> Using cache
 ---> 7ed41e4acae8
Successfully built 7ed41e4acae8
Successfully tagged cmdentrypoint:1.0
```

Dockerfile : CMD 와 ENTRYPOINT (4/4)

③ 스카우터 수집서버를 시작

```
[root@dockeredu ~]# scouter.server.boot
nohup: redirecting stderr to stdout

 ____                                _
/ ___|  ___ ___  _   _| |_ ___ _ __
\___ \ / __/   \| | | | __/ _ \ '__|
 ___) | (_| (+) | |_| | ||  __/ |
|____/ \___\___/ \__,_|\__\___|_|
Open Source S/W Performance Monitoring
Scouter version 2.6.
```

④ 스카우터 클라이언트 시작

```
[root@dockeredu ~]# scouter.client
[1] 28451
```

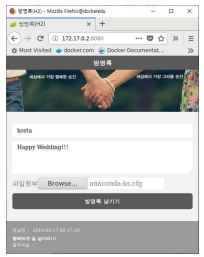
⑤ cmdentrypoint:1.0 이미지로부터 컨테이너 시작

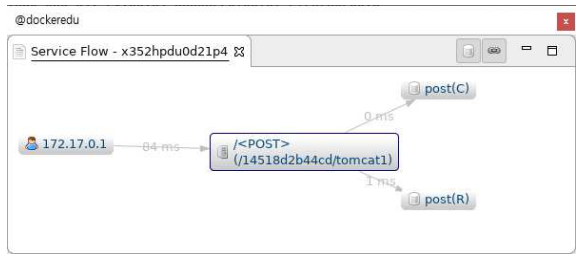
```
[root@dockeredu ~]# docker container run -it cmdentrypoint:1.0
 ____                                _
/ ___|  ___ ___  _   _| |_ ___ _ __
\___ \ / __/   \| | | | __/ _ \ '__|
 ___) | (_| (+) | |_| | ||  __/ |
|____/ \___\___/ \__,_|\__\___|_|
Open Source S/W Performance Monitoring
Scouter version 2.6.1

20190428 11:14:22 [SCOUTER] Version 2.6.1 2019-03-17 08:45 GMT_ENV_java8plus
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

20190428 11:14:23 [A119] Agent UDP local.port=0
```

- **참고) Spring Boot with Docker**
  - https://spring.io/guides/gs/spring-boot-docker/

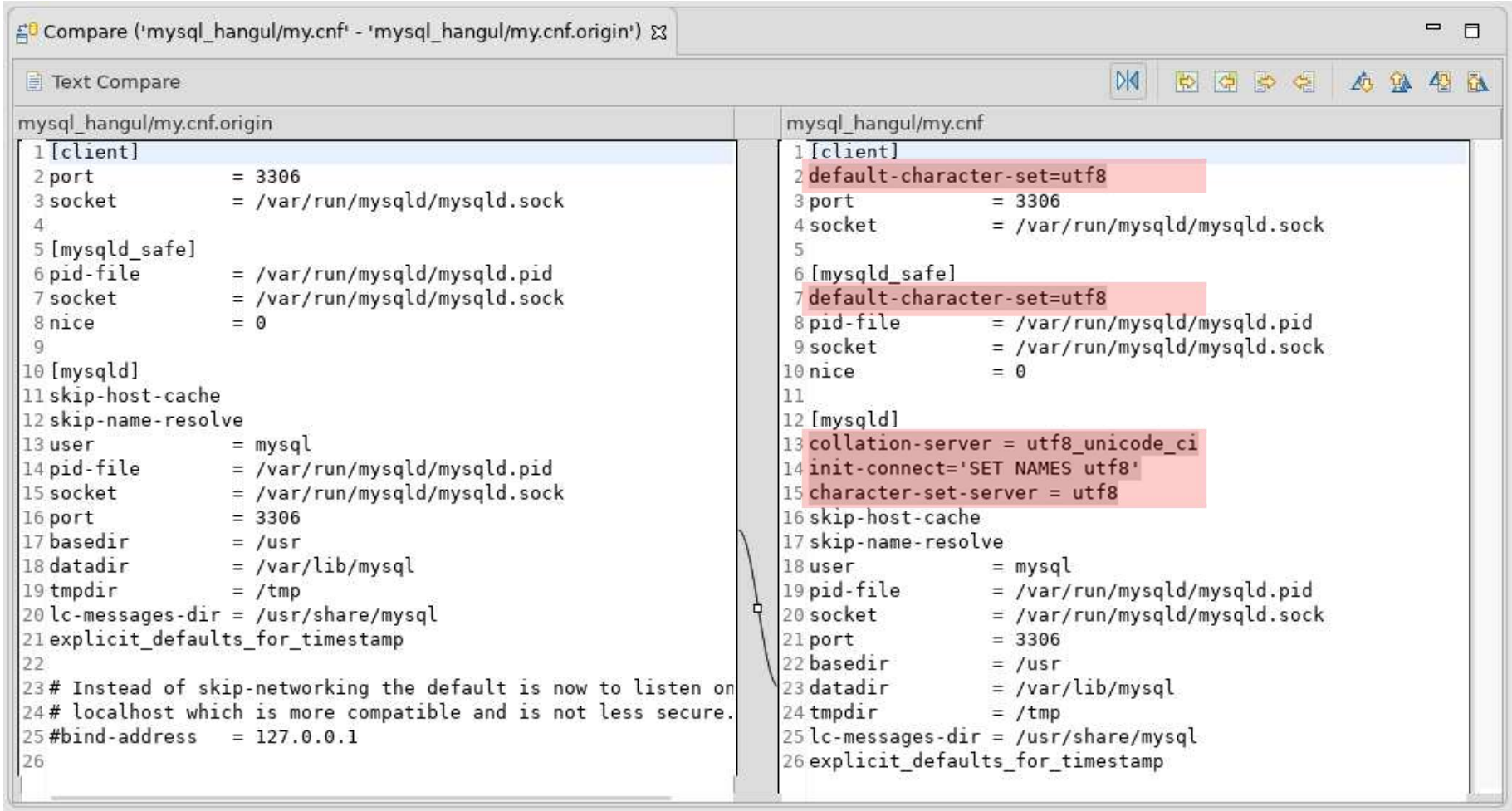
⑥ 웹 브라우저로 http://172.17.0.2:8080 으로 접속 후 방명록에 글을 게시한 후 정상 저장되는지 확인하고, 스카우터에서 모니터링 되는 지 확인





LAB : 한글이 지원되는 MySQL DB를 사용하는 방명록 (1/5)

- **MySQL에서 한글 설정을 위해서는 character-set을 utf8으로 설정**
  - MySQL 설정 파일은 /etc/mysql/my.cnf



LAB : 한글이 지원되는 MySQL DB를 사용하는 방명록 (2/5)

① 이미지 작업할 폴더로 이동한 후 Dockerfile 편집

```
[root@dockeredu ~]# cd ~/workspace/mysql_hangul/
[root@dockeredu mysql_hangul]# vi Dockerfile
```

```
FROM mysql:5.7.8

COPY my.cnf /etc/mysql/
```

② docker build 명령을 실행하여 yu3papa/mysql_hangul:1.0 이미지 작성

```
[root@dockeredu mysql_hangul]# docker build -t yu3papa/mysql_hangul:1.0 .
Sending build context to Docker daemon   7.68kB
Step 1/2 : FROM mysql:5.7.8
 ---> adedf30d6136
Step 2/2 : COPY my.cnf /etc/mysql/
 ---> Using cache
 ---> 5ff562d02d1a
Successfully built 5ff562d02d1a
Successfully tagged yu3papa/mysql_hangul:1.0
```

③ yu3papa/mysql_hangul:1.0 이미지로 guestbookdb 컨테이너 시작

```
[root@dockeredu ~]# docker container run -d --name=guestbookdb -e MYSQL_ROOT_PASSWORD=edu -e MYSQL_DATABASE=guestbook
yu3papa/mysql_hangul:1.0
ea2a32bce56237cbd8b02148b444c0cb8bee31925499cea384ccd99d8906ddb5
```

LAB : 한글이 지원되는 MySQL DB를 사용하는 방명록 (3/5)

- **guestbook_MYSQL.jar를 사용하는 SpringBoot용 이미지 만들기**

④ 이미지 작업할 폴더를 생성하고 이동한 후 guestbook_H2.jar 파일 복사하고 Dockerfile 편집:

```
[root@dockeredu ~]# mkdir -p ~/buildlab/guestbook && cd $_
[root@dockeredu guestbook]# cp ~/guestbook/guestbook_MYSQL.jar .
[root@dockeredu guestbook]# vi Dockerfile
```

```
FROM openjdk:8
RUN apt-get update; \
    apt-get install -y net-tools; \
    apt-get clean;\
    rm -rf /var/lib/apt/lists/*
COPY guestbook_MYSQL.jar /app/
ADD https://github.com/scouter-project/scouter/releases/download/v2.6.1/scouter-min-2.6.1.tar.gz /
RUN tar xfz scouter-min-2.6.1.tar.gz; \
    echo "net_collector_ip=172.17.0.1" > /scouter/agent.java/conf/scouter.conf; \
    rm -fr scouter-min-2.6.1.tar.gz
LABEL maintainer="HwanYeoul Jeong<coordinatorj@jadecross.com>" \
      title="guestbook App" \
      version="2.0" \
      description="This image is guestbook service(MySQL)"
ENV APP_HOME /app
EXPOSE 8080
VOLUME /app/upload

WORKDIR $APP_HOME
ENTRYPOINT ["java"]
CMD ["-javaagent:/scouter/agent.java/scouter.agent.jar", "-jar", "guestbook_MYSQL.jar"]
```

⑤ docker build 명령을 실행하여 yu3papa/guestbook:1.0 이미지 작성

```
[root@dockeredu guestbook]# docker build -t yu3papa/guestbook:1.0 .
Sending build context to Docker daemon  47.06MB
Step 1/12 : FROM openjdk:8
~~~
Successfully tagged yu3papa/guestbook:2.0
```

LAB : 한글이 지원되는 MySQL DB를 사용하는 방명록 (4/5)

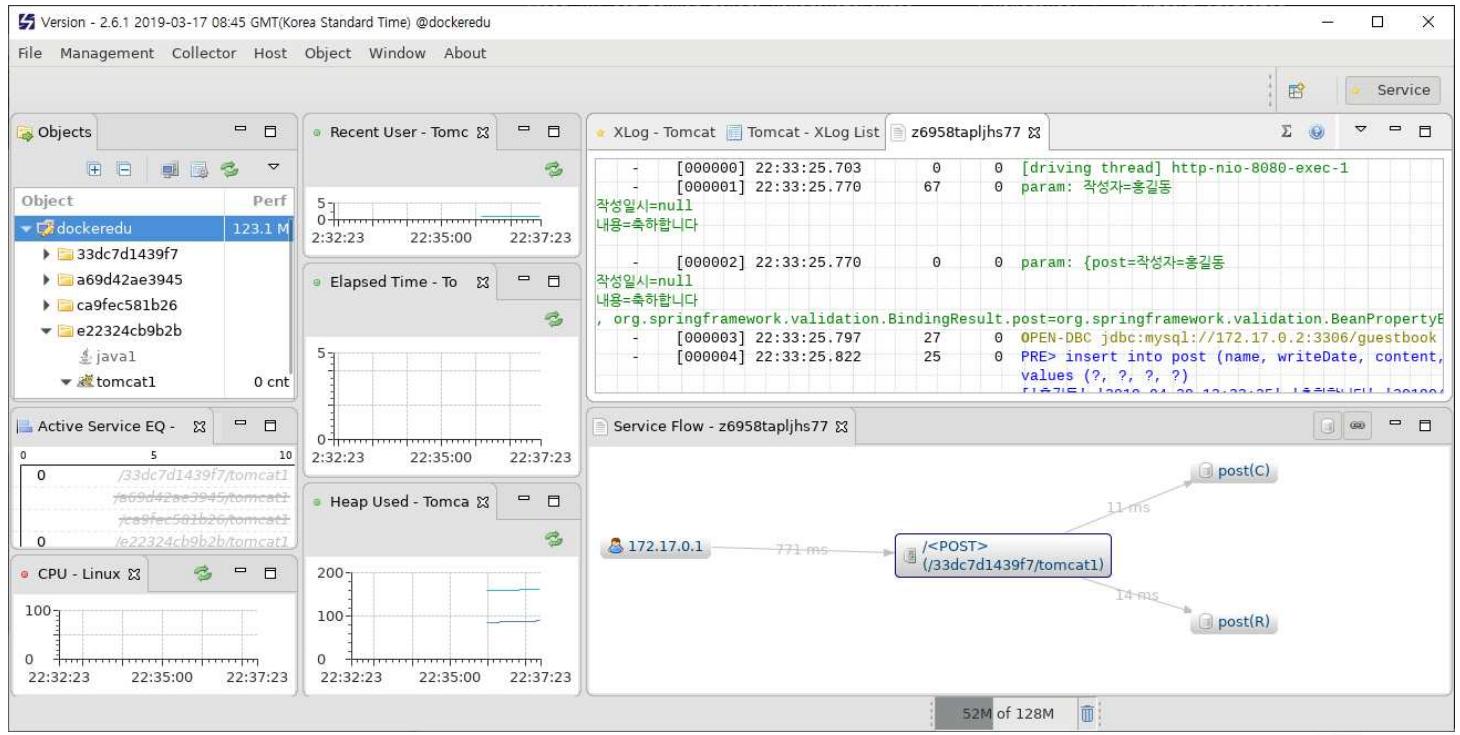
⑥ yu3papa/guestbook:1.0 이미지로부터 guestbookdb 컨테이너를 링크하는 컨테이너 시작

```
[root@dockeredu ~]# docker container run -d --name=guestbookapp -p 80:8080 --link guestbookdb:mysql yu3papa/guestbook:1.0
33dc7d1439f7041fb9c81df184e9a686d2f0ea82aa75ff2fea4ee85a422d5175
```

⑦ guestbookapp 컨테이너의 IP를 확인하고 브라우저로 접속(또는 http://localhost 로 접속)하여 방명록 작성하여 한글이 정상적으로 입력되는지 확인

```
[root@dockeredu ~]# docker container inspect --format="{{.NetworkSettings.Networks.bridge.IPAddress}}" guestbookapp
172.17.0.4
```

⑧ 스카우터를 이용하여 트랜잭션 프로파일 확인



LAB : 한글이 지원되는 MySQL DB를 사용하는 방명록 (5/5)

- **Docker Hub에 이미지 업로드**

⑨ Docker Hub에 로그인

```
[root@dockeredu ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to
https://hub.docker.com to create one.
Username: yu3papa
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

⑩ yu3papa/mysql_hangul:1.0 이미지를 Docker Hub에 업로드(push)

```
[root@dockeredu ~]# docker image push yu3papa/mysql_hangul:1.0
The push refers to repository [docker.io/yu3papa/mysql_hangul]
a377c5092c1f: Pushed
~~~
65f3b0435c42: Mounted from library/mysql
1.0: digest: sha256:8c8189041a255d9bab0f36f365c47ff65d0787744b96aee8834e40fec31b68da size: 3849
```

⑪ yu3papa/guestbook:1.0 이미지를 Docker Hub에 업로드(push)

```
[root@dockeredu ~]# docker image push yu3papa/guestbook:1.0
The push refers to repository [docker.io/yu3papa/guestbook]
15edddf5d8a6: Pushed
~~~
fbb641a8b943: Mounted from library/openjdk
1.0: digest: sha256:372fdade8cb3d44c015ce9f5041c20f2e1d06af00ed5314101ca629df21c567f size: 2849
```

연습문제 (1/2)

① 이미지 작업할 폴더를 생성하고 이동한 후 fortune rpm 파일과 cowsay-loop.sh 파일을 복사하고 Dockerfile 편집:

```
[root@dockeredu ~]# cd ~/buildlab/cowsay
[root@dockeredu cowsay]# vi cowsay-loop.sh
```

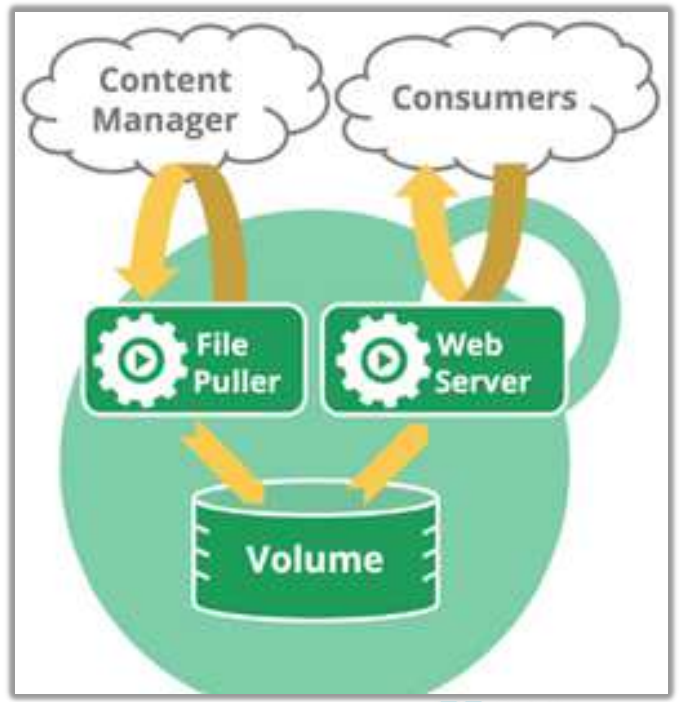
```
#!/bin/bash

while true
do
  echo '<pre>' > /var/htdocs/index.html
  date >> /var/htdocs/index.html
  fortune | cowsay >> /var/htdocs/index.html
  echo '</pre>' >> /var/htdocs/index.html
  cat /var/htdocs/index.html
  sleep $INTERVAL
done
```

```
[root@dockeredu cowsay]# vi Dockerfile
```

```
FROM centos:7.6.1810
COPY fortune-mod-1.99.1-17.sdl7.x86_64.rpm /
COPY cowsay-loop.sh /
WORKDIR /
VOLUME /var/htdocs
RUN yum install -y epel-release; \
    yum install -y cowsay; \
    yum install -y fortune-mod-1.99.1-17.sdl7.x86_64.rpm; \
    chmod u+x cowsay-loop.sh
ENTRYPOINT ["/cowsay-loop.sh"]
```

② docker build 명령을 실행하여 yu3papa/cowsay:1.0 이미지 작성

```
[root@dockeredu cowsay]# docker build -t yu3papa/cowsay:1.0 .
Sending build context to Docker daemon  1.121MB
Step 1/7 : FROM centos:7.6.1810
 ---> f1cb7c7d58b7
~~~
Successfully built 2718dc90defd
Successfully tagged yu3papa/cowsay:1.0
```



연습문제 (2/2)

③ 이름있는 도커 볼륨을 생성

```
# docker volume create cowsayVol
cowsayVol
```

④ yu3papa/cowsay:1.0 이미지로 컨테이너를 생성 log 명령으로 결과를 확인

```
# docker container run -d --name=cowsaycon -e INTERVAL=2 -v cowsayVol:/var/htdocs --rm yu3papa/cowsay:1.0
aedb990cc15e6be860ec4f6b050e983a0b558ea75aefb105ca875b8525def11e
```

⑤ 컨테이너의 log 를 확인해보고 cowsayVol에 생성되는 index.html의 내용도 확인

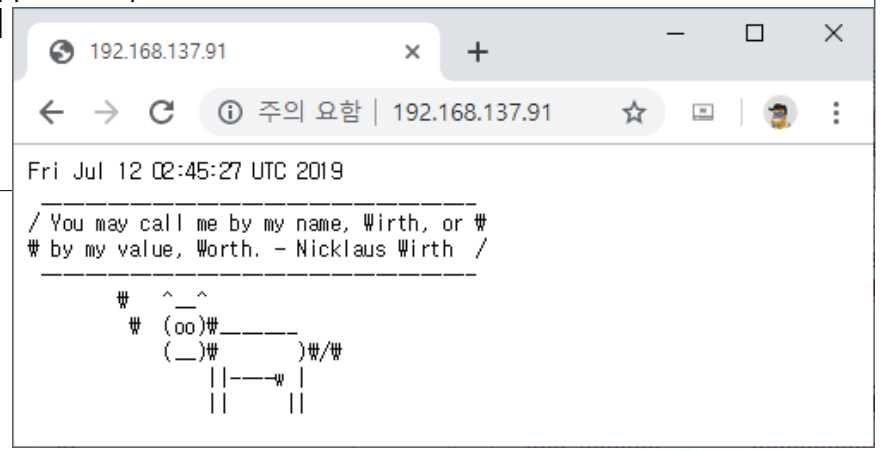
```
# docker container logs -f cowsaycon
<pre>
Fri Jul 12 02:33:11 UTC 2019
 ________________________________________
/ Euch ist bekannt, was wir beduerfen;   \
| Wir wollen stark Getraenke schluerfen. |
|                                        |
\ -- Goethe, "Faust"                     /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
</pre>
```

```
# watch cat /var/lib/docker/volumes/cowsayVol/_data/index.html
<pre>
Fri Jul 12 03:04:17 UTC 2019
 ____________________________
< Santa Claus is watching! >
 ----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||
</pre>
```

⑥ cowsayVol 볼륨을 읽기전용으로 마운트하고 80번 포트에서 웹서비스를 수행하는 nginx 컨테이너 생성

```
# docker container run -d --name=nginxcon -v cowsayVol:/usr/share/nginx/html:ro
-p 80:80 nginx:alpine
600fe7baf75cdc49c3f02f95f58c874c9b20cc12c41da40e469a7d4cc1a67124
```

⑦ dockeredu VM 외부에서 브라우저를 통해 192.168.56.91 서버에서 웹서비스가 되는지 확인



vscode 프로그램(GUI Program)을 이용한 개발환경 설정 (1/2)

Dockerfile

① 이미지 작업할 폴더로 이동하고 Dockerfile 확인

```
[root@dockeredu ~]# cd ~/buildlab/vscode
[root@dockeredu vscode]# mkdir work
[root@dockeredu vscode]# cat Dockerfile
```

```
FROM centos:centos7.7.1908
# vscode
ADD code-1.52.1-1608137084.el7.x86_64.rpm /
# Language, Locale
ENV TZ=Asia/Seoul
RUN localedef -f UTF-8 -i ko_KR ko_KR.UTF-8; \
    ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
# x-window + sshd + vscode
RUN yum install -y xorg-x11-xauth; \
    yum install -y dbus-x11; \
    yum install -y openssh-server openssh-clients openssh-askpass; \
    yum localinstall -y /code-1.52.1-1608137084.el7.x86_64.rpm
RUN echo "jadecross" | passwd root --stdin; \
    sed -i --follow-symlinks 's/#X11UseLocalhost yes/X11UseLocalhost no/g' /etc/ssh/sshd_config; \
    sed -i --follow-symlinks 's/#HandleLidSwitch=suspend/HandleLidSwitch=ignore/g' /etc/systemd/logind.conf; \
    mkdir ~/.code-workspace; \
    echo "alias code='code --user-data-dir ~/.code-workspace > /dev/null 2>&1 &'" >> ~/.bashrc; \
    echo "export LANG=ko_KR.UTF-8" >> ~/.bash_profile

EXPOSE 22
VOLUME /work

CMD ["/usr/sbin/init"]
```

② docker build 명령을 실행하여 yu3papa/centos7vscode:1.0 이미지 작성후 컨테이너 실행

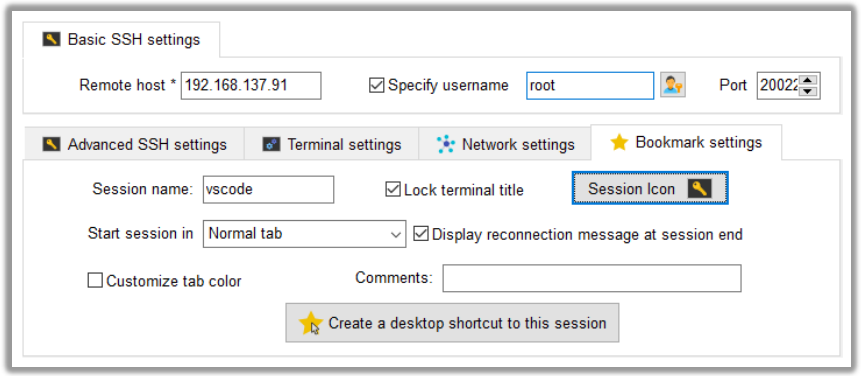
```
[root@dockeredu vscode]# docker image build --no-cache -t yu3papa/centos7vscode:1.0 .
~~~
[root@dockeredu vscode]# docker container run --privileged -d --name=vscode \
                         -v /root/buildlab/vscode/work:/work \
                         -p 20022:22 yu3papa/centos7vscode:1.0
```

--privileged

By default, Docker containers are "unprivileged" and cannot, for example, run a Docker daemon inside a Docker container. This is because by default a container is not allowed to access any devices, but a "privileged" container is given access to all devices (see the documentation on cgroups devices)

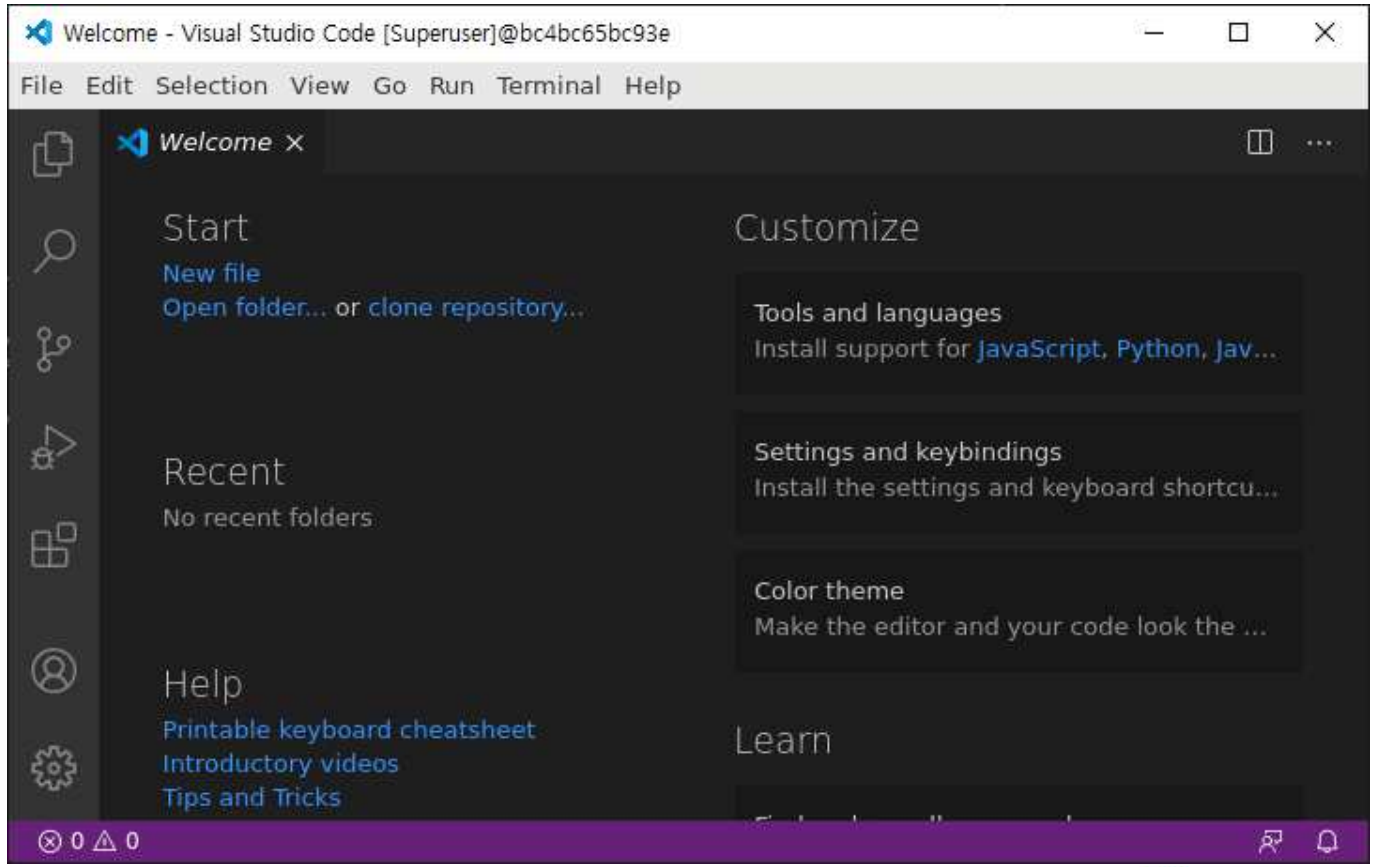
vscode 프로그램(GUI Program)을 이용한 개발환경 설정 (2/2)

③ MobaXterm에서 vscode 세션 정보를 확인하고 SSH 연결



④ 쉘이서 code 명령을 실행하여 vscode GUI 프로그램이 활성화 확인

```
[root@88e977f94bdc ~]# code
[1] 266
```



Docker Compose

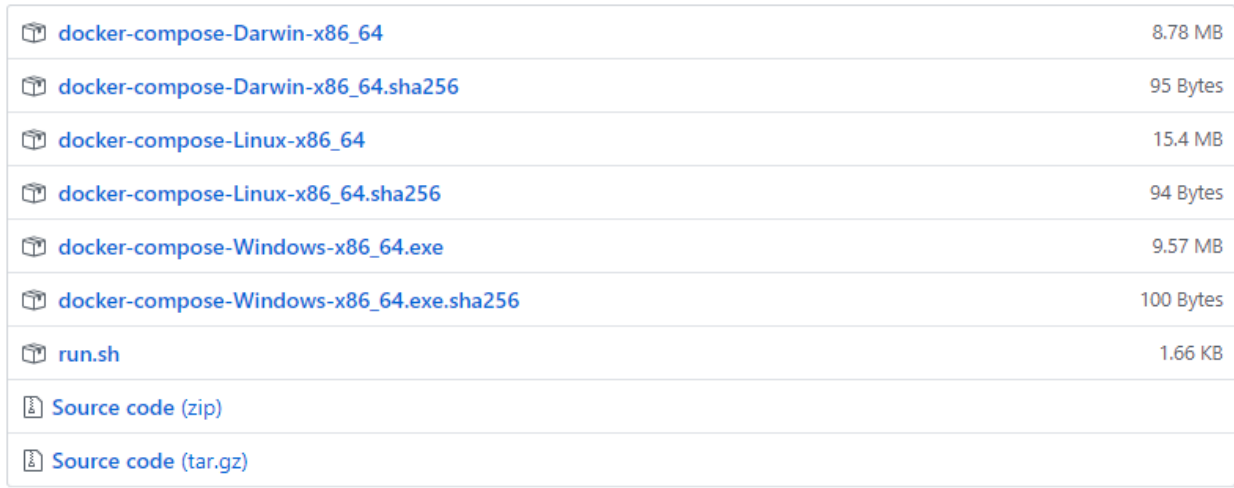
Docker Compose

여러 개의 컨테이너를 연동하는 경량의 오케스트레이션 툴

- Docker Compose is used fo running multiple containers as a single service
- Here, containers run in isolation but cat interact with each other
- All Docker Compose files are YAML files
- In Docker Compose, a user can start all ther services(containers) using a single command

docker-compose 설치

- **binary 다운로드**
  - curl -L https://github.com/docker/compose/releases/download/1.24.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose



docker-compose-Darwin-x86_64	8.78 MB
docker-compose-Darwin-x86_64.sha256	95 Bytes
docker-compose-Linux-x86_64	15.4 MB
docker-compose-Linux-x86_64.sha256	94 Bytes
docker-compose-Windows-x86_64.exe	9.57 MB
docker-compose-Windows-x86_64.exe.sha256	100 Bytes
run.sh	1.66 KB
Source code (zip)	
Source code (tar.gz)	

- **실행 퍼미션 추가**
  - chmod +x /usr/local/bin/docker-compose

- **설치 확인**
  - docker-compose --version
    - docker-compose version 1.24.0, build 0aa59064

https://docs.docker.com/compose/install/

docker-compose.yml

- **Docker Compose의 구성 파일**
  - 한 파일 안에 여러 컨테이너 설정 내용 저장
  - 도커 애플리케이션을 위한 서비스, 네트워크, 볼륨을 정의
  - 텍스트 파일 형태인 YAML 형식을 사용
  - Version 3.7(최신) > Version 2 > Version 1
    - 각 버전에 맞게 지원하는 도커 엔진과 명령들이 상이함

- **YAML**
  - YAML Ain't Markup Language
  - https://ko.wikipedia.org/wiki/YAML
  - 사람이 쉽게 읽을 수 있는 데이터 표현 양식
  - 탭 대신 공백 사용
  - 배열 데이터의 앞에는 '-' 기호 사용

- **Compose file format compatibility matrix**

Compose file format	Docker Engine
1	1.9.0+
2.0	1.10.0+
2.1	1.12.0+
2.2, 3.0, 3.1, 3.2	1.13.0+
2.3, 3.3, 3.4, 3.5	17.06.0+
2.4	17.12.0+
3.6	18.02.0+
3.7	18.06.0+

방명록 서비스를 docker-compose로 운영하기 (1/8)

- **docker-compose.yml 구성**
  - Compose file version 3 참고 : https://docs.docker.com/compose/compose-file/

① 작업할 폴더를 생성하고 이동한 후 docker-compose.yml 파일 내용 확인

```
[root@dockeredu ~]# cd ~/visitorbook
[root@dockeredu visitorbook]# cat docker-compose.yml
```

compose file format 버전 지정 →
서비스할 컨테이너 섹션 →
서비스명 →
컨테이너의 이미지 지정 →
컨테이너의 환경변수 지정 →
컨테이너에서 사용할 볼륨정의 →
컨테이너의 포트 매핑 →
다른 컨테이너의 링크 정보 →
컨테이너의 디펜던시 →
db 서비스(컨테이너) 설정 정보 →
볼륨 정보 →

```
version: "3.7"
services:
  app:
    image: yu3papa/guestbook:1.0
    environment:
      - MYSQL_PORT_3306_TCP_ADDR=db
      - MYSQL_PORT_3306_TCP_PORT=3306
      - MYSQL_ENV_MYSQL_DATABASE=guestbook
      - MYSQL_ENV_MYSQL_ROOT_PASSWORD=edu
    volumes:
      - upload-data:/app/upload
    ports:
      - "80:8080"
    links:
      - db
    depends_on:
      - db
  db:
    image: yu3papa/mysql_hangul:1.0
    environment:
      - MYSQL_DATABASE=guestbook
      - MYSQL_ROOT_PASSWORD=edu
    volumes:
      - mysql-data:/var/lib/mysql
volumes:
  mysql-data:
  upload-data:
```

방명록 서비스를 docker-compose로 운영하기 (2/8)

▪ docker-compose up

Builds, (re)creates, starts, and attaches to containers for a service.

Unless they are already running, this command also starts any linked services.

The `docker-compose up` command aggregates the output of each container. When the command exits, all containers are stopped. Running `docker-compose up -d` starts the containers in the background and leaves them running.

If there are existing containers for a service, and the service's configuration or image was changed after the container's creation, `docker-compose up` picks up the changes by stopping and recreating the containers (preserving mounted volumes). To prevent Compose from picking up changes, use the `--no-recreate` flag.

If you want to force Compose to stop and recreate all containers, use the `--force-recreate` flag.

✓ **docker-compose up -d 명령을 사용하여 app, db 컨테이너 실행**

```
[root@dockeredu visitorbook]# docker-compose up -d
1 Creating network "visitorbook_default" with the default driver
Creating volume "visitorbook_mysql-data" with default driver
2 Creating volume "visitorbook_upload-data" with default driver
Creating visitorbook_db_1 ... done
3 Creating visitorbook_app_1 ... done
```

<docker-compose up 명령시 작업 내역>

① <폴더명>_default 사용자 정의 네트워크 생성

```
[root@dockeredu ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
53e4ef83a712        bridge              bridge              local
0ca6bf22a5f0        host                host                local
b9f1994e5609        none                null                local
3268b8527890        visitorbook_default bridge              local
```

② <폴더명>_<volumes 섹션 하위 볼륨명> 볼륨 생성

```
[root@dockeredu ~]# docker volume ls
DRIVER              VOLUME NAME
local               visitorbook_mysql-data
local               visitorbook_upload-data
```

③ <폴더명>_<services 섹션 하위 서비스명>_<#> 컨테이너 생성

```
[root@dockeredu ~]# docker container ls -a
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS          PORTS                  NAMES
86a676ffd9ed   yu3papa/guestbook:1.0       "java -javaagent:/sc…"   53 seconds ago   Up 51 seconds   0.0.0.0:80->8080/tcp   visitorbook_app_1
466f750f8587   yu3papa/mysql_hangul:1.0    "/entrypoint.sh mysq…"   53 seconds ago   Up 52 seconds   3306/tcp               visitorbook_db_1
```

방명록 서비스를 docker-compose로 운영하기 (3/8)

- **docker-compose ps**
  - List containers.

① **docker-compose ps 명령을 이용하여 컨테이너 목록 표시**

```
[root@dockeredu visitorbook]# docker-compose ps
      Name                     Command               State          Ports
-------------------------------------------------------------------------------
visitorbook_app_1   java -javaagent:/scouter/a ...   Up      0.0.0.0:80->8080/tcp
visitorbook_db_1    /entrypoint.sh mysqld            Up      3306/tcp
```

방명록 서비스를 docker-compose로 운영하기 (4/8)

- **docker-compose exec**
  - Execute a command in a running container.

② **docker-compose exec 명령을 이용하여 db 서비스에서 쉘을 실행시키고 IP와 hosts 파일 목록 확인**

```
[root@dockeredu visitorbook]# docker-compose exec db /bin/bash
root@eac8571c2934:/#
root@eac8571c2934:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
53: eth0@if54: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:16:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.22.0.4/16 brd 172.22.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@eac8571c2934:/#
root@eac8571c2934:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.22.0.4      eac8571c2934
root@eac8571c2934:/#
```

방명록 서비스를 docker-compose로 운영하기 (5/8)

- **docker-compose logs**
  - View output from containers.

③ **docker-compose logs 명령을 이용하여 서비스의 output을 확인**

```
[root@dockeredu visitorbook]# docker-compose logs
app_1  |   .   ____          _            __ _ _
app_1  |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
app_1  | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
app_1  |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
app_1  |   '  |____| .__|_| |_|_| |_\__, | / / / /
app_1  |  =========|_|==============|___/=/_/_/_/
app_1  |  :: Spring Boot ::        (v2.1.3.RELEASE)
app_1  |
app_1  | 2019-05-01 03:07:19.021  INFO 1 --- [           main] c.j.guestbook.GuestbookMysqlApplication  : Starting GuestbookMysqlApplication
v0.0.1-SNAPSHOT on 86a676ffd9ed with PID 1 (/app/guestbook_MYSQL.jar started by root in /app)
~~~
db_1   | Initializing database
db_1   | 2019-05-01T03:07:10.801664Z 0 [Warning] InnoDB: New log files created, LSN=45790
db_1   | 2019-05-01T03:07:10.895051Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
db_1   | 2019-05-01T03:07:10.969099Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time that this server
has been started. Generating a new UUID: 3640e215-6bbe-11e9-93c0-0242ac160002.
db_1   | 2019-05-01T03:07:10.972845Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
db_1   | 2019-05-01T03:07:10.972893Z 0 [Warning] Failed to setup SSL
db_1   | 2019-05-01T03:07:10.972899Z 0 [Warning] SSL error: SSL context is not usable without certificate and private key
db_1   | 2019-05-01T03:07:10.973350Z 1 [Warning] root@localhost is created with an empty password ! Please consider switching off the --
initialize-insecure option.
db_1   | Database initialized
~~~
```

방명록 서비스를 docker-compose로 운영하기 (6/8)

- **docker-compose start/stop/restart**
  - 여러 컨테이너의 시작/정지/재시작

④ **docker-compose stop 명령을 이용하여 서비스를 중지**

```
[root@dockeredu visitorbook]# docker-compose stop
Stopping visitorbook_app_1 ... done
Stopping visitorbook_db_1  ... done
```

⑤ **docker-compose start 명령을 이용하여 서비스를 시작**

```
[root@dockeredu visitorbook]# docker-compose start
Starting db  ... done
Starting app ... done
```

⑥ **docker-compose restart 명령을 이용하여 서비스를 재시작**

```
[root@dockeredu visitorbook]# docker-compose restart
Restarting visitorbook_app_1 ... done
Restarting visitorbook_db_1  ... done
```

방명록 서비스를 docker-compose로 운영하기 (7/8)

- **docker-compose config**
  - Validate and view the Compose file.

⑦ **docker-compose config 파일을 이용하여 docker-compose.yml 파일의 Syntax 오류를 검증하고 구성 내용을 확인**

```
[root@dockeredu visitorbook]# docker-compose config
services:
  app:
    depends_on:
    - db
    environment:
      MYSQL_ENV_MYSQL_DATABASE: guestbook
      MYSQL_ENV_MYSQL_ROOT_PASSWORD: edu
      MYSQL_PORT_3306_TCP_ADDR: db
      MYSQL_PORT_3306_TCP_PORT: '3306'
    image: yu3papa/guestbook:1.0
    links:
    - db
    ports:
    - published: 80
      target: 8080
    volumes:
    - upload-data:/app/upload:rw
  db:
    environment:
      MYSQL_DATABASE: guestbook
      MYSQL_ROOT_PASSWORD: edu
    image: yu3papa/mysql_hangul:1.0
    volumes:
    - mysql-data:/var/lib/mysql:rw
version: '3.7'
volumes:
  mysql-data: {}
  upload-data: {}
```

방명록 서비스를 docker-compose로 운영하기 (8/8)

▪ docker-compose down

Stops containers and removes containers, networks, volumes, and images created by `up`.

By default, the only things removed are:

- Containers for services defined in the Compose file
- Networks defined in the `networks` section of the Compose file
- The default network, if one is used

Networks and volumes defined as `external` are never removed.

⑧ docker-compose down 명령을 사용하여 서비스를 중지하고 관련 리소스 삭제

```
[root@dockeredu visitorbook]# docker-compose down
Stopping visitorbook_app_1 ... done
Stopping visitorbook_db_1  ... done
Removing visitorbook_db_run_65bff93ce2ca  ... done
Removing visitorbook_app_run_27c656a3f053 ... done
Removing visitorbook_db_run_61388caed67d  ... done
Removing visitorbook_app_run_5ecad66089e9 ... done
Removing visitorbook_app_run_1ea6967b4fe1 ... done
Removing visitorbook_app_1                ... done
Removing visitorbook_db_1                 ... done
Removing network visitorbook_default
```

1 2 3

<docker-compose down 명령 시 작업 내역>

① 컨테이너 중지

② 컨테이너 삭제

```
[root@dockeredu visitorbook]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

③ <폴더명>_default 사용자 정의 네트워크 삭제

```
[root@dockeredu visitorbook]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
53e4ef83a712        bridge              bridge              local
0ca6bf22a5f0        host                host                local
b9f1994e5609        none                null                local
```

compose file format 버전 3 주의 사항

- **links 항목의 기능은 version 1까지만 지원**
  - version 3에서는 환경 변수 population은 지원하지 않음
  - version 3에서는 서비스명으로 컨테이너 IP 접속 가능
- **scale, replicas 는 지원 안함**
  - 서비스 수평확장인 scale/replicas는 더 이상 compose에서 지원하지 않고 swarm에서 지원
- **depends_on 항목의 condition은 지원 안함**

docker-compose를 이용한 CI 환경 구성 (1/5)

- **지속적 통합 환경 구축**
  - https://github.com/hussainweb/gitlab-jenkins-sonarqube

GitLab

GIT 형상관리
계획 및 태스크 관리
웹 인터페이스 지원

Jenkins

소스코드 변경 감지
스케쥴 빌드 및 정적분석
파이프라인을 통한 다양한 작업

SonarQube

소스코드 품질 확인

docker-compose를 활용하여 쉽고 빠르게 구축

docker-compose를 이용한 CI 환경 구성 (2/5)

① 작업할 폴더(~/gitlab-jenkins-sonarqube)로 이동후 docker-compose.yml 파일 확인

```
[root@dockeredu ~]# cd ~/gitlab-jenkins-sonarqube/
[root@dockeredu gitlab-jenkins-sonarqube]# cat docker-compose.yml
```

```yaml
version: "3"
services:
  web:
    image: 'gitlab/gitlab-ce:11.10.3-ce.0'
    restart: always
    hostname: 'gitlab'
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://localhost:8000'
    ports:
      - '8000:8000'
      - '8443:8443'
      - '23:22'
    volumes:
      - 'gitlab-config:/etc/gitlab'
      - 'gitlab-logs:/var/log/gitlab'
      - 'gitlab-data:/var/opt/gitlab'
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile.jenkins
    restart: always
    hostname: 'jenkins'
    ports:
      - '8080:8080'
      - '50000:50000'
    volumes:
      - 'jenkins_home:/var/jenkins_home'
      - '/var/run/docker.sock:/var/run/docker.sock'
  sonarqube:
    image: 'sonarqube:7.6-community'
    ports:
      - '9000:9000'
    volumes:
      - 'sonarqube_conf:/opt/sonarqube/conf'
      - 'sonarqube_data:/opt/sonarqube/data'
      - 'sonarqube_logs:/opt/sonarqube/logs'
      - 'sonarqube_extensions:/opt/sonarqube/extensions'

volumes:
  gitlab-data:
  gitlab-logs:
  gitlab-config:
  jenkins_home:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_logs:
  sonarqube_extensions:
```

Docker-out-of-Docker

② docker-compose 명령으로 컨테이너 일괄 실행

```
[root@dockeredu gitlab-jenkins-sonarqube]# docker-compose up -d
Creating network "gitlab-jenkins-sonarqube_default" with the default driver
Creating gitlab-jenkins-sonarqube_jenkins_1   ... done
Creating gitlab-jenkins-sonarqube_sonarqube_1 ... done
Creating gitlab-jenkins-sonarqube_web_1       ... done
```

Dockerfile.jenkins

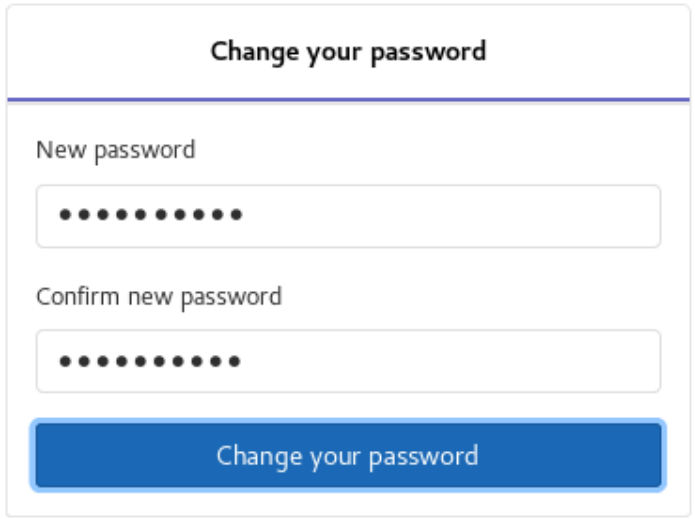
```
FROM jenkins/jenkins:lts-jdk11
MAINTAINER miiro@getintodevops.com
USER root

# Install the latest Docker CE binaries
RUN apt-get update && \
    apt-get -y install apt-transport-https \
      ca-certificates \
      curl \
      gnupg2 \
      software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo
"$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
      "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release;
echo "$ID") \
      $(lsb_release -cs) \
      stable" && \
   apt-get update && \
   apt-get -y install docker-ce

## SonarQube runner needs NodeJS
RUN curl -sL https://deb.nodesource.com/setup_10.x | bash - && \
    apt-get install -y nodejs
```

Container안에서
Docker 명령 수행 가능

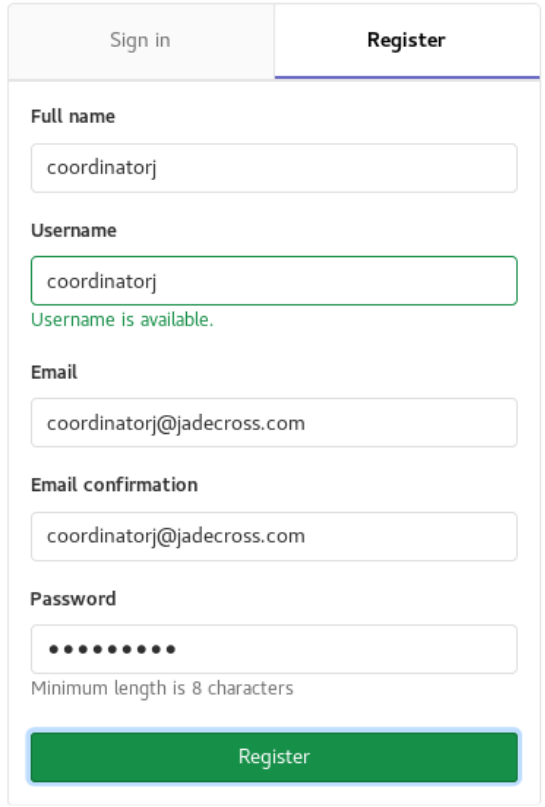
docker-compose를 이용한 CI 환경 구성 (3/5)

- **GitLab 구성**
  - http://192.168.56.91:8000 접속

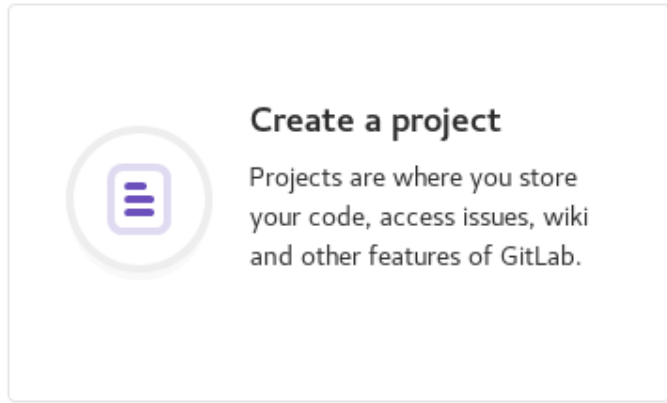
① 관리자 암호 변경



② 사용자 계정 생성

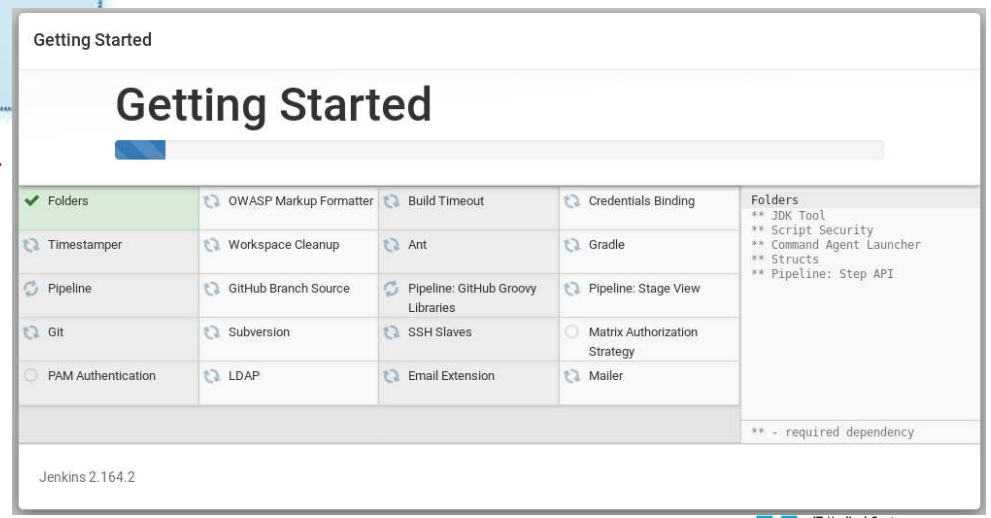


③ 프로젝트 생성



docker-compose를 이용한 CI 환경 구성 (4/5)

- **Jenkins 구성**
  - http://192.168.56.91:8080

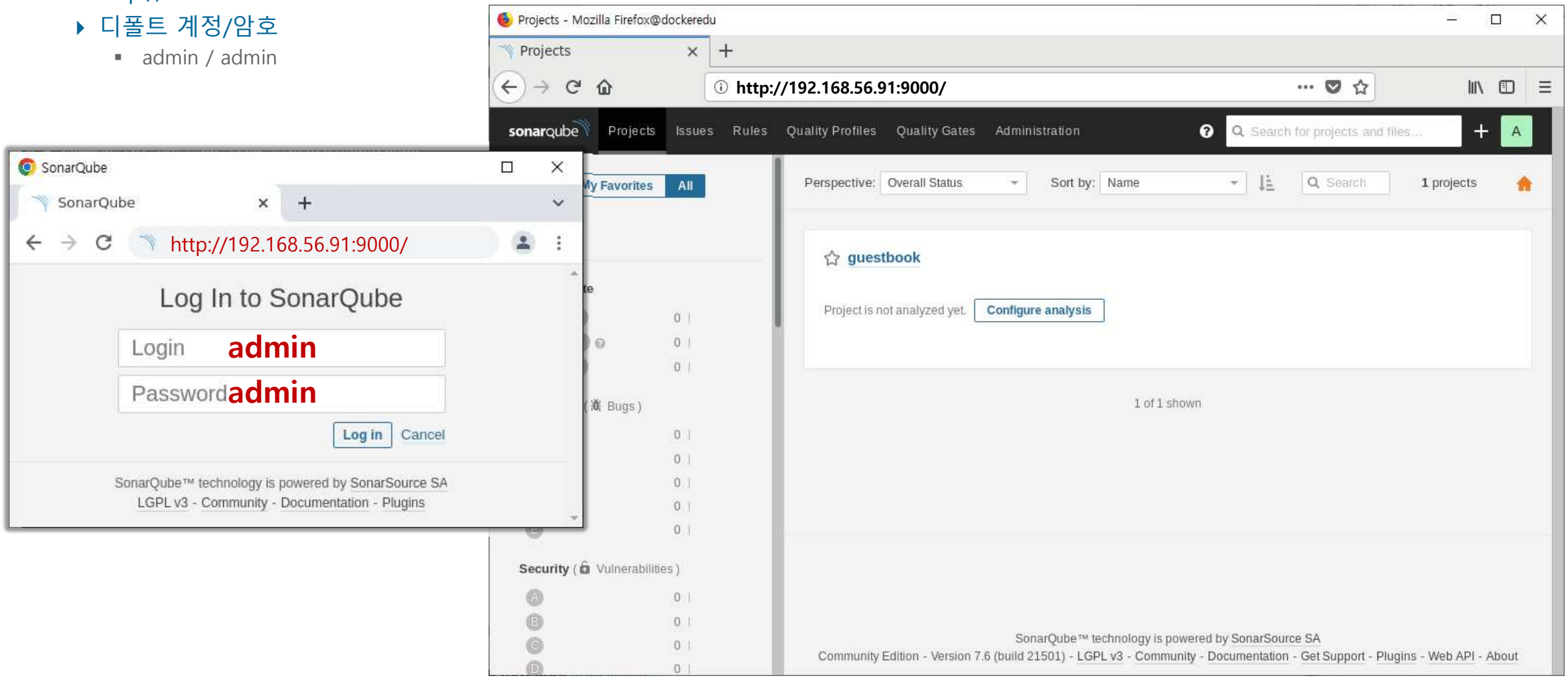


✓ Jenkins 컨테이너 로그에서 admin 초기 암호 확인

```
[root@dockeredu gitlab-jenkins-sonarqube]# docker container logs gitlab-jenkins-sonarqube_jenkins_1
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
~~~
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

e81faa57efaa452f8fb62728eac5a4ec

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*************************************************************
```

docker-compose를 이용한 CI 환경 구성 (5/5)

- **SonarQube 구성**
  - http://localhost:9000
  - 디폴트 계정/암호
    - admin / admin



docker-compose를 이용한 minecraft (1/2)

① minecraft에 대한 docker-compose.yml 파일이 있는 폴더로 이동

```
[root@dockeredu ~]# cd ~/minecraft
[root@dockeredu minecraft]#
```

② minecraft 서버 이미지를 생성하는 Dockerfile 내용 확인

```
[root@dockeredu minecraft]# cat Dockerfile

FROM openjdk:8

WORKDIR /minecraft

ADD minecraft_server.1.12.2.jar .
ADD eula.txt .
ADD server.properties .

VOLUME /minecraft/world

EXPOSE 25565

ENTRYPOINT ["java", "-jar", "minecraft_server.1.12.2.jar", "nogui"]
```

③ minecraft 서비스를 실행하는 docker-compose.yml 파일 내용 확인

```
[root@dockeredu minecraft]# cat docker-compose.yml

version: "3.7"
services:
  minecraft:
    image: yu3papa/minecraft:1.0
    ports:
    - "25565:25565"
    volumes:
    - world-data:/minecraft/world

volumes:
  world-data:
```

④ docker-compose up -d 명령으로 minecraft 서비스 실행

```
[root@dockeredu minecraft]# docker-compose up -d
Creating network "minecraft_default" with the default driver
Creating volume "minecraft_world-data" with default driver
Creating minecraft_minecraft_1 ... done
```

docker-compose를 이용한 minecraft (2/2)

⑤ C:\jadeedu-docker\sw\minecraft 하위 폴더의 Minecraft_ver_1.7.2.exe 실행

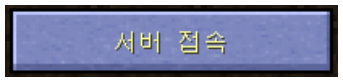
└▶ Minecraft_ver_1.7.2.exe 실행

└▶ Play

└▶ 멀티플레이

└▶ 서버 추가

└▶ 서버 접속 후 플레이

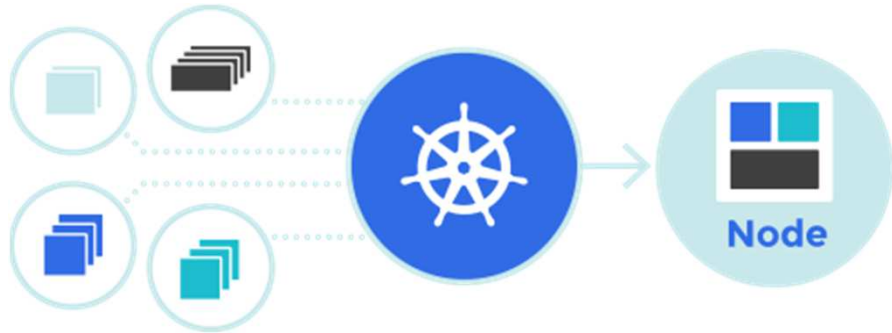


Kubernetes 데모

https://kubernetes.io/ko/

✓ **쿠버네티스(K8s)는 컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈소스 시스템입니다.**

애플리케이션을 구성하는 컨테이너들의 쉬운 관리 및 발견을 위해서 컨테이너들을 논리적인 단위로 그룹화합니다. 쿠버네티스는 Google에서 15년간 프로덕션 워크로드 운영한 경험을 토대로 구축되었으며, 커뮤니티에서 제공한 최상의 아이디어와 방법들이 결합되어 있습니다.



✓ **행성 규모 확장성**

Google이 일주일에 수십억 개의 컨테이너들을 운영하게 해준 원칙들에 따라 디자인되었기 때문에, 쿠버네티스는 운영팀의 규모를 늘리지 않고도 확장될 수 있습니다.

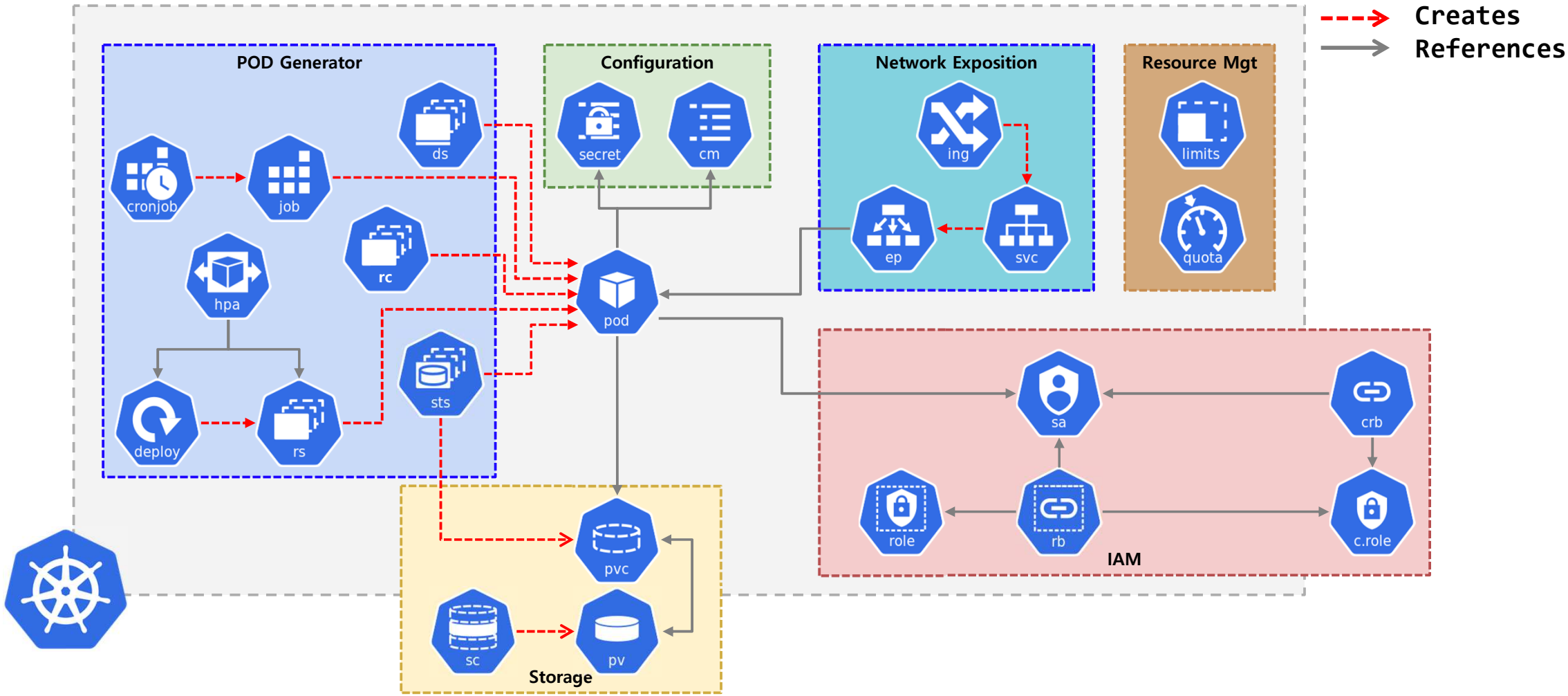
✓ **무한한 유연성**

지역적인 테스트든지 글로벌 기업 운영이든지 상관없이, 쿠버네티스의 유연성은 사용자의 복잡한 니즈를 모두 수용하기 때문에 사용자의 애플리케이션들을 끊임없고 쉽게 전달할 수 있습니다.

✓ **어디서나 동작**

쿠버네티스는 오픈소스로서 온-프레미스, 하이브리드, 또는 퍼블릭 클라우드 인프라스트럭처를 활용하는데 자유를 제공하며, 워크로드를 사용자에게 관건이 되는 곳으로 손쉽게 이동시켜 줄 수 있습니다.

Kubernetes Resources Map



https://kubernetes.io/ko/docs/reference/kubectl/cheatsheet/

guestbook 앱을 쿠버네티스 클러스터에 배포 및 운영하기 (1/2)

① K8S 클러스터에 guestbook 앱 배포

```
$ kubectl run guestbook --image=yu3papa/k8s_guestbook:1.0 --port=8080 --generator=run/v1
replicationcontroller/guestbook created
```

② 생성된 POD 목록 보기

```
$ kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
guestbook-t4kfh   1/1       Running   0          1m
```

③ 서비스 객체 생성

```
$ kubectl expose rc guestbook --type=LoadBalancer --name guestbook-http
service/guestbook-http exposed
```

④ 서비스 목록 확인

```
$ kubectl get services
NAME             TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
guestbook-http   LoadBalancer   10.7.249.233   34.80.200.79     8080:31060/TCP   1m
```

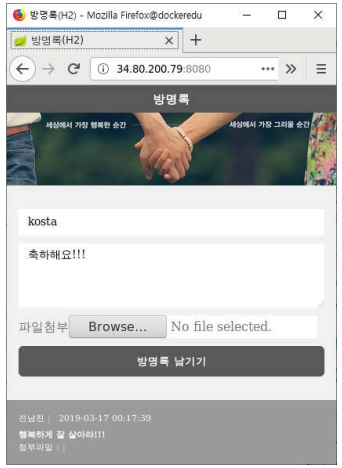
⑤ Replication Controller 개수 확인

```
$ kubectl get rc
NAME        DESIRED   CURRENT   READY     AGE
guestbook   1         1         1         12m
```

guestbook 앱을 쿠버네티스 클러스터에 배포 및 운영하기 (2/2)

⑥ 확인된 EXTERNAL-IP 와 PORT로 웹 브라우저로 접속

```
[root@dockeredu ~]# curl http://34.80.200.79:8080
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link href="common.css" rel="stylesheet">
<link href="guestbook.css" rel="stylesheet">

<title>방명록(H2)</title>
</head>
~~~
</body>
```



⑦ 레플리카 수 증가 시키기 (스케일 아웃)

```
$ kubectl scale rc guestbook --replicas=2
replicationcontroller/guestbook scaled
$
```

⑧ 스케일 아웃 결과 보기

```
$ kubectl get rc
NAME        DESIRED   CURRENT   READY   AGE
guestbook   2         2         1       22m

$ kubectl get po
NAME              READY   STATUS    RESTARTS   AGE
guestbook-hpgtg   1/1     Running   0          48s
guestbook-njfbd   1/1     Running   0          48s
```

⑨ 워커 노드에서 스케일아웃된 컨테이너 목록 확인

```
$ watch docker container ls -f ancestor=yu3papa/k8s_guestbook:1.0

CONTAINER ID  IMAGE                    COMMAND                  CREATED         STATUS        PORTS  NAMES
60da30a757f8  yu3papa/guestbook_h2@... "java -jar guestbo..." 7 minutes ago  Up 7 minutes         k8s_guestbook_...
f5c76a861278  yu3papa/guestbook_h2@... "java -jar guestbo..." 12 minutes ago Up 12 minutes        k8s_guestbook_...
```

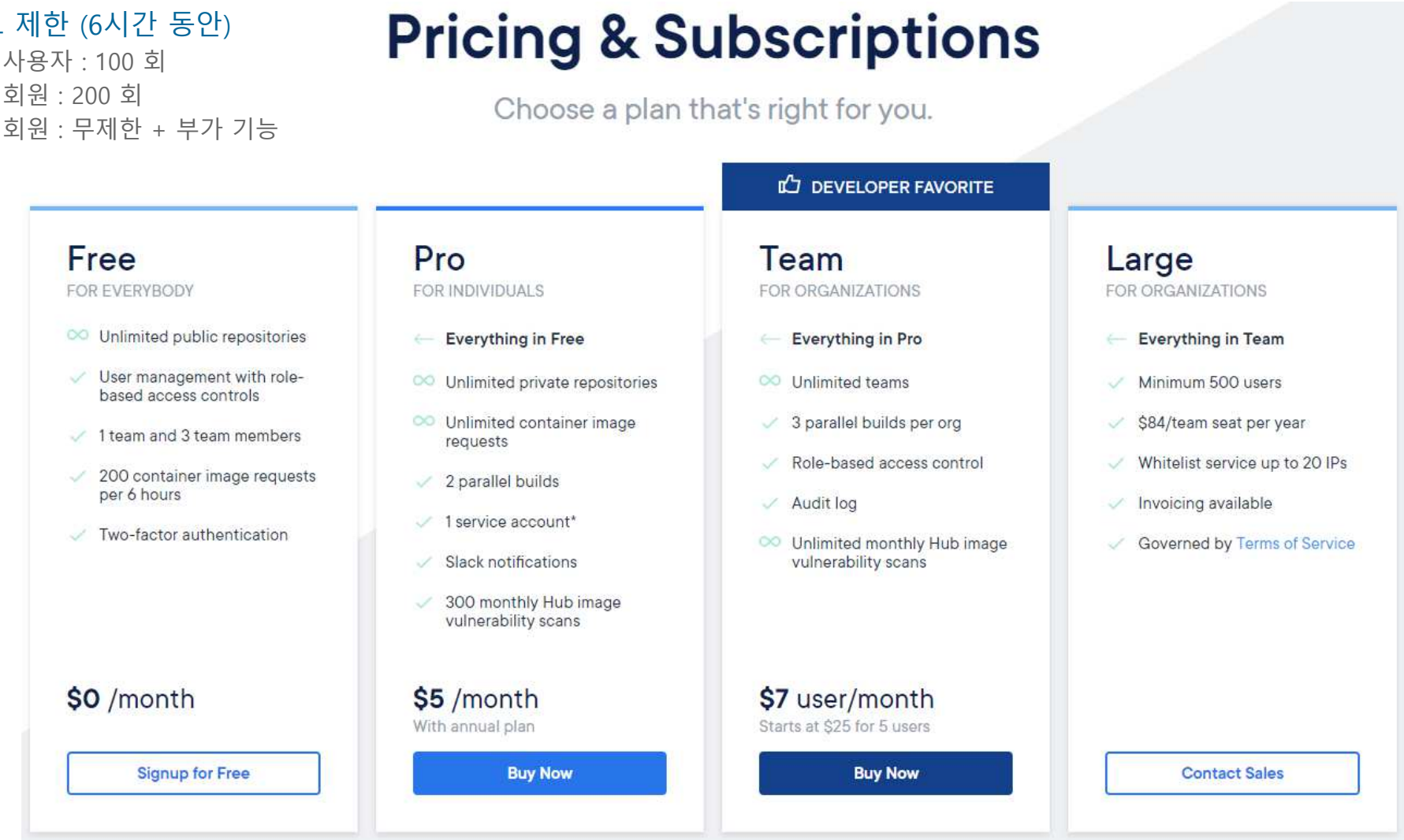
Appendix

A. Pull rate limits
B. 도커 한방에 정리
C. 실습에 필요한 도커 이미지 로드
D. cgroup 데모 (디바이스 제어)
E. 패키지를 이용한 Registry Server 구성

Pull rate limits for certain users are being introduced to Docker Hub starting November 2nd.

- Understanding Docker Hub Rate Limiting
  - 다운로드 제한 (6시간 동안)
    - 익명 사용자 : 100 회
    - 무료 회원 : 200 회
    - 유료 회원 : 무제한 + 부가 기능

2020년 12월 이후



도커 한방에 정리

- https://youtu.be/LXJhA3VWXFA

실습에 필요한 도커 이미지 로드

- **hub.docker.com 의 pull 횟수 제한에 따른 조치**

① 실습에 필요한 도커 이미지 로드 - **dockeredu 호스트**

```
[root@dockeredu ~]# cd
[root@dockeredu ~]# wget http://jadecross.iptime.org:7778/dockeredu/docker_images_dockeredu.tar.gz
[root@dockeredu ~]# docker image load -i docker_images_dockeredu.tar.gz
* SSD 디스크에서 약 10분 소요
```

② 실습에 필요한 도커 이미지 로드 - **registry 호스트**

```
[root@dockeredu ~]# cd
[root@dockeredu ~]# docker image load -i docker_images_registry.tar.gz
```

cgroups LAB : device access control을 제어 (1/3)

- device 서브시스템 cgroup을 만들어서 /dev/null 장치에 대해 특정 프로세스의 access 기능을 제어

① cgroup 서브시스템 확인

```
[root@dockeredu ~]# cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset          3               1               1
cpu             7               1               1
cpuacct         7               1               1
memory          11              1               1
devices         4               53              1
freezer         6               1               1
net_cls         8               1               1
blkio           2               1               1
perf_event      10              1               1
hugetlb         5               1               1
pids            9               1               1
net_prio        8               1               1
```

② /sys/fs/cgroup/devices 폴더로 이동하고 파일 내용을 확인

```
[root@dockeredu ~]# cd /sys/fs/cgroup
[root@dockeredu cgroup]# ls
blkio  cpu  cpuacct  cpu,cpuacct  cpuset  devices  freezer  hugetlb  memory  net_cls  net_cls,net_prio  net_prio  perf_event  pids  system
[root@dockeredu cgroup]# cd devices
[root@dockeredu devices]# ls -l
total 0
-rw-r--r--  1 root root 0 May  9 10:35 cgroup.clone_children
--w--w--w-  1 root root 0 May  9 10:35 cgroup.event_control
-rw-r--r--  1 root root 0 May  9 10:35 cgroup.procs
-r--r--r--  1 root root 0 May  9 10:35 cgroup.sane_behavior
--w-------  1 root root 0 May  9 10:35 devices.allow
--w-------  1 root root 0 May  9 10:35 devices.deny
-r--r--r--  1 root root 0 May  9 10:35 devices.list
-rw-r--r--  1 root root 0 May  9 10:35 notify_on_release
-rw-r--r--  1 root root 0 May  9 10:35 release_agent
drwxr-xr-x 19 root root 0 May  9 11:01 system.slice
-rw-r--r--  1 root root 0 May  9 10:35 tasks
```

cgroups LAB : device access control을 제어 (2/3)

③ devices 하위에 test 폴더를 생성하고 test 폴더 하위에 자동으로 생성된 파일 내역을 확인

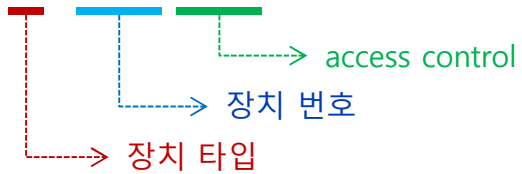
```
[root@dockeredu devices]# mkdir test
[root@dockeredu devices]# ls -l /sys/fs/cgroup/devices/test
total 0
-rw-r--r-- 1 root root 0 May  9 12:03 cgroup.clone_children
--w--w--w- 1 root root 0 May  9 12:03 cgroup.event_control
-rw-r--r-- 1 root root 0 May  9 12:03 cgroup.procs
--w------- 1 root root 0 May  9 12:03 devices.allow
--w------- 1 root root 0 May  9 12:03 devices.deny
-r--r--r-- 1 root root 0 May  9 12:03 devices.list
-rw-r--r-- 1 root root 0 May  9 12:03 notify_on_release
-rw-r--r-- 1 root root 0 May  9 12:03 tasks
```

✓ test 폴더를 생성하면 자동으로 list, allow, deny와 같은 제어파일이 생성됨
- devices.allow - 접근 허용
- devices.deny - 접근 거부
- devices.list - 사용할수 있는 장치

✓ 각제어 파일에는 4개의 필드가 존재
- device type - all / character / block
- device number - major_num.minor_num
- access control - read / write / mknod

④ devices.list 파일의 내용을 조회하고, 디폴트로 모든 장치에 모든 권한이 있음을 확인

```
[root@dockeredu devices]# cat /sys/fs/cgroup/devices/test/devices.list
a *:* rwm
```



cgroups LAB : device access control을 제어 (3/3)

⑤ tasks 파일에 현재 쉘의 PID를 추가한다.
└▶ tasks 파일에 현재 쉘이 추가되면 현재 쉘에서 동작하는 모든 하위 프로세스는 /sys/fs/cgroup/devices/test cgroup의 속성을 적용 받게 된다

```
[root@dockeredu devices]# echo $$
3403
[root@dockeredu devices]# cat /sys/fs/cgroup/devices/test/tasks
[root@dockeredu devices]# echo $$ > /sys/fs/cgroup/devices/test/tasks
[root@dockeredu devices]# cat /sys/fs/cgroup/devices/test/tasks
3403 ←  현재 쉘의 PID
3431 ←  현재 쉘에서 수행한 cat 명령의 PID
```

⑥ devices.deny 파일에 a (all)를 기록한 후 /dev/null 장치에 acces를 시도해보고, 접근이 차단되는 것을 확인

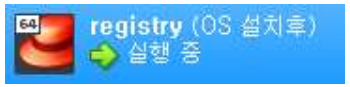
```
[root@dockeredu ~]# echo "hello world" > /dev/null
[root@dockeredu ~]# echo a > /sys/fs/cgroup/devices/test/devices.deny
[root@dockeredu ~]# echo "hello world" > /dev/null
-bash: /dev/null: Operation not permitted
```

⑦ devices.allow 파일에 a (all)를 기록한 후 /dev/null 장치에 acces를 시도해보고, 접근이 되는 것을 확인

```
[root@dockeredu ~]# echo a > /sys/fs/cgroup/devices/test/devices.allow
[root@dockeredu ~]# echo "hello world" > /dev/null
[root@dockeredu ~]#
```

패키지를 이용한 Registry Server 구성 (1/6)

① registry 가상머신을 시작하고 ssh 접속




```
[root@registry ~]#
```

② docker-ce-18.06.2.ce-3.el7 버전 설치

```
1. TCP 5000번 방화벽 포트 해제
[root@registry ~]# firewall-cmd --add-port=5000/tcp --permanent
[root@registry ~]# firewall-cmd --reload

2. 도커 설치
[root@registry ~]# yum install -y yum-utils device-mapper-persistent-data lvm2
[root@registry ~]# yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
[root@registry ~]# yum install -y docker-ce-20.10.0
[root@registry ~]# systemctl enable docker
[root@registry ~]# systemctl start docker
```

③ /etc/docker/daemon.json 파일 생성하고 docker 데몬을 재시작
→ 192.168.56.0/24 네트워크의 HOST 들은 인증서 기반의 통신이 완전한 인증 절차를 거치지 않고도, 이 레지스트리의 이미지를 가져올 수 있도록 daemon.json 파일을 생성

```
[root@registry ~]# vi /etc/docker/daemon.json
{
  "insecure-registries" : ["192.168.56.0/24"]
}
[root@registry ~]# systemctl restart docker
```

④ /etc/hosts 파일 내용 확인

```
[root@registry ~]# cat /etc/hosts
~~~

192.168.56.91  dockeredu   dockeredu.jadeedu.com
192.168.56.92  registry    registry.jadeedu.com
```

패키지를 이용한 Registry Server 구성 (2/6)

⑤ yum을 이용하여 docker-distribution 패키지 설치

```
[root@registry ~]# yum install -y docker-distribution
~~~
Installed:
  docker-distribution.x86_64 0:2.6.2-2.git48294d9.el7

Complete!
```

⑥ 인증서 생성

```
[root@registry ~]# mkdir -p /etc/certs; cd $_
[root@registry certs]# openssl req -newkey rsa:4096 -nodes -sha256 -keyout jadeedu.com.key -x509 -days 365 -out jadeedu.com.crt
Generating a 4096 bit RSA private key
..............................................++
..................++
writing new private key to 'jadeedu.com.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:KR
State or Province Name (full name) []:Korea
Locality Name (eg, city) [Default City]:Seoul
Organization Name (eg, company) [Default Company Ltd]:JADECROSS
Organizational Unit Name (eg, section) []:TA
Common Name (eg, your name or your server's hostname) []:registry.jadeedu.com
Email Address []:coordinatorj@registry.jadeedu.com
[root@registry certs]#
[root@registry certs]# ls -l
total 8
-rw-r--r--. 1 root root 2167 May  1 21:25 jadeedu.com.crt
-rw-r--r--. 1 root root 3272 May  1 21:25 jadeedu.com.key
```

패키지를 이용한 Registry Server 구성 (3/6)

⑦ 레지스트리 구성 yaml 파일(/etc/docker-distribution/registry/config.yml)에 인증서 및 네트워크 정보 설정

```
[root@registry ~]# vim /etc/docker-distribution/registry/config.yml
version: 0.1
log:
  fields:
    service: registry
storage:
    cache:
        layerinfo: inmemory
    filesystem:
        rootdirectory: /var/lib/registry
http:
    addr: 192.168.56.92:5000
    net: tcp
    host: https://registry.jadeedu.com:5000
    tls:
      certificate: /etc/certs/jadeedu.com.crt
      key: /etc/certs/jadeedu.com.key
auth:
    htpasswd:
      realm: jadeedu.com
      path: /etc/certs/dockerpasswd
```

⑧ htpasswd 프로그램 사용을 위해 httpd 패키지 설치

```
[root@registry ~]# yum install -y httpd
~~~
Installed:
  httpd.x86_64 0:2.4.6-89.el7.centos

Dependency Installed:
  apr.x86_64 0:1.4.8-3.el7_4.1    apr-util.x86_64 0:1.5.2-6.el7    httpd-tools.x86_64 0:2.4.6-89.el7.centos    mailcap.noarch 0:2.1.41-2.el7

Complete!
```

패키지를 이용한 Registry Server 구성 (4/6)

⑨ 인증접속에 사용할 계정 생성

```
[root@registry ~]# cd /etc/certs/
[root@registry certs]# htpasswd -c -B dockerpasswd trainee
New password: docker
Re-type new password: docker
Adding password for user trainee
```

⑩ 데몬 실행

```
[root@registry ~]# systemctl restart docker.service
[root@registry ~]# systemctl restart docker-distribution.service
[root@registry ~]# systemctl enable docker.service
[root@registry ~]# systemctl enable docker-distribution.service
```

⑪ 로컬에서 로그인이 되는지 확인

```
[root@registry ~]# docker login registry.jadeedu.com:5000
Username: yu3papa
Password:
Error response from daemon: login attempt to https://registry.jadeedu.com:5000/v2/ failed with status: 401 Unauthorized
[root@registry ~]# docker login registry.jadeedu.com:5000
Username: trainee
Password: docker
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

⑫ 로그인에 성공하면 ~/.docker/config.json 파일에 인증정보가 기록되는데, 해당 파일의 내용을 확인

```
[root@registry ~]# cat ~/.docker/config.json
{
        "auths": {
                "registry.jadeedu.com:5000": {
                        "auth": "dHJhaW5lZTpkb2NrZXI="
                }
        },
        "HttpHeaders": {
                "User-Agent": "Docker-Client/18.06.2-ce (linux)"
        }
}
```

패키지를 이용한 Registry Server 구성 (5/6)

⑬ Docker Hub에서 redis:5.0 이미지를 pull

```
[root@registry ~]# docker image pull redis:5.0
5.0: Pulling from library/redis
~~~
cfbdd870cf75: Pull complete
Digest: sha256:000339fb57e0ddf2d48d72f3341e47a8ca3b1beae9bdcb25a96323095b72a79b
Status: Downloaded newer image for redis:5.0
[root@registry ~]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               5.0                 a55fbf438dfd        5 weeks ago         95MB
```

⑭ redis:5.0 이미지를 registry.jadeedu.com:5000/myredis:5.0 으로 태깅하고 이미지 목록 확인

```
[root@registry ~]# docker image tag redis:5.0 registry.jadeedu.com:5000/myredis:5.0
[root@registry ~]# docker image ls
[root@registry ~]# docker image ls
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
redis                               5.0                 a55fbf438dfd        5 weeks ago         95MB
registry.jadeedu.com:5000/myredis   5.0                 a55fbf438dfd        5 weeks ago         95MB
```

⑮ myredis:5.0 이미지를 registry.jadeedu.com:5000 프라이빗 레지스트리에 push

```
[root@registry ~]# docker image push registry.jadeedu.com:5000/myredis:5.0
The push refers to repository [registry.jadeedu.com:5000/myredis]
5dacd731af1b: Pushed
5.0: digest: sha256:478392fbb4f132b36b8ead3435087553ee7e6f0395b3d64987ae195268d13493 size: 1572
```

⑯ 정상적으로 push가 되었는지 https://registry.jadeedu.com:5000/v2/_catalog 주소로 접속하여 확인

패키지를 이용한 Registry Server 구성 (6/6)

- **지금부터는 dockeredu 서버에서 작업**

① dockeredu 서버에서 registry.jadeedu.com:5000 으로 로그인 시도

```
[root@dockeredu ~]# docker login registry.jadeedu.com:5000
Username: trainee
Password: docker
Error response from daemon: Get https://registry.jadeedu.com:5000/v2/: x509: certificate signed by unknown authority
```

② /etc/docker/daemon.json 파일 생성하고 docker 데몬을 재시작

→ 192.168.56.0/24 네트워크의 HOST 들은 인증서 기반의 통신이 완전한 인증 절차를 거치지 않고도, 이 레지스트리의 이미지를 가져올 수 있도록 daemon.json 파일을 생성

```
[root@dockeredu ~]# vi /etc/docker/daemon.json
{
  "insecure-registries" : ["192.168.56.0/24"]
}
[root@dockeredu ~]# systemctl restart docker
```

③ dockeredu 서버에서 registry.jadeedu.com:5000 으로 로그인 재시도

```
[root@dockeredu ~]# docker login registry.jadeedu.com:5000
Username: trainee
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

④ registry.jadeedu.com:5000/myredis:5.0 이미지가 pull 되는지 확인

```
[root@dockeredu ~]# docker image pull registry.jadeedu.com:5000/myredis:5.0
5.0: Pulling from myredis
27833a3ba0a5: Already exists
cde8019a4b43: Pull complete
97a473b37fb2: Pull complete
c6fe0dfbb7e3: Pull complete
39c8f5ba1240: Pull complete
cfbdd870cf75: Pull complete
Digest: sha256:478392fbb4f132b36b8ead3435087553ee7e6f0395b3d64987ae195268d13493
Status: Downloaded newer image for registry.jadeedu.com:5000/myredis:5.0
```

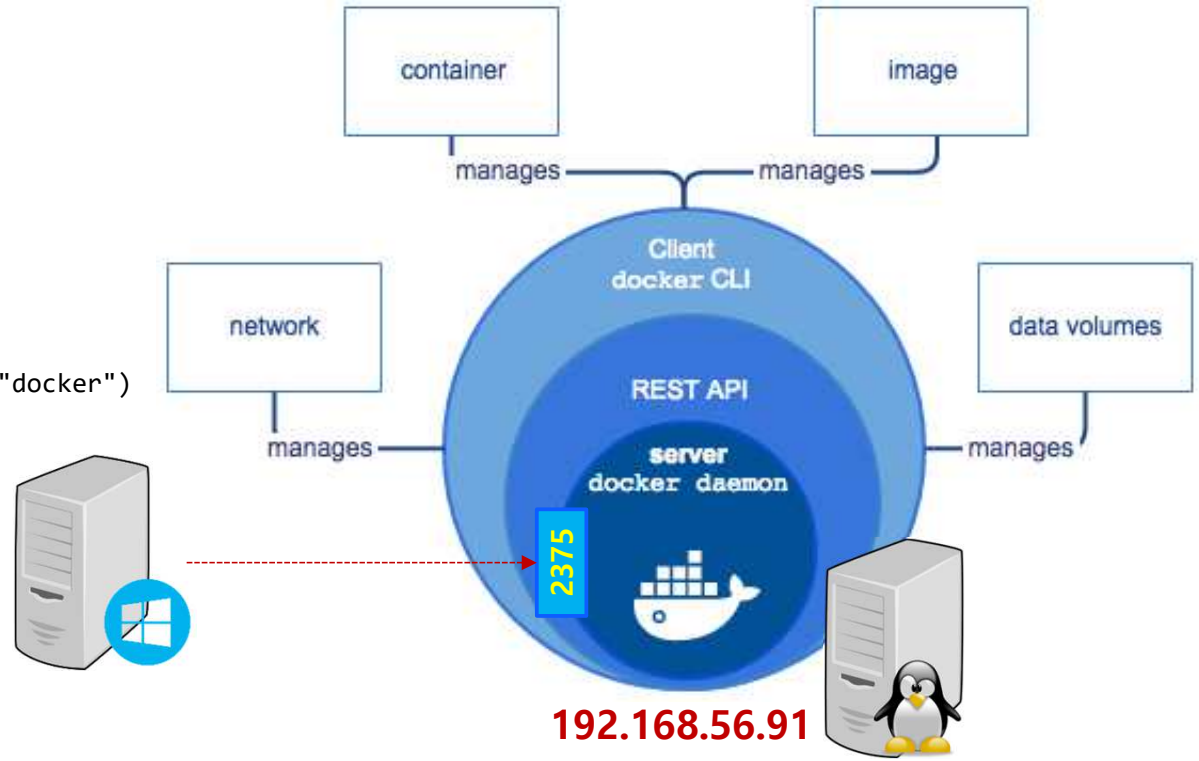
Docker 원격 접속

도커 데몬

- **dockerd**
  - https://docs.docker.com/engine/reference/commandline/dockerd/
  - 디폴트 설정으로 로컬에서 Unix Domain Socket을 이용한 접속만 가능

✓ dockerd 도움말을 출력하고 –H 옵션의 의미 파악

```
[root@dockeredu ~]# dockerd --help

Usage:  dockerd [OPTIONS]

A self-sufficient runtime for containers.

Options:
~~~ 중간생략 ~~~
  -G, --group string              Group for the unix socket (default "docker")
      --help                      Print usage
  -H, --host list                 Daemon socket(s) to connect to
~~~ 이하생략 ~~~
```



Docker 원격 실행 환경 (1/6)

- **Docker 데몬 설정**
  - 로컬에서는 Unix Domain Socket을 이용
  - 원격 클라이언트를 위해 TCP 소켓 이용

✓ dockeredu 서버에서 원격 클라이언트 접속을 위한 설정 추가
- **-H tcp://0.0.0.0**

```
[root@dockeredu ~]# vi /usr/lib/systemd/system/docker.service
```

/usr/lib/systemd/system/docker.service

```
~~~중간생략~~~
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0
ExecReload=/bin/kill -s HUP $MAINPID
~~~이하생략~~~
```

추가

✓ docker 데몬 재시작

```
[root@dockeredu ~]# systemctl daemon-reload
[root@dockeredu ~]# systemctl restart docker
[root@dockeredu ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2022-06-28 16:29:22 KST; 13s ago
     Docs: https://docs.docker.com
 Main PID: 11382 (dockerd)
    Tasks: 9
   Memory: 36.4M
   CGroup: /system.slice/docker.service
           └─11382 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0
```

```
WARNING: API is accessible on http://0.0.0.0:2375 without encryption.
         Access to the remote API is equivalent to root access on the host. Refer
         to the 'Docker daemon attack surface' section in the documentation for
         more information: https://docs.docker.com/go/attack-surface/
```

Docker 원격 실행 환경 (2/6)

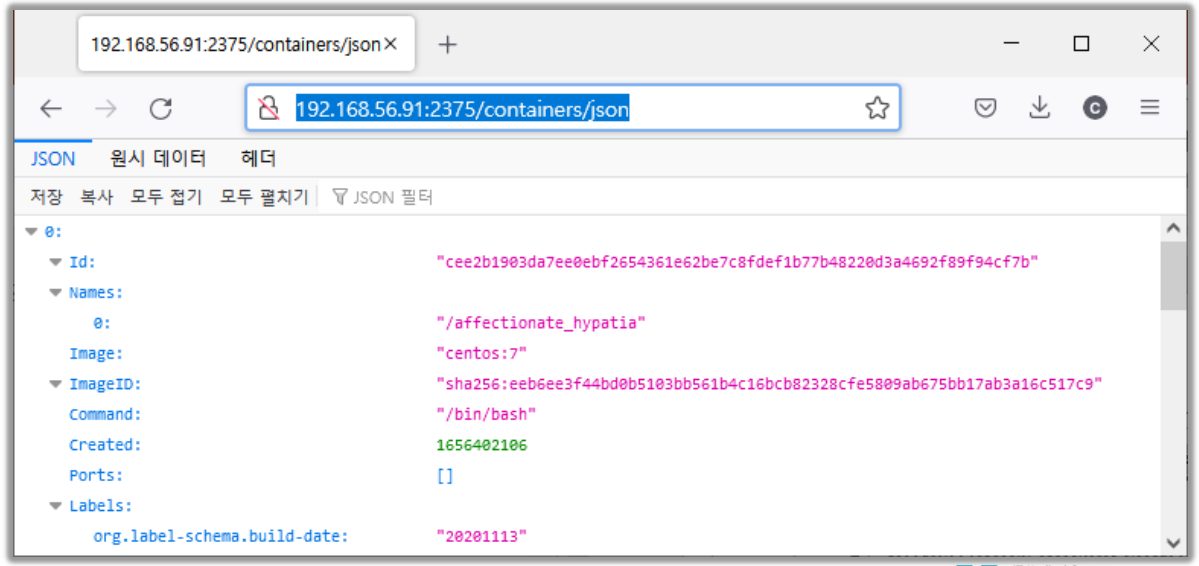
- **윈도우에서 REST API를 이용한 도커 데몬 접속**
  - 도커 REST API
    - https://docs.docker.com/engine/api/v1.41/

✓ dockeredu(192.168.56.91) 서버에서 실행중인 도커 데몬에 윈도우 머신에서 REST API를 이용하여 컨테이너 목록 출력

```
c:\> curl http://192.168.56.91:2375/containers/json
[{"Id":"cee2b1903da7ee0ebf2654361e62be7c8fdef1b77b48220d3a4692f89f94cf7b","Names":["/affectionate_hypatia"],"Image":"centos:7","ImageID":"sha256:eeb6e
e3f44bd0b5103bb561b4c16bcb82328cfe5809ab675bb17ab3a16c517c9","Command":"/bin/bash","Created":1656402106,"Ports":[],"Labels":{"org.label-schema.build-
date":"20201113","org.label-schema.license":"GPLv2","org.label-schema.name":"CentOS Base Image","org.label-schema.schema-version":"1.0","org.label-
schema.vendor":"CentOS","org.opencontainers.image.created":"2020-11-13 00:00:00+00:00","org.opencontainers.image.licenses":"GPL-2.0-
only","org.opencontainers.image.title":"CentOS Base Image","org.opencontainers.image.vendor":"CentOS"},"State":"running","Status":"Up 32
seconds","HostConfig":{"NetworkMode":"default"},"NetworkSettings":{"Networks":{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"NetworkID":"27
c52fb5e72b1949a1b33c6ddf59787668d1e359cfc4e650ee17019ccf512a4f","EndpointID":"302fac57f196fca1763a940a4831ce9617349c06e8219ba99f67e3b1784212d4","Gatew
ay":"172.17.0.1","IPAddress":"172.17.0.2","IPPrefixLen":16,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:11:0
0:02","DriverOpts":null}}},"Mounts":[]}]
```

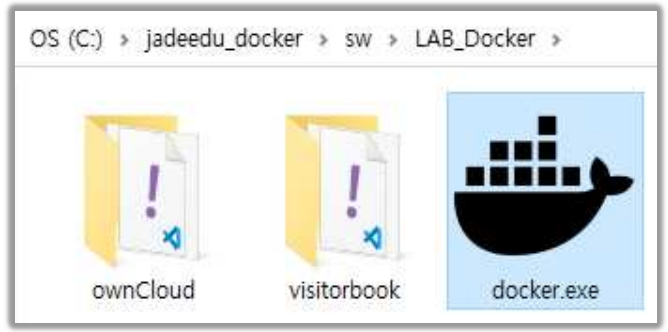
✓ 브라우저를 이용하여 원격 도커 데몬에 REST API를 이용하여 컨테이너 목록 출력

http://192.168.56.91:2375/containers/json



Docker 원격 실행 환경 (3/6)

- **윈도우용 docker 클라이언트 프로그램 이용**
  - 윈도우용 도커 클라이언트 프로그램 다운로드
    - https://download.docker.com/win/static/stable/x86_64/



✓ "C:\jadeedu_docker\sw" 경로에서 윈도우용 docker.exe 프로그램을 이용하여 dockeredu$_{(192.168.56.91)}$에서 실행중인 도커 컨테이너 목록 조회

```
C:\jadeedu_docker\sw\LAB_Docker> docker -H 192.168.56.91 container ls
CONTAINER ID   IMAGE      COMMAND       CREATED         STATUS         PORTS     NAMES
cee2b1903da7   centos:7   "/bin/bash"   7 minutes ago   Up 7 minutes             affectionate_hypatia

C:\jadeedu_docker\sw\LAB_Docker> docker -H 192.168.56.91 image ls
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
centos       7         eeb6ee3f44bd   9 months ago   204MB
```

✓ 윈도우 OS상의 PATH 환경변수 경로중 한곳에 docker.exe 파일 복사

```
C:\jadeedu_docker\sw> copy docker.exe %SystemRoot%\
C:\jadeedu_docker\sw> cd C:\
C:\> docker -H 192.168.56.91 image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
centos 7 eeb6ee3f44bd 9 months ago 204MB
```

Docker 원격 실행 환경 (4/6)

- **docker context (1/3)**
  - https://docs.docker.com/engine/reference/commandline/context/

✓ docker context 도움말을 확인하고 사용 가능한 sub command 확인

```
C:\jadeedu_docker\sw\LAB_Docker> docker context --help

Usage:  docker context COMMAND

Manage contexts

Commands:
  create      Create a context
  export      Export a context to a tar or kubeconfig file
  import      Import a context from a tar or zip file
  inspect     Display detailed information on one or more contexts
  ls          List contexts
  rm          Remove one or more contexts
  update      Update a context
  use         Set the current docker context

Run 'docker context COMMAND --help' for more information on a command.
```

Docker 원격 실행 환경 (5/6)

- **docker context (2/3)**
  - https://docs.docker.com/engine/reference/commandline/context/

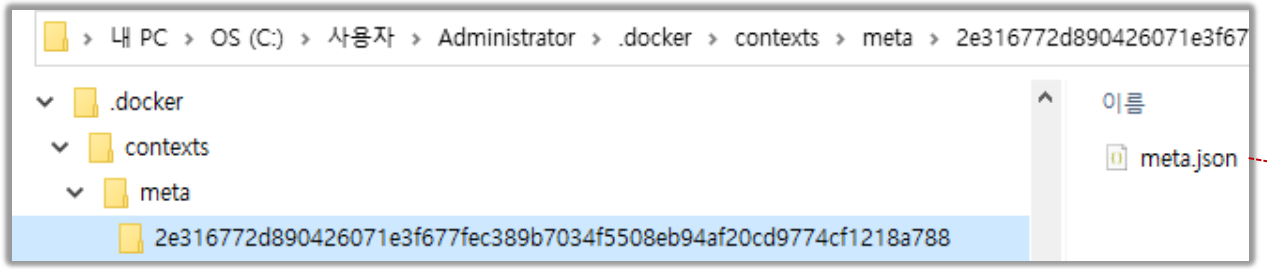
✓ 현재 docker.exe 클라이언트 머신의 context 확인

```
C:\jadeedu_docker\sw\LAB_Docker> docker context ls
NAME          DESCRIPTION                               DOCKER ENDPOINT                 KUBERNETES ENDPOINT   ORCHESTRATOR
default       Current DOCKER_HOST based configuration   npipe:////./pipe/docker_engine                        swarm
```

✓ dockeredu$_{(192.168.56.91)}$ 서버의 도커 데몬에 대한 context 를 생성하고, 조회

```
C:\jadeedu_docker\sw\LAB_Docker> docker context create --docker host=tcp://192.168.56.91:2375 dockeredu
dockeredu
Successfully created context "dockeredu“
```

→ docker context 가 생성되면 윈도우 OS 사용자의 홈경로\.docker\contexts\meta\<UUID>\meta.json 파일이 생성됨



```
{
    "Name": "dockeredu",
    "Metadata": {},
    "Endpoints": {
        "docker": {
            "Host": "tcp://192.168.56.91:2375",
            "SkipTLSVerify": false
        }
    }
}
```

```
C:\jadeedu_docker\sw\LAB_Docker> docker context ls
NAME          DESCRIPTION                               DOCKER ENDPOINT                 KUBERNETES ENDPOINT   ORCHESTRATOR
default *     Current DOCKER_HOST based configuration   npipe:////./pipe/docker_engine                        swarm
dockeredu                                               tcp://192.168.56.91:2375
```

Docker 원격 실행 환경 (6/6)

- **docker context (3/3)**
  - https://docs.docker.com/engine/reference/commandline/context/

✓ dockeredu 컨텍스트 사용 설정하고 docker 명령을 이용하여 실행중인 컨테이너 목록 조회

```
C:\jadeedu_docker\sw\LAB_Docker> docker context use dockeredu
dockeredu
Current context is now "dockeredu"
```

→ docker context 가 설정되면 홈경로\.docker\config.json 의 내용이 변경됨

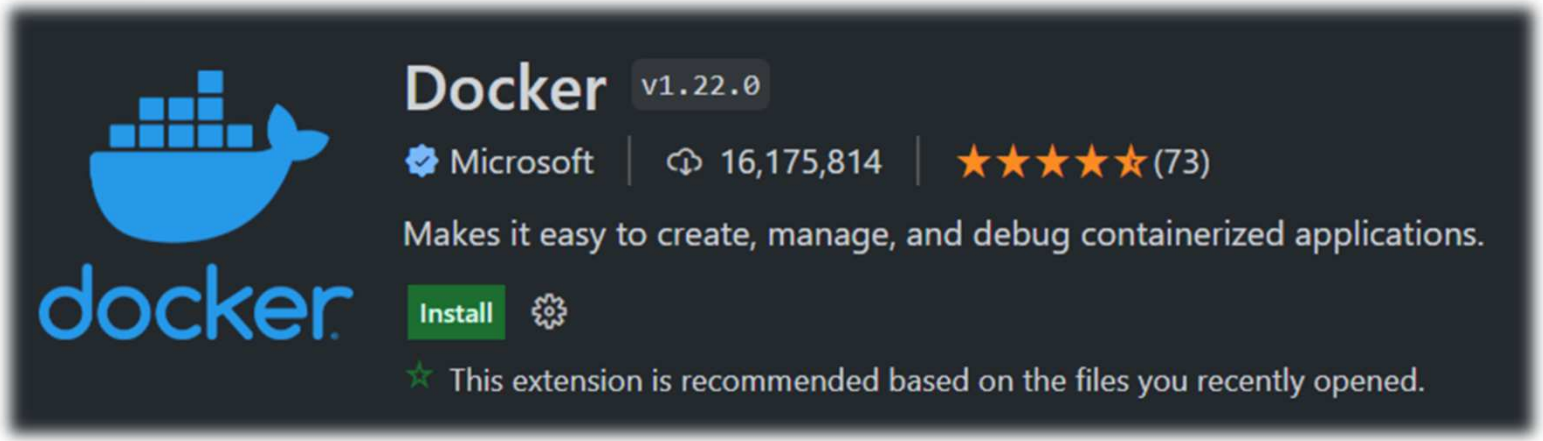
```
{
    "auths": {},
    "currentContext": "dockeredu"
}
```

```
C:\jadeedu_docker\sw\LAB_Docker> docker context ls
NAME          DESCRIPTION                               DOCKER ENDPOINT                  KUBERNETES ENDPOINT   ORCHESTRATOR
default       Current DOCKER_HOST based configuration   npipe:////./pipe/docker_engine                         swarm
dockeredu *                                             tcp://192.168.56.91:2375

C:\jadeedu_docker\sw\LAB_Docker> docker container ls
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          PORTS     NAMES
cee2b1903da7   centos:7   "/bin/bash"   37 minutes ago   Up 37 minutes             affectionate_hypatia
```

vscode로 원격 도커 컨테이너 개발환경 설정 (1/4)

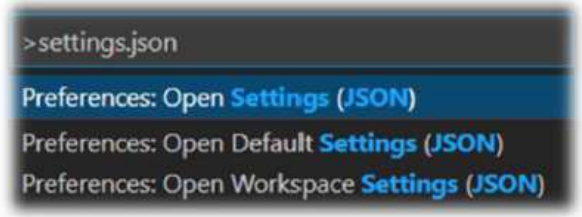
- **Docker in Visual Studio Code**
  - https://code.visualstudio.com/docs/containers/overview

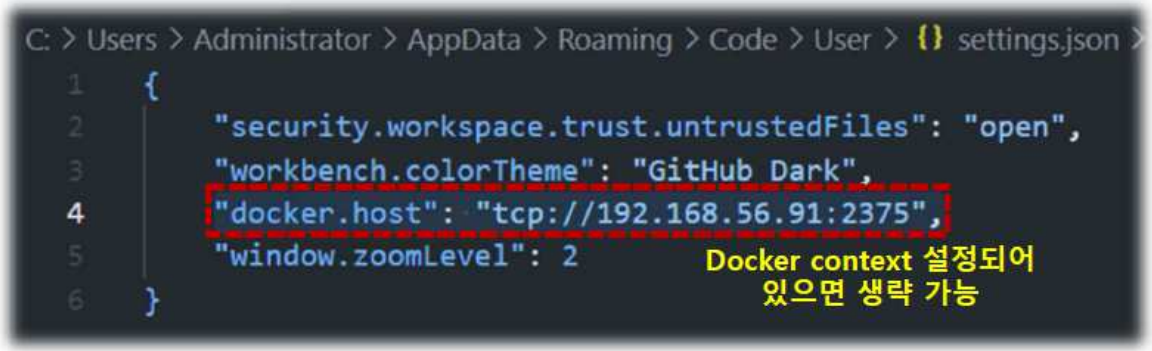
- **Extension 설치**
  - Docker



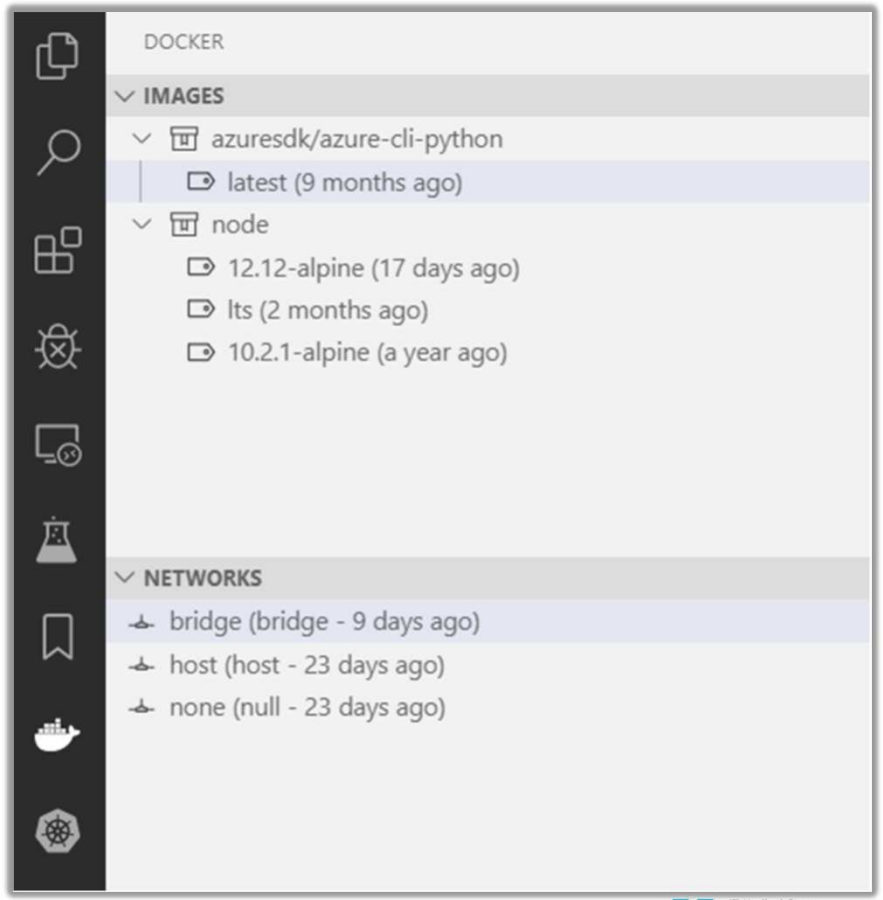
vscode로 원격 도커 컨테이너 개발환경 설정 (2/4)

- **settings.json 수정**
  - Default 나 Workspace의 settings.json 수정하면 안됨!!!





- **Docker Explorer**



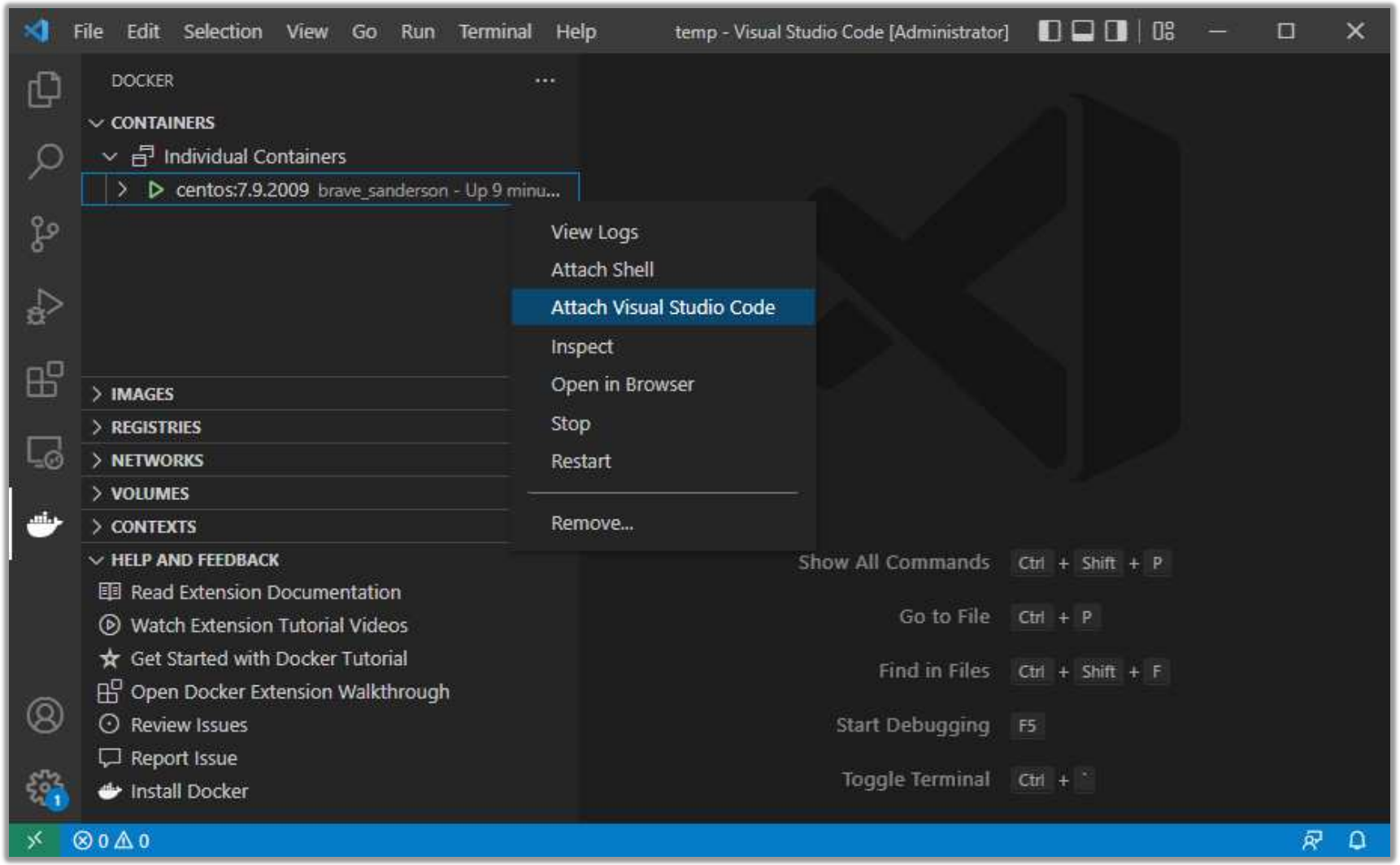
vscode로 원격 도커 컨테이너 개발환경 설정 (3/4)

- **Developing inside a Container**
  - https://code.visualstudio.com/docs/remote/containers

- **Extension 설치**
  - Remote - Containers

vscode로 원격 도커 컨테이너 개발환경 설정 (4/4)

- **Attach Visual Studio Code**

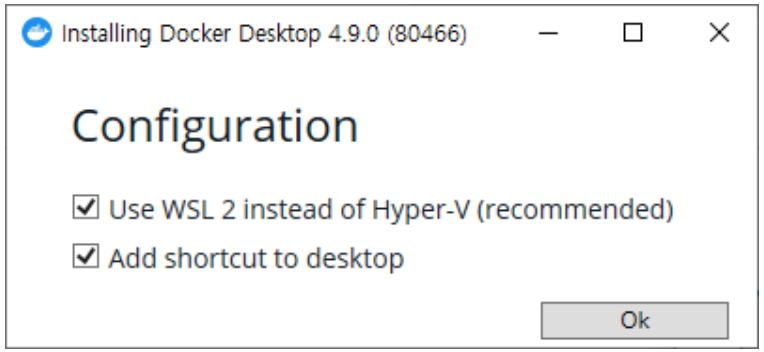


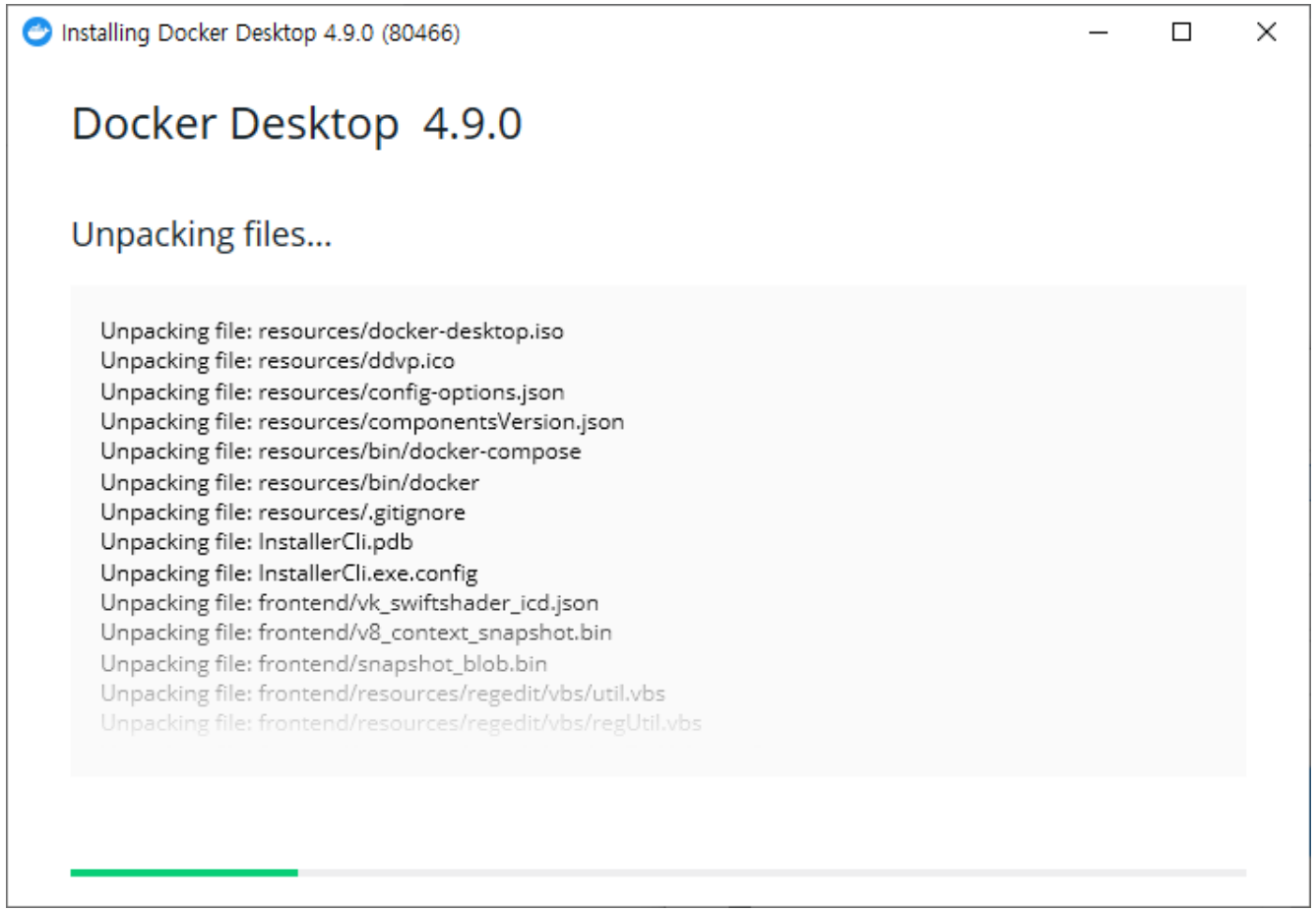
Docker Desktop

Docker Desktop 설치 (1/

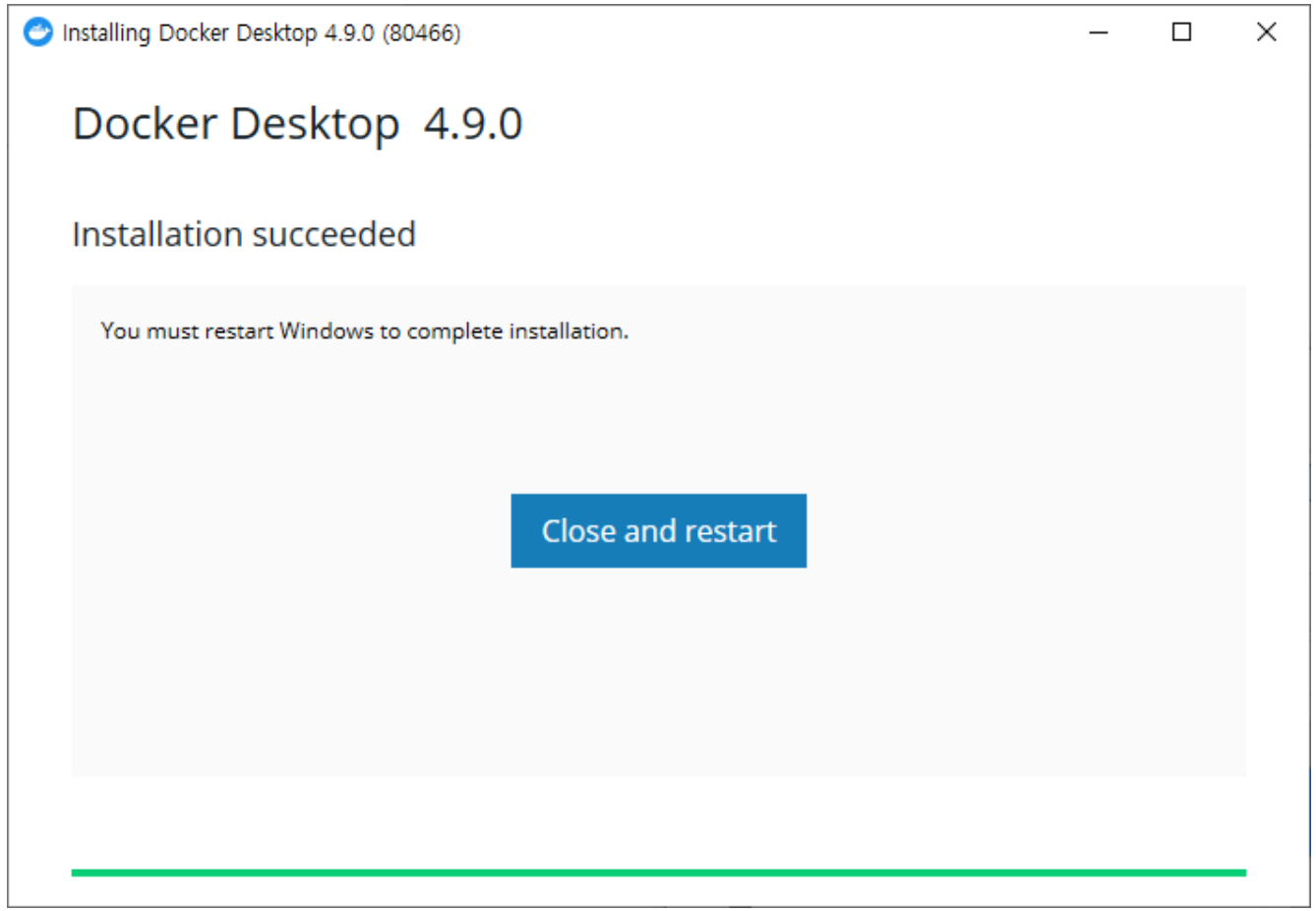
- https://www.docker.com/products/docker-desktop/

Docker Desktop 설치 (2/

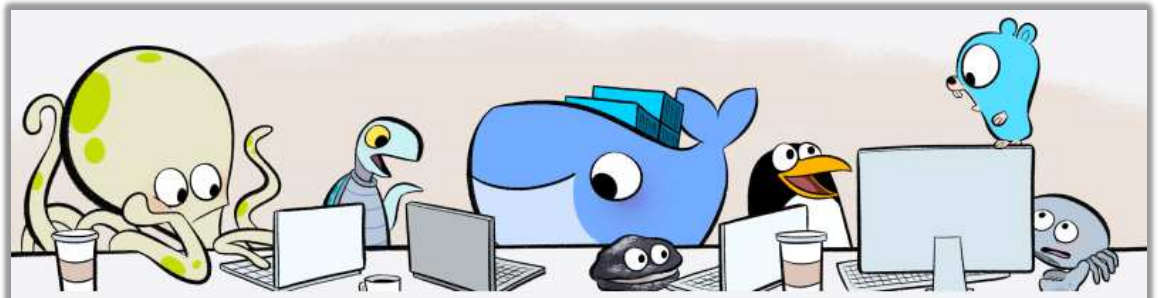




Docker Desktop 설치 (3/



Docker Desktop 설치 (4/



Our Service Agreement has Changed

We've updated the Docker Subscription Service Agreement. Please read the Blog and FAQs to learn how companies using Docker Desktop may be affected. By checking "I accept the terms" you agree to the Subscription Service Agreement, the Data Processing Agreement, and the Data Privacy Policy.

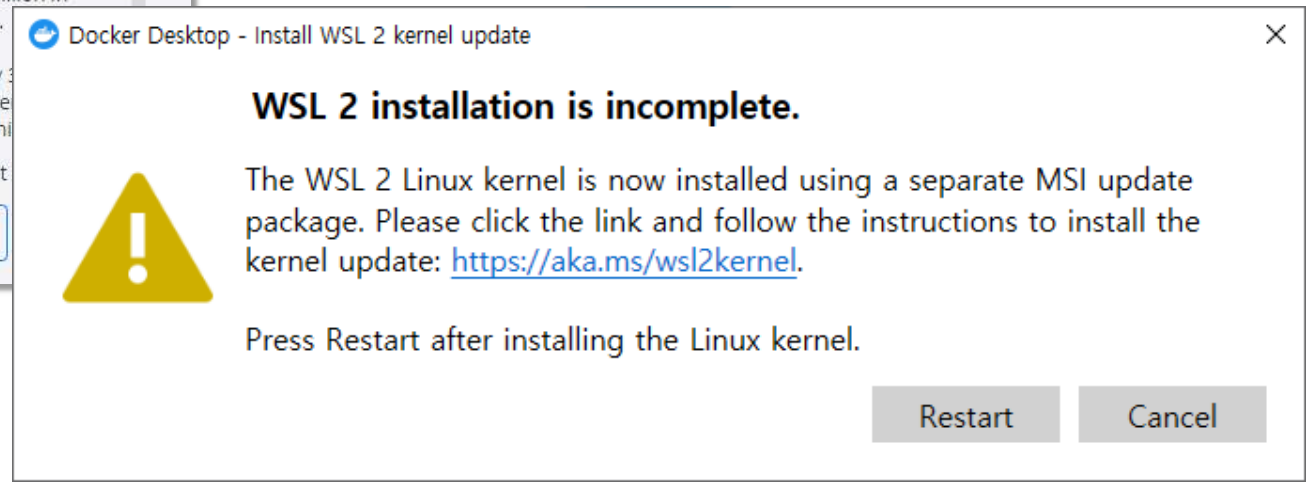
Here's a summary of key changes:

- Our Docker Subscription Service Agreement include a change to the terms of use for Docker Desktop.
  - It **remains free** for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), personal use, education, and non-commercial open source projects.
  - It requires a paid subscription for professional use in larger enterprises.
- The effective date of these terms is August 31, 2021. There was a **grace period** until January
  for those that require a paid subscription to use Docker Desktop. Docker trusts our custome
  compliance and Docker Desktop will continue to function normally after January 31st, but thi

I accept

View Full Terms

Decline and Close Application



WSL(Windows Subsystem for Linux)2 커널 업데이트 (1/

- https://aka.ms/wsl2kernel

4단계 - Linux 커널 업데이트 패키지 다운로드

1. 최신 패키지를 다운로드합니다.
   - x64 머신용 최신 WSL2 Linux 커널 업데이트 패키지

   ⓘ 참고
   ARM64 머신을 사용하는 경우 **ARM64 패키지**를 대신 다운로드하세요. 사용하고 있는 머신의 종류를 잘 모르는 경우 명령 프롬프트 또는 PowerShell을 열고 `systeminfo | find "System Type"`을 입력합니다. **주의:** 비 영어 Windows 버전에서는 "시스템 유형" 문자열을 변환하여 검색 텍스트를 수정해야 할 수 있습니다. find 명령에 대한 따옴표는 이스케이프해야 할 수도 있습니다. 예를 들어 독일어 `systeminfo | find '"Systemtyp"'` 입니다.
2. 이전 단계에서 다운로드한 업데이트 패키지를 실행합니다. (실행하려면 두 번 클릭 - 관리자 권한을 요구하는 메시지가 표시되면 '예'를 선택하여 이 설치를 승인합니다.)

설치가 완료되면 새 Linux 배포를 설치할 때 WSL 2를 기본 버전으로 설정하는 다음 단계로 이동합니다. (새 Linux 설치를 WSL 1로 설정하려면 이 단계를 건너뜁니다.)

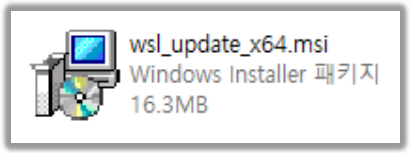
ⓘ 참고
자세한 내용은 Windows 명령줄 블로그에서 사용할 수 있는 **WSL2 Linux 커널업데이트 변경** 문서를 참조하세요.

5단계 - WSL 2를 기본 버전으로 설정

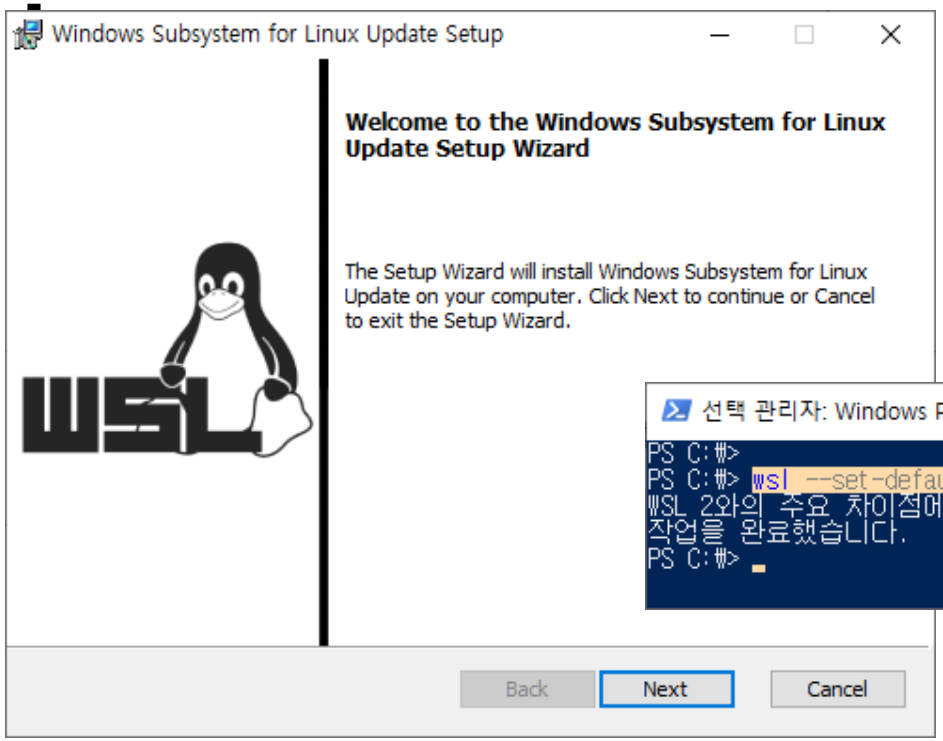
PowerShell을 열고 이 명령을 실행하여 새 Linux 배포를 설치할 때 WSL 2를 기본 버전으로 설정합니다.

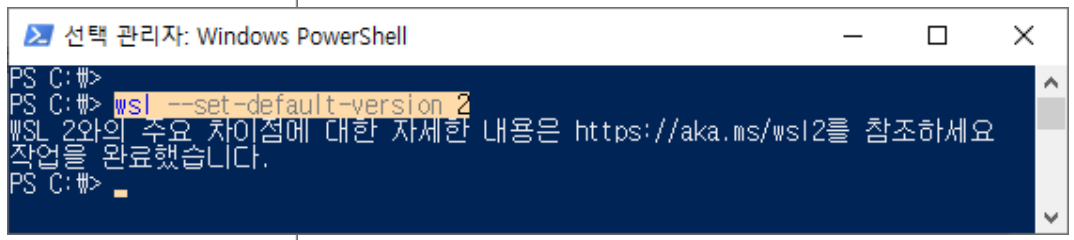
PowerShell 복사

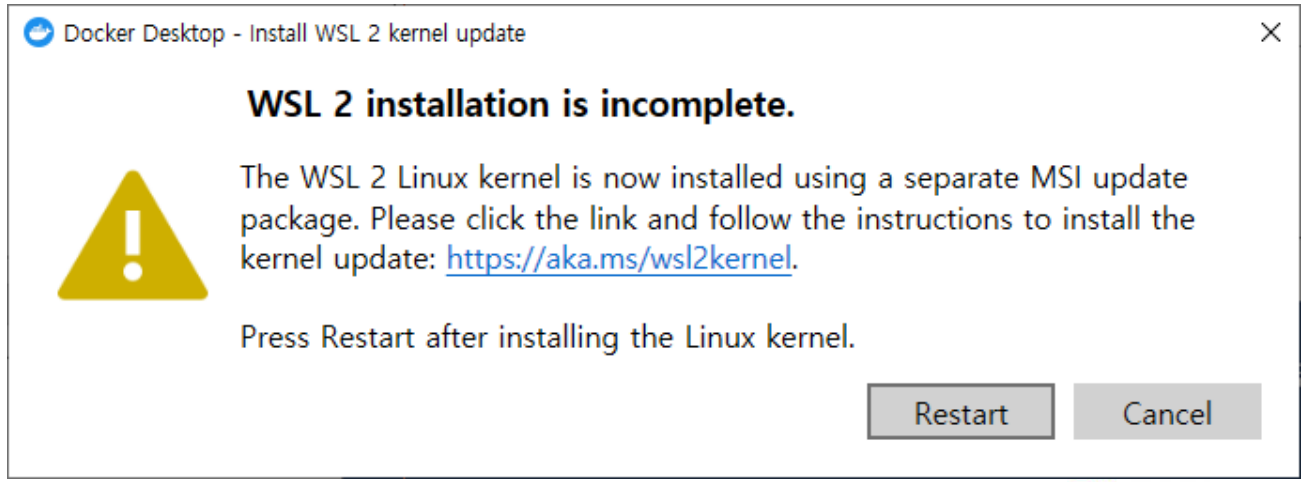
```
wsl --set-default-version 2
```



WSL(Windows Subsystem for Linux)2 커널 업데이트 (2/

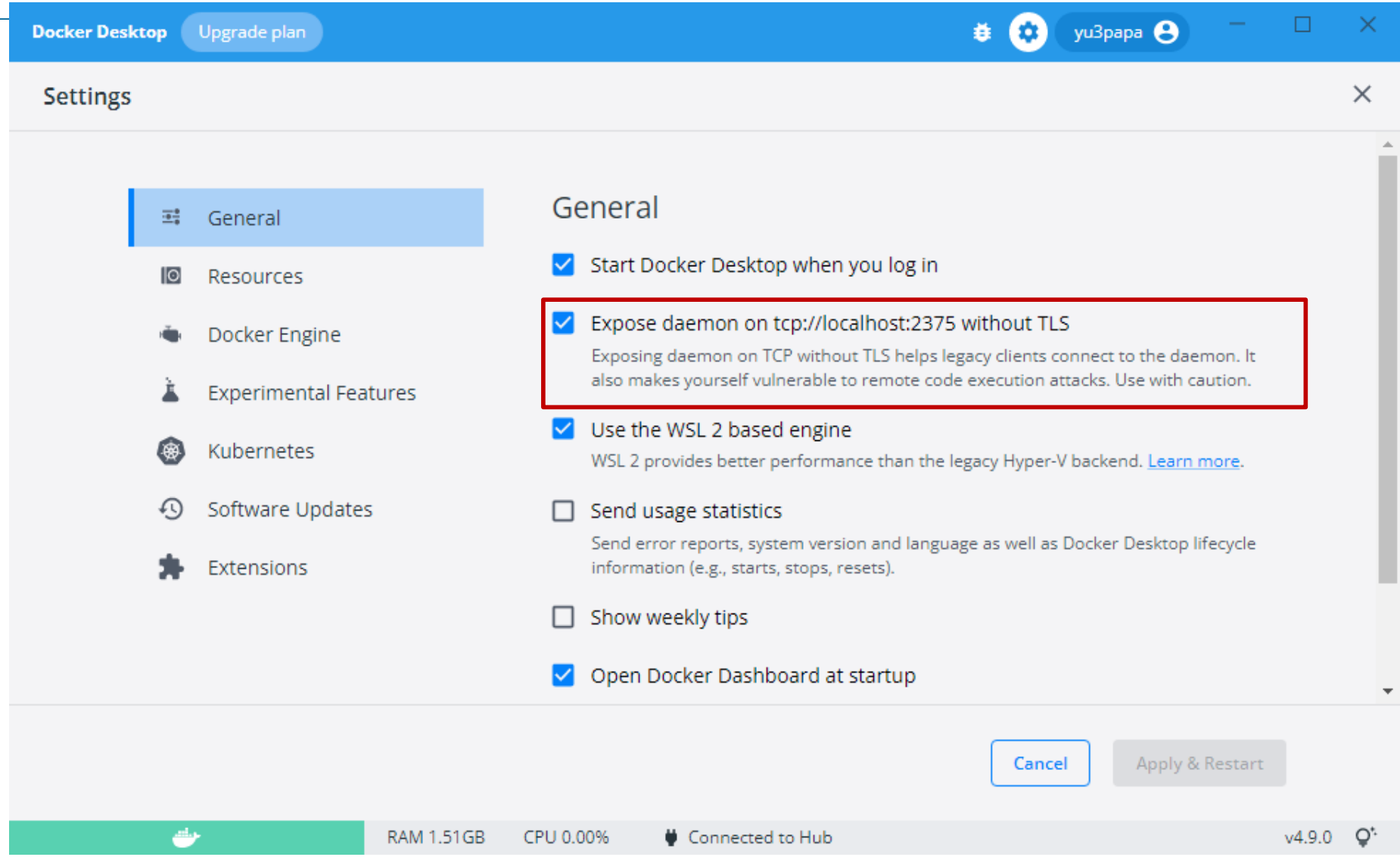






Vscode로 원격 도커 컨테이너 개발환경 설정 (1/

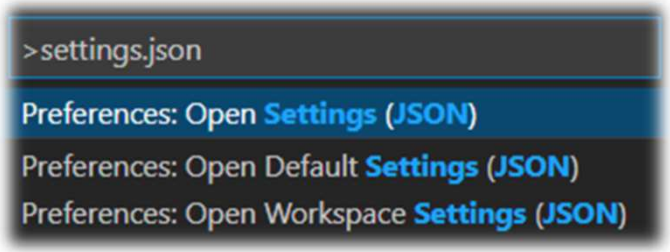
- .

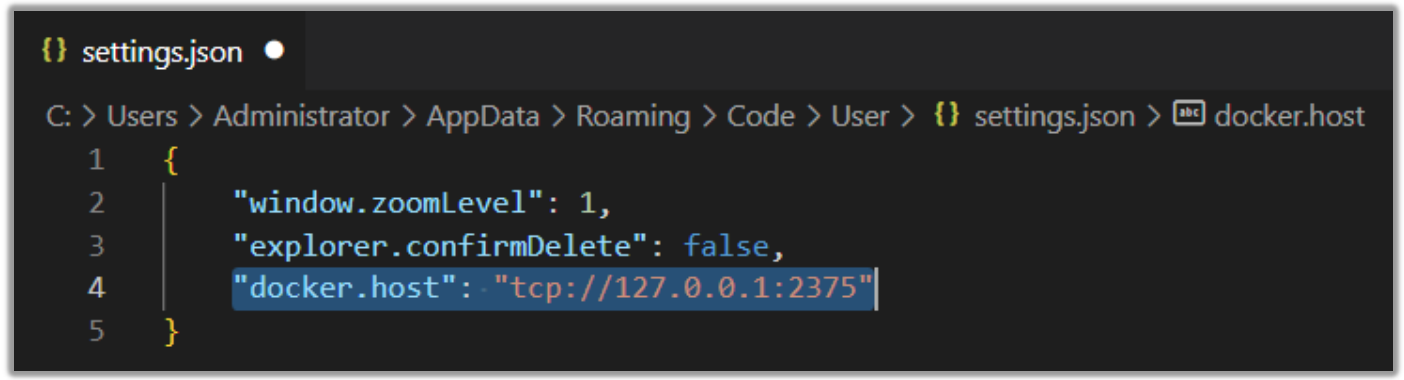


https://youtu.be/PgfH94rWsv8

Vscode로 원격 도커 컨테이너 개발환경 설정 (3/

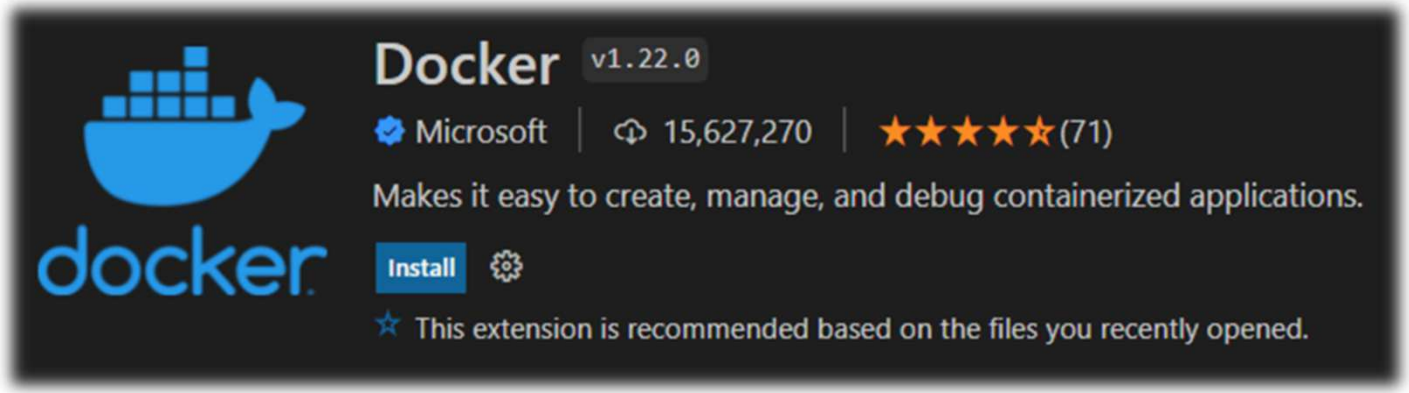
- **settings.json 수정**
  - Default 나 Workspace의 settings.json 수정하면 안됨!!!





Vscode로 원격 도커 컨테이너 개발환경 설정 (2/

- **Extension 설치**
  - Docker



Vscode로 원격 도커 컨테이너 개발환경 설정 (2/

- **Extension 설치**
  - Remote - Containers

Vscode로 원격 도커 컨테이너 개발환경 설정 (2/

- **Attach Visual Studio Code**