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

컨테이너의 역사

- FreeBSD 의 jail
- Solaris 의 zone
- Linux 의 LXC(namespace: 프로세스를 실행할 때 시스템의 리소스를 분리해서 실행할 수 있도록 도와주는 기능 와 cgroups: 자원할당 관리 로 구성)

일반적인 시스템 개발 흐름

개발환경 - 테스트환경 - 스테이징환경 - 제품환경

Docker를 활용한 시스템 개발 흐름

Git repository -> 빌드 -> Docker registry -> Docker image 다운로드 -> 컨테이너 실행

Docker registry : Docker image의 저장소
Container : Docker image 를 실행하거나 또는 실행가능한 상태로 생성한것.
Docker image: application 실행에 필요한 바이러리, 라이브러리가 패키징된 파일

자주 사용되는 docker 명령어

docker **pull** httpd - 다운로드 받을 repository 주소를 적지 않으면 기본적으로 docker hub 에서 다운로드.

```
[vagrant@server1 work]$ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
13808c22b207: Pull complete
6e9a8835eae4: Pull complete
4f4fb700ef54: Pull complete
b927d001db70: Pull complete
559cc51378ed: Pull complete
d2b091e65160: Pull complete
Digest: sha256:10758fe1fe13980e0d7bbdf8f0bbb40cf04e7c996248aac4ea390670b2420bd1
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

docker **image ls** == docker **image**s

```
[vagrant@server1 work]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
httpd        latest    fa0099f1c09d   5 days ago      148MB
hello        latest    af848c99cf17   13 months ago   2.33MB
```

docker **run** httpd : 컨테이너 실행

```
[vagrant@server1 work]$ docker run httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerN
message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerN
message
[Thu Apr 11 06:13:50.699781 2024] [mpm_event:notice] [pid 1:tid 140418659420032] AH00489: Apache/2.4.59 (Unix) configured
[Thu Apr 11 06:13:50.700009 2024] [core:notice] [pid 1:tid 140418659420032] AH00094: Command line: 'httpd -D FOREGROUND'
```

docker run -d httpd : 컨테이너를 백그라운드로 실행

```
[vagrant@server1 work]$ docker run -d httpd
27841c70543990d3507b9d31f01cd1c233ff0be1a8da77ccccbffc082145b9c5
```

docker [container] **ps** : 실행중인 컨테이너 출력

```
[vagrant@server1 work]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS     NAMES
27841c705439   httpd     "httpd-foreground"       About a minute ago   Up About a minute   80/tcp    silly_pike
```

docker [container] **ps -a** : 중단된 컨테이너가 까지 전부 출력

```
[vagrant@server1 work]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                     PORTS     NAMES
27841c705439   httpd     "httpd-foreground"       3 minutes ago   Up 3 minutes               80/tcp    silly_pike
edc9034f07fe   httpd     "httpd-foreground"       4 minutes ago   Exited (0) 3 minutes ago             jovial_chaum
e66ea4cdbe90   hello     "/hello"                 13 months ago   Exited (0) 13 months ago             upbeat_swartz
```

docker run -d --name centos8 centos:8

```
[vagrant@server1 work]$ docker run -d --name centos8 centos:8
Unable to find image 'centos:8' locally
8: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:8
519934af57e6413182862fd28dc2d9e029814dcc9fb8e3cc749a156932aed83d
```

도커이미지가 운영체제인 경우 컨테이너로 실행이 될때 shell 을 실행하게 되어 있습니다.

```
[vagrant@server1 work]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
27841c705439   httpd     "httpd-foreground"       7 minutes ago   Up 7 minutes   80/tcp    silly_pike
[vagrant@server1 work]$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS                      PORTS     NAMES
519934af57e6   centos:8   "/bin/bash"              2 minutes ago    Exited (0) 2 minutes ago              centos8
27841c705439   httpd      "httpd-foreground"       7 minutes ago    Up 7 minutes                80/tcp    silly_pike
edc9034f07fe   httpd      "httpd-foreground"       8 minutes ago    Exited (0) 7 minutes ago              jovial_chaum
e66ea4cdbe90   hello      "/hello"                 13 months ago    Exited (0) 13 months ago              upbeat_swartz
```

shell 을 실행하는 경우에는 -it 옵션이 필요합니다.

docker run -d name centos8-2 -it centos8

```
[vagrant@server1 work]$ docker run -d --name centos8-2 -it centos:8
112669dfe0d3c76e877820a4f1bfb2080caa5d95e071ee33a6e1a8b331e08895
[vagrant@server1 work]$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS     NAMES
112669dfe0d3   centos:8   "/bin/bash"              9 seconds ago    Up 8 seconds              centos8-2
27841c705439   httpd      "httpd-foreground"       11 minutes ago   Up 11 minutes   80/tcp    silly_pike
```

docker **exec** centos8 /bin/cal : 컨테이너 내의 cal 실행

```
[vagrant@server1 work]$ docker run -d --name centos8-2 -it centos:8
112669dfe0d3c76e877820a4f1bfb2080caa5d95e071ee33a6e1a8b331e08895
[vagrant@server1 work]$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS     NAMES
112669dfe0d3   centos:8   "/bin/bash"              9 seconds ago    Up 8 seconds              centos8-2
27841c705439   httpd      "httpd-foreground"       11 minutes ago   Up 11 minutes   80/tcp    silly_pike
```

docker **exec -it** centos8 /bin/bash ; 컨테이너 내부로 진입하게 된다.

```
[vagrant@server1 work]$ docker exec -it centos8-2 /bin/bash
[root@112669dfe0d3 /]# hostname
112669dfe0d3
[root@112669dfe0d3 /]#
```

container 삭제

docker [container] rm centos8-2

```
[vagrant@server1 work]$ docker rm centos8-2
Error response from daemon: You cannot remove a running container 112669dfe0d3c76e877820a4f1
bfb2080caa5d95e071ee33a6e1a8b331e08895. Stop the container before attempting removal or forc
e remove
```

삭제하기전에 중단을 먼저해야 한다.

(참고로 중단하지 않고 강제로 삭제하려면 docker [container] rm -f centos8-2)

docker [container] **stop** centos8-2

```
[vagrant@server1 work]$ docker stop centos8-2
centos8-2
[vagrant@server1 work]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
27841c705439   httpd     "httpd-foreground"       23 minutes ago   Up 23 minutes   80/tcp    silly_pike
[vagrant@server1 work]$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS                      PORTS
 NAMES
1de0a4b39888   centos:8   "/bin/bash"              42 seconds ago   Exited (0) 13 seconds ago
 centos8-2
519934af57e6   centos:8   "/bin/bash"              19 minutes ago   Exited (0) 19 minutes ago
 centos8
27841c705439   httpd      "httpd-foreground"       24 minutes ago   Up 24 minutes               80/tcp
 silly_pike
edc9034f07fe   httpd      "httpd-foreground"       25 minutes ago   Exited (0) 24 minutes ago
 jovial_chaum
e66ea4cdbe90   hello      "/hello"                 13 months ago    Exited (0) 13 months ago
 upbeat_swartz
[vagrant@server1 work]$ docker rm centos8-2
centos8-2
[vagrant@server1 work]$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS                      PORTS
 NAMES
519934af57e6   centos:8   "/bin/bash"              19 minutes ago   Exited (0) 19 minutes ago
 centos8
27841c705439   httpd      "httpd-foreground"       24 minutes ago   Up 24 minutes               80/tcp
 silly_pike
edc9034f07fe   httpd      "httpd-foreground"       26 minutes ago   Exited (0) 24 minutes ago
 jovial_chaum
e66ea4cdbe90   hello      "/hello"                 13 months ago    Exited (0) 13 months ago
 upbeat_swartz
[vagrant@server1 work]$
```

- 컨테이너 id 만 출력할수도 있다.

```
[vagrant@server1 work]$ docker container ps -q
27841c705439
[vagrant@server1 work]$ docker ps -aq
519934af57e6
27841c705439
edc9034f07fe
e66ea4cdbe90
```

- 여러개의 컨테이너를 한번에 전부 중단하거나 삭제 하려면 컨테이너 id 를 사용하면 된다.

docker container **stop $(docker ps -q)** : **실행중인 컨테이너 전부 중단**

docker container **rm $(docker ps -aq)** : 모든 컨테이너 전부 삭제

alias 설정을 하면 명령어를 간단하게 줄여쓸수 있다.
$ alias all_container_rm='docker container stop $(docker ps -q) ; docker container rm $(docker ps -aq)'
새로 로그인하더라도 alias 를 계속사용하려면 사용자 홈디렉토리 아래에 **.bashrc** 에 등록하면 된다.
$ echo "alias all_container_rm='docker container stop $(docker ps -q) ; docker container rm $(docker ps -aq)' >> ~/.bashrc
* 새로 로그인하거나 아니면 **source ~/.bashrc** 하면 바로 적용된다.

$ systemctl status docker.service
● docker.service - Docker Application Container Engine
Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
Active: active (running) since Fri 2023-09-01 03:39:39 UTC; 8h ago
Docs: https://docs.docker.com
Main PID: 765 (dockerd)
Tasks: 11
Memory: 325.5M
CGroup: /system.slice/docker.service
└─765 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
docker daemon 이 실행될 때 의존관계인 containerd 서비스가 자동으로 올라간다.

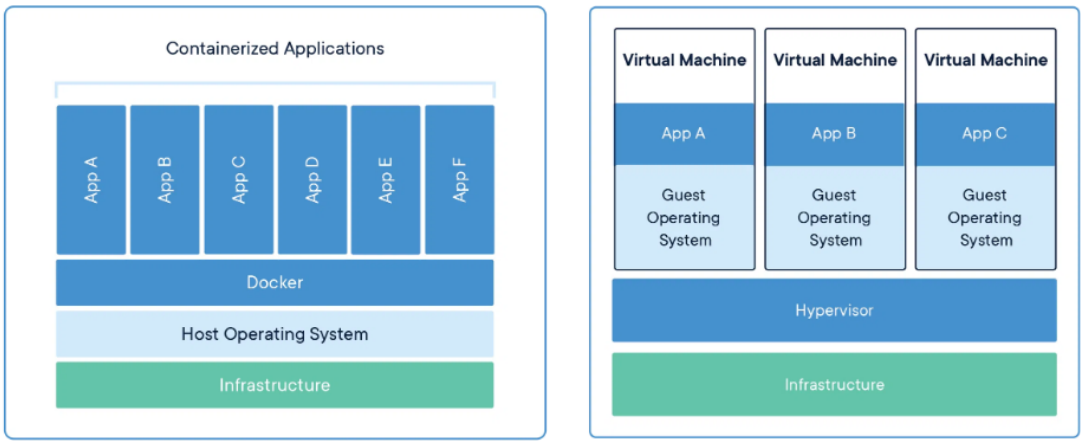
*. docker daemon 이 inactive 상태이면
systemctl start docker
* 그리고 부팅할때 자동으로 docker daemon 이 실행이 되도록 하려면
systemctl enable docker

containerd 데몬은 아래 명령의 결과에서 볼수 있듯이 docker daemon과 종속관계이므로
docker daemon 이 start 되면 containerd 데몬은 자동으로 실행이 된다.

$ sudo systemctl list-dependencies docker.service
docker.service
● ├─**containerd.service**
● ├─docker.socket
● ├─system.slice
…

일반계정 권한으로 docker 명령어를 실행할수 없다.
$ docker container ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/json": dial unix **/var/run/docker.sock:** connect: **permission denied**
$ ls -l /var/run/docker.sock
srw-rw----. 1 root docker 0 Sep 1 03:39 /var/run/docker.sock : 권한문제로 실행할수 없다.
$ sudo usermod -**aG docker** vagrant ; vagrant 그룹에 속하도록 하면 docker.sock 파일에 엑세스 할수 있다.
새로 로그인하면 vagrant 계정이 docker 명령어를 사용할수 있게 된다.

container vs vm



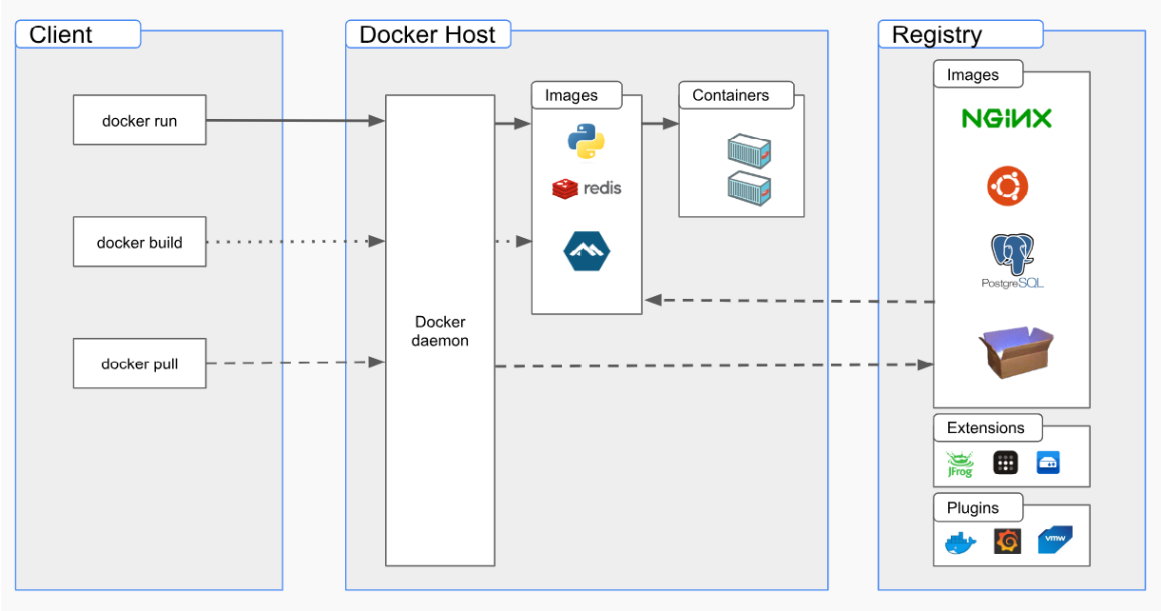
container

어플리케이션이 실행되는데 필요한 코드와 종속성을 함께 패키징.
여러 컨테이너가 동일한 시스템에서 실행될수 있으며, 각 컨테이너는 격리된 프로세스로 실행이 된다.
컨테이너는 vm 보다 필요한 리소스가 작다

vm

하나의 서버를 여러서버로 바꾸는 물리적 하드웨어의 추상화.
하이퍼바이저를 사용해서 단일시스템에서 여러 개의 vm을 실행 할 수 있다.
각각의 vm 에는 운영체제, 어플리케이션, 바이너리파일 및 라이브러리가 포함되어 있으며
일반적으로 container 에 비해서 크기가 크고 리소스를 많이 차지한다.

docker architecture



docker daemon(dockerd)

- docker api 요청을 처리하고 도커이미지, 컨테이너, 네트워크, 볼륨등 도커 오브젝트를 관리한다.
  또한 도커데몬은 도커서비스를 관리하기 위한 다른 데몬과 통신한다.

docker client(docker)

- docker run 과 같은 명령어를 도커데몬에게 전달하는 역할을 한다. 도커명령어는 docker api를 사용한다.

docker desktop

- mac, ms window 또는 리눅스 환경에서 쉽게 설치할수 있다. 도커 데스크탑은 도커서버 및 클라이언트, 도커 컴포즈를 포함한다.

docker registries

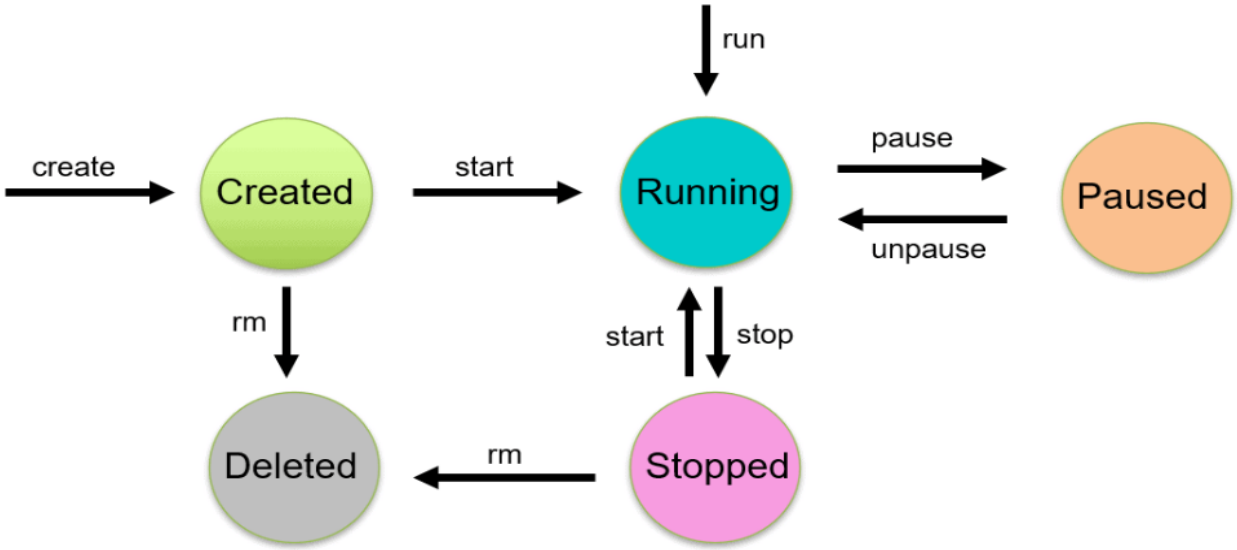
- docker image 가 저장되어 있는 장소, 퍼블릭 도커허브는 누구나 사용할수 있다. private registry를 사용하려면 docker hub 에 로그인을 해야 한다.

- **docker images**

도커 컨테이너를 생성하기 위한 읽기전용 템플릿이미지.

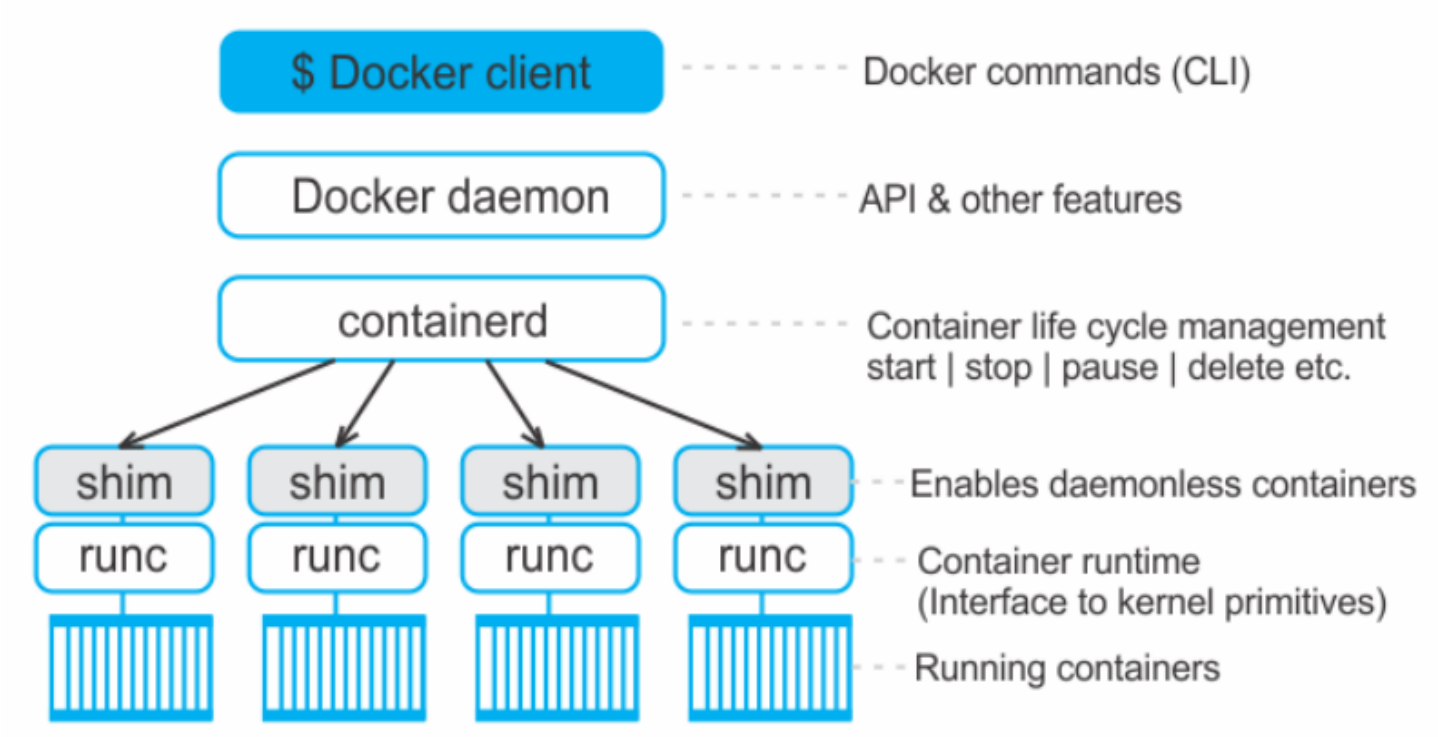
컨테이너

- 도커 이미지의 실행가능한 상태.



docker container lifecycle

created : 컨테이너 생성
running: 컨테이너 실행
paused: 컨테이너 실행 일시정시
stooped: 컨테이너 중단
deleted: 컨테이너 삭제



도커 아키텍처

containerd - high level container runtime
runc - low level container runtime
runc 는 컨테이너를 생성할때 실행되고 생성이 끝나면 종료된다.

```
Every 0.2s: ps -ef |grep -i runc                                                Sat Apr 13 02:45:36 2024

vagrant   1729  1313  2 02:43 pts/2    00:00:02 watch -n 0.2 ps -ef |grep -i runc
root      4587     1  0 02:45 ?        00:00:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id f13d055
root      4595  4587  0 02:45 ?        00:00:00 runc --root /var/run/docker/runtime-runc/moby --log /run/con
root      4606  4595  0 02:45 ?        00:00:00 runc init
vagrant   4624  1729  0 02:45 pts/2    00:00:00 watch -n 0.2 ps -ef |grep -i runc
vagrant   4625  4624  0 02:45 pts/2    00:00:00 sh -c ps -ef |grep -i runc
vagrant   4627  4625  0 02:45 pts/2    00:00:00 sh -c ps -ef |grep -i runc
```

docker 명령어

docker image 삭제
$ docker rmi docker image 이름
$ docker rmi httpd:2.4
Untagged: httpd:2.4
Untagged: httpd@sha256:333f7bca9fb72248f301fd2ae4892b86d36cc2fdf4c6aa49a6700f27c8e06daf
Deleted: sha256:76e5ad98b58ec86acd1e38d6ff79761144292113cecee2087b8827c12a5fd0b5
Deleted: sha256:5b3e0155029265a480ac5a1426fc212e01b2fd5ecb215e215c76643e6e69efd5
Deleted: sha256:f6cf6f4fc18f2bae60358109213f42a0eaca5d47372801b9a605b7e64a83da86
Deleted: sha256:e255d2b6be9eb525534cdb2bfbf620892736fa9f5b98452d231c5fedca37b40c
Deleted: sha256:faf9bd124eae3a078e2f2478d6a7ac6a74344724a866a8d94747a862df31b175
$

container 생성
$ docker container create -it --name alpine alpine
14345f742c58769a88734088489344ec04e49b2e1144dd8e5e1dbf007919a818
$ docker container ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
14345f742c58	alpine	"/bin/sh"	6 seconds ago	**Created**		alpine

$

생성된 컨테이너 실행
$ docker container **start -i** alpine # start 는 정지중인 컨테너를 시작할 때 사용
/ # ctrl + p,q => container를 종료하지 않은채로 셸로 빠져나감

$ docker run -d --name apache httpd:2.4
755d873a005e720491dd963bb1f9c8a22dfa3de5eb66bde6a9ce791bb0a14e2f
httpd:2.4 도커이미지를 apache 라는 컨테이너 이름으로 백그라운드로 실행

$ docker run --name nginx nginx ; -d 옵션을 생략하면 foreground 로 실행된다.
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
...
ctrl + c 누르면 container 가 중단된다.

컨테이너 확인
$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES

```
983430c106f2    nginx          "/docker-entrypoint.…"    4 minutes ago    Up 4 minutes    80/tcp      nginx
755d873a005e    httpd:2.4      "httpd-foreground"        4 minutes ago    Up 4 minutes    80/tcp      apache
$
```

중단되거나 또는 실행중인 모든 컨테이너 확인은

```
$ docker ps -a
CONTAINER ID    IMAGE          COMMAND                  CREATED              STATUS                      PORTS       NAMES
287b8907d0dc    nginx          "/docker-entrypoint.…"   3 minutes ago        Exited (0) About a minute ago       nginx2
983430c106f2    nginx          "/docker-entrypoint.…"   5 minutes ago        Up 5 minutes                80/tcp      nginx
755d873a005e    httpd:2.4      "httpd-foreground"       6 minutes ago        Up 6 minutes                80/tcp      apache
$
```

컨테이너의 명령어 실행

```
$ docker exec apache hostname
755d873a005e
```

실행할 명령어를 셸을 실행하면 컨테이너 내부로 진입한다.

```
$ docker exec -it apache /bin/bash
(i: interactive, t: tty)
root@755d873a005e:/usr/local/apache2#
```

컨테이너 밖으로 나갈때는 exit

컨테이너를 전부 삭제 할때는

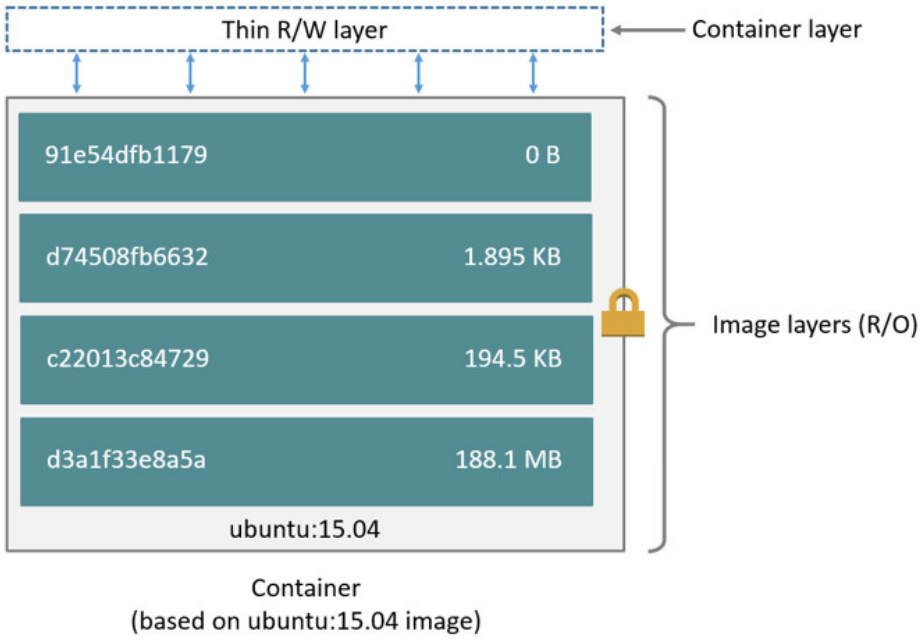
```
$ docker stop $(docker ps -q)
983430c106f2
755d873a005e
$ docker rm  $(docker ps -aq)
287b8907d0dc
983430c106f2
755d873a005e
```

```
$ docker run -d --name mydb mysql
5913234cc6af23cb13986eadb06d7cc0830d6ce832bdd901008f2c1cba4bcb8e
$ docker container ps
CONTAINER ID    IMAGE      COMMAND      CREATED             STATUS             PORTS      NAMES
3fb1e2cfc7ce    alpine     "/bin/sh"    31 minutes ago      Up 31 minutes                 tmpfs01
61b534987e9a    alpine     "/bin/sh"    38 minutes ago      Up 35 minutes                 tmpfs00
14345f742c58    alpine     "/bin/sh"    4 hours ago         Up 4 hours                    alpine
$ docker container ps -a
CONTAINER ID    IMAGE          COMMAND                  CREATED              STATUS              PORTS       NAMES
```

```
5913234cc6af   mysql     "docker-entrypoint.s…"   12 seconds ago   Exited (1) 9 seconds ago
 mydb
3fb1e2cfc7ce   alpine    "/bin/sh"                31 minutes ago   Up 31 minutes
tmpfs01
61b534987e9a   alpine    "/bin/sh"                38 minutes ago   Up 35 minutes
 tmpfs00
8595e6fcac66   alpine    "/bin/sh"                2 hours ago      Exited (0) 2 hours ago
alpine3
03db57d24a9a   alpine    "/bin/sh"                2 hours ago      Exited (0) 2 hours ago
alpine2
e1851171d89d   alpine    "14345f742c58 /bin/sh"   4 hours ago      Created
elated_carver
9a4db24f9413   alpine    "14345f742c58"           4 hours ago      Created
gifted_blackburn
0fe216434b65   alpine    "/bin/sh"                4 hours ago      Exited (0) 4 hours ago
sad_blackwell
190f9a7cb588   alpine    "/bin/sh"                4 hours ago      Exited (126) 4 hours ago
gracious_perlman
ac6fb4eea27c   centos:7  "/bin/bash"              4 hours ago      Exited (0) 4 hours ago
centos7
ea8063a450d7   alpine    "/bin/sh"                4 hours ago      Exited (0) 4 hours ago
hardcore_wilson
14345f742c58   alpine    "/bin/sh"                4 hours ago      Up 4 hours
alpine
$ docker logs mydb
2023-09-01 08:22:13+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.34-1.el8 started.
2023-09-01 08:22:13+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-09-01 08:22:13+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.34-1.el8 started.
2023-09-01 08:22:14+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not
specified
    You need to specify one of the following as an environment variable:
    - MYSQL_ROOT_PASSWORD
    - MYSQL_ALLOW_EMPTY_PASSWORD
    - MYSQL_RANDOM_ROOT_PASSWORD

$ docker run -d --name mydb --env MYSQL_ROOT_PASSWORD=mypass mysql
0e97a7cbdc47b73e3649fcd0cc6160be215e1fab8f053a4b47b8b39d1988a6be
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
          NAMES
0e97a7cbdc47   mysql     "docker-entrypoint.s…"   3 seconds ago    Up 3 seconds    3306/tcp,
33060/tcp   mydb
```

docker volume (https://docs.docker.com/storage/volumes/)



docker layer 는 read only 이미지이므로 수정이 되지 않는다. 컨테이너가 실행이되면 container layer 가 read write layer 로 생성이 된다. 로그데이터나 수정해야할 데이터는 container layer 에서 이루어진다.
thin 방식이므로 저장공간을 절약할수 있다.

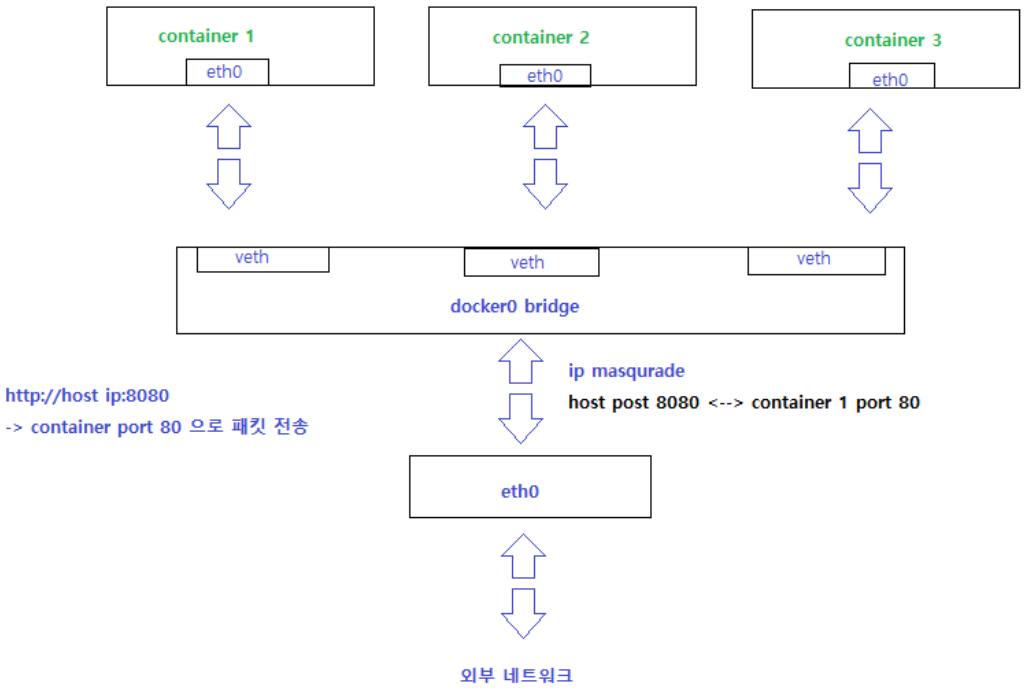
```
$ docker run --name alpine2 -it -v /myvolume alpine
/ # df -h /myvolume
Filesystem                Size      Used Available Use% Mounted on
/dev/sda1                40.0G     10.1G     29.9G  25% /myvolume

$ docker container inspect alpine2 |grep -A 11 Mounts
        "Mounts": [
            {
                "Type": "volume",
                "Name": "94531daaaa870005d749fdecb6a072550dbf4a2800f50154b4b3e84a86a2e589",
                "Source": "/var/lib/docker/volumes/94531daaaa870005d749fdecb6a072550dbf4a2800f50154b4b3e84a86a2e589/_data",
                "Destination": "/myvolume",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
```

```
$ docker volume create sample
$ docker volume ls
DRIVER      VOLUME NAME
local       sample
$ docker volume inspect sample
[
    {
        "CreatedAt": "2023-09-01T06:17:12Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/sample/_data",
        "Name": "sample",
        "Options": null,
        "Scope": "local"
    }
]
$ docker container run -it --name alpine3 -v sample:/mydata alpine
/ # df -h /mydata
Filesystem                Size      Used Available Use% Mounted on
/dev/sda1                40.0G     10.1G     29.9G  25% /mydata
/ # mount |grep mydata
/dev/sda1 on /mydata type xfs (rw,seclabel,relatime,attr2,inode64,noquota)
/ # exit
$ sudo find / -name sample
/var/lib/docker/volumes/sample

$ sudo ls /var/lib/docker/volumes/sample
_data
$ sudo ls /var/lib/docker/volumes/sample/_data
test.txt
```

docker network




```
$ docker network ls
b26884f09fc8        bridge              bridge              local
ad5d0190d3ed        host                host                local
6b449bfb4118        none                null                local
```

사용자정의 브리지

```
$ docker network create --driver=bridge mybridge
$ docker network inspect mybridge
$ docker run -d  --name mybridge_container --network=mybridge httpd:2.4

$ docker ps
2ed271a33096        httpd:2.4   "httpd-foreground"      2 seconds ago       Up          2seconds    80/tcp
mybridge_container2

$ docker network mybridge
$ docker network inspect mybridge
$ docker network disconnect mybridge mybridge_container
$ docker network inspect mybridge
$ docker network connect mybridge mybridge_container
$ docker network inspect mybridge
```

사용자 정의 브리지를 생성할때 ip 대역을 원하는 대역으로 설정할 수도 있다.

```
$ docker network create --driver bridge --subnet 172.30.0.0/16 --ip-range 172.30.0.0/24 --gateway 172.30.0.1 newbridge
4f727c15c28ce155c17702c6db357dad2fbba169c4c7502005a589e3d069beb4

$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
db171c735f86        bridge              bridge              local
ad5d0190d3ed        host                host                local
3d015af08a38        mybridge            bridge              local
4f727c15c28c        newbridge           bridge              local
6b449bfb4118        none                null                local
```

$ ip a show : 사용자 정의 브리지 생성후 ip address 확인

```
$ brctl show :  사용자 정의 브리지 확인( *. bridge-utils 가 설치되어 있어야 한다)
bridge name             bridge id           STP enabled     interfaces
br-3d015af08a38         8000.02423d4ea6bb       no          veth1c54519
br-4f727c15c28c         8000.0242aba52808       no
docker0                 8000.0242da27e8af       no          veth5c0de4c
                                                            veth6f73827
                                                            vethf8de259
```

```
$ docker network inspect newbridge
[
    {
        "Name": "newbridge",
        "Id": "4f727c15c28ce155c17702c6db357dad2fbba169c4c7502005a589e3d069beb4",
        "Created": "2020-11-11T02:53:10.532181127+09:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.30.0.0/16",
                    "IPRange": "172.30.0.0/24",
                    "Gateway": "172.30.0.1"
                }

            ]
```

```
        },

        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },

        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]

$ docker run -d -it --network newbridge --name love_apache2 httpd:2.4
ef3942202f48ca9a4e812b9e995ef90758d3e3aa807b47a3b108e7667b34bae7


# docker ps
CONTAINER ID        IMAGE               COMMAND              CREATED              STATUS
      PORTS               NAMES
ef3942202f48        httpd:2.4           "httpd-foreground"   5 seconds ago        Up 4 seconds
80/tcp              love_apache2
2ed271a33096        httpd:2.4           "httpd-foreground"   16 minutes ago       Up 16 minutes
80/tcp              mybridge_container2

# docker network inspect newbridge
[

    {

        "Name": "newbridge",
        "Id": "4f727c15c28ce155c17702c6db357dad2fbba169c4c7502005a589e3d069beb4",
        "Created": "2020-11-11T02:53:10.532181127+09:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
```

```
            "Config": [
                {
                    "Subnet": "172.30.0.0/16",
                    "IPRange": "172.30.0.0/24",
                    "Gateway": "172.30.0.1"
                }

            ]

        },

        "Internal": false,
        "Attachable": false,
        "Ingress": false,

        "ConfigFrom": {
            "Network": ""
        },

        "ConfigOnly": false,
        "Containers": {
            "ef3942202f48ca9a4e812b9e995ef90758d3e3aa807b47a3b108e7667b34bae7": {
                "Name": "love_apache2",
                "EndpointID": "c3ff3842aebf302f5b189848643034c21b579f363996daad5360c4d06cacc077",
                "MacAddress": "02:42:ac:1e:00:02",
                "IPv4Address": "172.30.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
```


---

host network

```
# docker network create --driver host myhost
Error response from daemon: only one instance of "host" network is allowed
```

<-- host network 른 한개만 허용된다.

#
docker network inspect host


```
[
        "Name": "host",

        "Id": "ad5d0190d3ed714804a980eba4f9d60dd481da4b04744ccccc43513db9399f33",
        "Created": "2020-11-10T23:33:45.172157421+09:00",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "54715eb8370e5399a27bdd5ed73a50616ef61fefe344771a56fc2a2d8b8cea69": {
                "Name": "host_network",
                "EndpointID": "1bed7ad398577653c389fad2b0eed44eab0adab796db537f32f71fca2db4d676",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```


=> contaner 의 네트워크 환경과 hostname , 그리고 /etc/hosts 파일이 host 머신설정과 같다.

none network

- 네트워크를 사용하지 않으며 nic 도 없다. lo 만 있음

$ docker network inspect none

```
[
    {
        "Name": "none",
        "Id": "6b449bfb41189d5d25bf763a5d0602bfa2db57381fea1b1e7e7839e25e808437",
        "Created": "2020-11-10T23:33:45.12051045+09:00",
        "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": {}
    }
]
$
$ docker run -it --name none_network2 --net=none centos:latest
[root@4d65c62aff3a /]$ ip a show
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
[root@4d65c62aff3a /]$
```


network driver 타입이 none 이면 컨테이너에 nic 가 생성이 되지 않으므로 컨테이너내에서 네트워크를 사용할 수 없다.

도커이미지 만들기

1. 컨테이너로부터 도커 이미지 만들기
docker container commit [옵션] 컨테이너 식별자 [이미지[:태그명]]

ex)
docker container comit [-a "hong kildong<kildong@mail.com>] " test_container [kildong/]test_image[:1.0]

생성된 도커이미지의 메타 데이터는 docker image inspect 로 확인할수 있다

컨테이너 실행후 변경된 부분은 docker container diff 명령으로 확인할수 있다.
docker container diff container이름

2. 컨테이너를 tar 파일로 출력

$ docker container export httpd > httpd.tar
tar -tf httpd.tar 명령어로 확인

tar 파일로 부터 docker image 생성

$ cat httpd.tar | docker image import - [kildong/]test_image[:1.0]

3. docker image save
- docker image 를 tar 파일로 저장
docker image save [옵션] 저장파일명 [도커이미지명]

ex)
$ docker image save -o nginx.tar nginx

저장된 docker 의 tar 파일로 부터docker image 읽어들이기

docker image load [옵션]
ex)
$ docker image load -i nginx.tar

- Dockerfile 주요 명령어 -

https://docs.docker.com/engine/reference/builder/

RUN 리눅스 명령어 ; docker image 가 만들어질 때 생성된다.
RUN apt-get install -y nginx
RUN ["/bin/bash","-c","apt-get install -y nginx"] <= json array 형식

RUN apt-get install -y nginx <= - exec 형식 /bin/sh -c 'apt-get install -y nginx' 로 실행된다.
CMD 명령어는 Dockerfile 에서 한번만 사용가능, 컨테이너가 실행될 때 실행된다.
CMD ["nginx","-g","daemon off;"]
CMD nginx -g 'daemon off;'

ENTRYPOINT
- docker container run 명령을 실행했을때 실행된다.
- Dockerfile에서 한번만 사용가능

ENTRYPOINT ["nginx","-g","daemon off;"]
ENTRYPOINT nginx -g 'daemon off;'

CMD 를 ENTRYPOINT 와 같이 사용하는경우 CMD 명령은 ENTRYPOINT 명령의
디폴트 인수로 사용된다.
ENTRYPOINT ["top"]
CMD ["-d","5"]

ONBUILD
- ONBUILD 를 사용하여 만들어진 도커이미지를 베이스 이미지로 사용해서 빌드하면 실행된다.

Dockerfile.base
```
FROM ubuntu:18.04
RUN apt-get -y update
RUN apt-get -y install nginx

EXPOSE 80

ONBUILD ADD website.tar /var/www/html
CMD ["nginx", "-g", "daemon off;"]
```

이 파일을 빌드할때 ADD website.tar /var/www/html 명령을 건너뛰고 실행한다.

$ docker run -t nginx_base:latest -f Dockerfile.base .
이 명령의 빌드 결과로 생성된 아래처럼 도커이미지를 사용해서 빌드하면
cat Dockerfile
FROM nginx_base:latest

$ docker build --tag mynginx .
이때 website.tar /var/www/html 명령이 실행된다.

ADD

ADD test.txt /tmp
ADD d1/ ./test

ADD 는 tar 아카이브를 도커이미지 내부로 복사하며 tar 아카이브가 풀려서 저장된다.
COPY 는 tar 아카이브를 풀지 않고 그대로 도커이미지 내부로 복사.

WORKDIR
- 현재의 작업디렉토리 지정
WORKDIR /tmp

USER 사용자명/UID
USER kildong ; <= kildong 계정이 미리 생성되어 있어야 한다.
RUN ["whoami"] <= kildong 문자이 출력됨

LABEL - 도커이미지에 버전정보, 작성자정보, 코멘트 등과 같은 정보를 제공한다.
LABEL maintainer "lee<lee@naver.com>"
LABEL title="webserver"
LABEL description="This image is for test"

EXPOSE 포트번호
- container port 번호를 알려주는 역할

ARG
- Dockerfile 안에서 변수 설정
- ENV 와 다르게 ARG 로 설정한 변수는 Dockerfile 안에서만 사용가능

AGR myname="kildong"
RUN echo $myname

SHELL ["shell의 경로","파라미터"]
SHELL ["/bin/csh","-C"]
RUN echo hello

COPY
- 호스트의 파일을 도커이미지 내부로 복사
COPY 호스트파일경로 도커이미지내부의 파일경로

VOLUME
- 도커이미지에 볼륨할당
VOLUME ["마운터포인터]
- 호스트나 그외 다른컨테이너로부터 볼륨 외부를 마운트

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

* hello 바이너리 파일은 정적라이브러리를 이용해서 컴파일된 파일
```
$ gcc -static -o hello hello.c
$ ldd hello
  not a dynamic executable
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.32, BuildID[sha1]=5c81a74883cffb5a16d1ed99d1d593f12f49d100, not stripped
```

* static 옵션으로 컴파일 하기 위해서는 glibc-static 패키지가 설치되어 있어야 한다.
```
$ sudo yum -y install glibc-static
```

동적라이브러 방식으로 컴파일된 hello

```
FROM scratch
COPY src/lib64 /lib64
COPY hello /
CMD ["/hello"]
```

* sample file
```
#include <stdio.h>
int main()
{
        print("Hello Docker Container \n");
        return 0;
}
```

* hello 바이너리 파일은 동적 라이브러리를 이용해서 컴파일된 파일
```
$ gcc -o hello hello.c
$ ldd hello
```

```
        linux-vdso.so.1 =>  (0x00007ffd6ba5f000) -> 가상의 공유 라이브러리, 실제 파일이 아님.
        libc.so.6 => /lib64/libc.so.6 (0x00007fde829d1000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fde82d9f000)
```

```
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=f3e22bbe81f48289a03ac5a9f3ace9ea00591306, not stripped
```

multi-stage build
https://docs.docker.com/develop/develop-images/multistage-build/

Dockerfile sample1.

```
FROM ubuntu:18.04
RUN apt-get update
RUN apt-get install -y gcc
COPY src/hello.c /tmp
WORKDIR /tmp
RUN gcc -o hello-world hello.c
CMD ["/tmp/hello-world"]
```

Dockerfile sample2.

```
FROM ubuntu:18.04 AS build-image
RUN apt-get update
RUN apt-get install -y gcc
COPY src/hello.c /tmp
WORKDIR /tmp
RUN gcc -o hello-world hello.c

FROM ubuntu:18.04
COPY --from=build-image /tmp/hello-world .
CMD ["./hello-world"]
```

public docker registry(hub.docker.com) login

```
$ 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: kildong
Password:
WARNING! Your password will be stored unencrypted in /home/vagrant/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

```
$ cat .docker/config.json
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "dHdvc2V2zw4xNDz4OmZpbmVGaW5lOTk="
                }
        }
}

$ docker logout  <=  logout 하면 자격증명 정보가 삭제된다.
Removing login credentials for https://index.docker.io/v1/
[vagrant@serverx ~]$ cat .docker/config.json
{
        "auths": {}
}
```

private docker registry.

1. registry server 에 docker 설치

```
$ sudo hostnamectl set-hostname registry.example.com
```

docker 가 설치안되어 있으면 먼저 docker 부터 설치
우분투인 경우 => https://docs.docker.com/engine/install/ubuntu/
centos 인 경우 => https://docs.docker.com/engine/install/centos/

2. root 권한으로 변경

$ sudo -s 또는 sudo -i

3. docker service start

#systemcl start docker

4. docker registry 이미지 다운로드

-> docker pull registry:latest

5. registry container 실행

-> docker run -d --name registry -p 5000:5000 --restart=always registry

* --restart=always 는 컨테이너가 중단되면 다시 실행하기 위한 옵션
--restart=always로 실행되고 있는 컨테이너를 중단시키기 위해서는
sudo docker update --restart=no 컨테이너 이름 그다음 중단시키면 된다.

---

docker registry 에 docker image push

docker 가 설치된다른장비에서 테스트.
tag 설정형식
docker tag 이미지이름:tag docker registry url/이미지이름:tag

* docker registry 에 엑세스 할때 기본적으로 보안연결(https)을 사용하게 된다.
인증서설치 없이 접속하기위해서는 아래처럼 비보안연결 설정 파일을 클라이언트에서 만들면 된다.

비보안(insecure) 레지스트리 추가
-> **vi /etc/docker/daemon.json**
{"insecure-registries": ["registry.example.com:5000"] }
설정파일후 docker daemon restart 를 해야 한다. => sudo systemctl restart docker
systemctl restart docker

- docker image upload
docker tag hello-world registry.example.com:5000/hello-world
docker images
docker push registry.example.com:5000/hello-world

확인 :
curl -X GET http://registry.example.com:5000/v2/_catalog

* tag version 정보까지 확인하는 방법은
ex)
curl -X GET http://registry.example.com:5000/v2/lee/myhttpd/tags/list
{"name":"lee/myhttpd","tags":["1.0"]}

yaml syntax

- 사람이 쉽게 읽을수 있다는 컨셉으로 개발된 데이터 양식
- XML 이나 JSON 보다 읽기 쉽다
- 계층구조로 되어 있다. 부모 자식간의 관계

yaml 문서의 시작은 '---' 로 시작한다, 생략할수 있다.
yaml 문서의 끝은 '...' 으로 끝난다. 보통 생략한다.
- 주석은'#' 으로 시작한다.
- 기본자료형은 리스트 및 딕셔너리(해시)로 표현
- 리스트형은 하이픈(-) 으로 시작, 하이픈대신 대괄호를 사용해도 된다.
- 들여쓰기를 일정한 간격으로 해야 한다. 들여쓰기 간격이 맞지 않으면 에러가 발생한다.

- 들여쓰기는 보통 2칸을 많이 사용한다.
- 딕셔너리는 key: value 로 표현한다. 콜론뒤에는 보통 1칸을 띄운다.
- yaml 파일은 확장자를 yaml 또는 yml 사용한다.
- yaml 파일에서 하위레벨(종속관계)은 들여쓰기로 구분된다

---
yaml 파일 sample

```
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```
=> ["Apple","Orange","StrawBerry","Mango"] 로 표현가능

```
# An employee record
martin:
    name: Martin D'vloper
    job: Developer
    skill: Elite
```
=> {"martin": {"name": "Martin D'vloper","job": "Developer","skill": "Eilte"}}

```
# Employee records
-  martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - java
-  cane:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - php
      - c
```
=> {"martin": {"name": "Martin D'vloper","job": "Developer","skills": ["python","perl","java"]},"cane": {"name": "Tabitha Bitumen","job": "Developer","skills": ["php","c"]}}

docker compose

docker-compose.yaml sample1

```
version: '3.8'    # <= version은 더 이상 사용되지 않는다.
services:
  myapache:  # <= container 이름(실제 이름은 directory명-myapache-1 이된다)
    image: httpd:2.4 #
    ports:
    - "9000:80"     # port forwarding 설정
  mydb: # <= container 이름(실제 이름은 directory명-mydb-1 이 된다)
    image: mysql:5.5
    ports:
    - "9001:3306"  # mysql database 의 디폴트번호는 3306 이다. portforwarding 설정
    environment:    # 환경변수 설정을 위한 key
    - MYSQL_ROOT_PASSWORD=mypass
    - MYSQL_USER=myuser  # 일반사용자 myuser 가 생성된다.
    - MYSQL_PASSWORD=userpass # myuser 의 암호
    - MYSQL_DATABASE=mydb # mydb 를 선택
    volumes:
    - /var/dbdata:/var/lib/mysql  # 데이터베이스 영구볼륨설정
```

* 명령어를 사용하는 경우에는 아래처럼 하면 된다.

```
$ docker run -d --name=mydb -p 9001:3306 --env MYSQL_ROOT_PASSWORD=mypass --env MYSQL_USER=myuser --env MYSQL_PASSWORD=userpass --env MYSQL_DATABASE=mydb -v /var/dbdata:/var/lib/mysql mysql:5.5

$ docker run -d --name myapache -p 9000:80 --link mydb:mysql5 httpd:2.4

$ docker exec -it myapache cat /etc/hosts  (link 옵션을 사용하면 아래처럼 hosts 파일에 등록된다)
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      mysql5 7635bc1c4bf7 mydb
172.17.0.3      7bcde0ae296b
```

docker compose up -f 파일명 (foreground 로 실행)
docker compose up -f 파일명 (background 로 실행) -d
ex)

```
[vagrant@server1 new]$ docker compose  -f docker-compose.yaml up -d
[+] Running 2/2
 ⠿ Container new-mydb-1     Healthy
 ⠿ Container new-myapche-1  Started
[vagrant@server1 new]$
```

* -f 옵션과 파일명을 생략하면 docker-comose.yaml(또는 docker-compose.yml) 을 파일을 참조한다.

docker-compose.yaml 예제.

```
networks:
  webapps:
    driver: bridge
    ipam:
      config:
      - subnet: 172.30.0.0/16

services:
  myapache:
    image: httpd:2.4
    ports:
    - "9000:80"
    networks:
      webapps:
        ipv4_address: 172.30.0.10

#   links:  (deprecated)
#     - mydb:db

    depends_on:
    - mydb
  mydb:
    image: mysql:5.7
    ports:
    - "9001:3306"
    networks:
      webapps:
        ipv4_address: 172.30.0.11
    environment:
    - MYSQL_ROOT_PASSWORD=mypass
    - MYSQL_USER=user1
    - MYSQL_PASSWORD=userpass
```

```
    - MYSQL_DATABASE=userdb
    volumes:
    - /var/dbfiles:/var/lib/mysql
```

* depends on:
  - mydb

=> 아파치 컨테이너 실행하기전에 myapache container 실행하기전에 mydb 컨테이너를 먼저 실행한다.
그러나 healthcheck 를 하지 않으므로 mydb 컨테이너가 완료될때까지 기다리지는 않기때문에 mydb 실행이 완료되기전에 myapache container 가 먼저 실행완료될수 있다.
healthcheck 설정을 하면 디펜던시 설정되어 있는 컨테이너가 unhealthy 상태이면 그다음 container 가 실행이 되지 않는다.

*. healthchek 를 위해서는 아래와 같은 설정이 필요하다

```
depends_on:
    mydb:
        condition: "service_healthy"
```

그리고 맨아래에 다음부분을 추가

```
healthcheck:
  test: ["CMD","mysqladmin","ping","-u","root","-pmypass","-h","localhost"]
  interval: 5s  : 5초간격으로 mysqladmin 명령어 실행
  timeout: 5s  : 명령어 실행후 응답을 5초동안 기다린다.
  retries: 2 : unhealthy로 간주되기 위한 연속 실패횟수.
  start_period 5s : 컨테이너 실행후 첫 헬스체크를 실행하는 시간간격
```

* mysqladmin 명령어로 mysql server 가 접속가능한 상태인지를 아래처럼 확인할수 있다.

```
[vagrant@server1 new]$ docker exec -it new-mydb-1 /bin/bash
bash-4.2# mysqladmin ping -u root -pmypass -h localhost
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
mysqld is alive
bash-4.2#
```

*. docker compose 명령어로 container 삭제는 아래처럼

```
[vagrant@server1 new]$ docker compose stop
[+] Running 2/2
 ⠿ Container new-myapche-1  Stopped                                          1.3s
 ⠿ Container new-mydb-1     Stopped                                          2.0s
[vagrant@server1 new]$ docker compose rm
? Going to remove new-myapche-1, new-mydb-1 Yes
[+] Running 2/0
 ⠿ Container new-mydb-1     Removed                                          0.0s
 ⠿ Container new-myapche-1  Removed                                          0.0s
[vagrant@server1 new]$
```

docker-compose.yaml 예제

```
networks:
  webapps:
    driver: bridge
    ipam:  # ip address management
      config:
      - subnet: 172.30.0.0/16

services:
  myapche:
    image: httpd:2.4
    ports:
    - "9000:80"
    networks:
      webapps:
        ipv4_address: 172.30.0.10

#   links:  (deprecated)
#     - mydb:db

     depends_on:
       mydb:
         condition: "service_healthy"
  mydb:
    image: mysql:5.7
    ports:
    - "9001:3306"
    networks:
      webapps:
        ipv4_address: 172.30.0.11
    environment:
    - MYSQL_ROOT_PASSWORD=mypass
    - MYSQL_USER=user1
    - MYSQL_PASSWORD=userpass
    - MYSQL_DATABASE=userdb
    volumes:
    - /var/dbfiles:/var/lib/mysql
    healthcheck:
      test: ["CMD","mysqladmin","ping","-u","root","-pmypass","-h","localhost"]
      interval: 5s
      timeout: 5s
      retries: 2
      start_period: 5s
```

- docker compose를 활용해서 web application 배포하기 실습