목차

1장 도커란 ?

- 가상머신 과 도커 컨테이너
- 도커엔진 설치
- 도커 데몬

도커(Docker)

- 도커
  - 리눅스 컨테이너에 여러기능을 추가
  - 애플리케이션을 컨테이너로 좀더 쉽게 사용할 수 있게 만든 오픈소스 프로젝트
  - 2013년 3월 dotCloud 창업자 Solomon Hykes 가 Pycon Conference 에서 발표
  - Go 언어로 작성 된 “The future of linux Containers”
  - 가상 머신과 달리 성능손실이 거의 없는 차세대 클라우드 솔루션으로 주목
- 도커 프로젝트
  - 도커 컴포즈, 도커머신, 레지스트리, Kitematic 등 다양한 프로젝트 존재
  - 일반적 도커 : 도커엔진을 의미
  - 도커 프로젝트는 도커 엔진을 효율적으로 사용하기 위한 도구들

가상머신 과 도커 컨테이너

- 가상머신
  - 하이퍼바이저를 통한 가상화로 성능 손실이 발생
  - 완벽한 독립적 공간을 생성 하나, 이미지 용량이 크고 가상머신 배포에 부담
- 도커 컨테이너
  - 리눅스 Chroot, 네임스페이스, Cgroup 를 사용한 프로세스 단위 격리 환경 구성
  - 애플리케이션 구동을 위한 라이브러리만 포함한 이미지생성, 용량이 작음

윈도우 도커엔진 설치

- 도커 툴박스(Docker Toolbox)
  - 설치 환경 : 윈도우 7 64비트 이상
  - 오라클 버추얼박스(VirtualBox) 의 가상화 기술을 이용해 리눅스 가상환경에 도커 엔진을 구성
  - https://www.docker.com/products/docker-toolbox

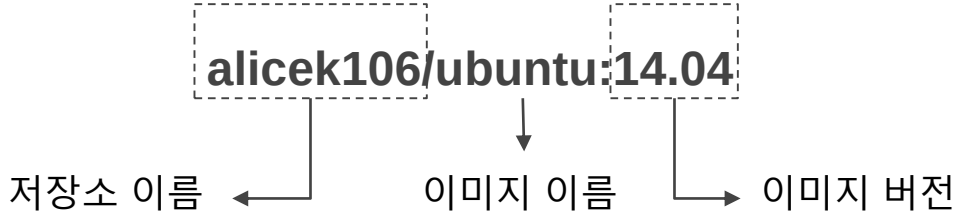
- Docker for Windows
  - 설치 환경 : 윈도우 10 64비트 이상
  - Windows Hyper-V 를 이용해 가상화 환경 제공
  - https://docs.docker.com/docker-for-windows/install/

2장 도커 엔진

- 도커이미지와 컨테이너
- 도커 컨테이너 다루기
- 도커 볼륨
- 도커 네트워크
- 도커 이미지
- 도커 파일(Dockerfile)

도커 이미지와 컨테이너

- 도커이미지
  - 가상머신 생성시 사용하는 ISO 와 비슷한 개념의 이미지
  - 여러 개의 층으로 된 바이너리 파일로 존재
  - 컨테이너 생성시 읽기 전용으로 사용됨
  - 도커 명령어로 레지스트리로 부터 다운로드 가능



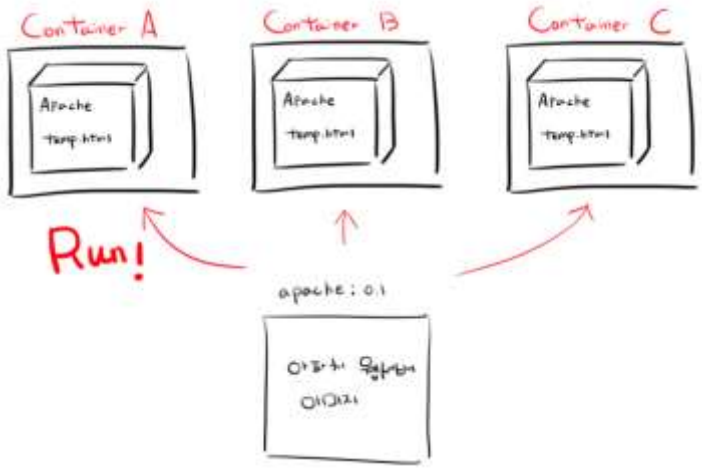
  - 저장소 이름 : 이미지가 저장된 장소, 이름이 없으면 도커 허브(Docker Hub)로 인식
  - 이미지 이름 : 이미지의 역활을 나타낼 이름, 생략 불가능
  - 이미지 버전 : 이미지 버전정보, 생략하면 latest 로 인식

도커 이미지와 컨테이너

- 도커컨테이너
  - 도커 이미지로 부터 생성됨
  - 격리된 파일시스템, 시스템 자원, 네트워크를 사용할 수 있는 독립공간 생성
  - 도커 이미지 목적에 맞게 컨테이너를 생성하는 것이 일반적

    예) 웹 서버 도커 이미지로 부터 여러개의 컨테이너 생성 = 개수만큼의 웹서버
  - 이미지를 읽기 전용으로 사용, 이미지 변경 데이터는 컨테이너 계층에 저장
  - 컨테이너의 애플리케이션 설치/삭제는 다른 컨테이너에 영향이 없음

    예) 우분투 이미지로 별도의 컨테이너 생성 후 Apache, Mysql 설치/삭제 가능



도커 컨테이너 다루기

- 도커 엔진 버전 확인

```
# docker -v
Docker version 17.09.0-ce, build afdb6d4
```

- 컨테이너 생성

```
# docker run -i -t ubuntu:14.04
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
bae382666908: Pull complete
...
b0de1abb17d6: Pull complete
Digest:
sha256:6e3e3f3c5c36a91ba17ea002f63e5607ed6a8c8e5fbbddb31ad3e15638b51ebc
Status: Downloaded newer image for ubuntu:14.04
root@de98b3c4d0e8:/#
```

  - run : 컨테이너 실행
  - -i : 테이너와 상호 입출력 가능 옵션
  - -t : 셀을 사용할 수 있는 tty 활성화
  - de98b3c4d0e8 : 컨테이너 고유 ID

도커 컨테이너 다루기

- 컨테이너 파일 시스템 확인

```
root@de98b3c4d0e8:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
```

- 컨테이너 내부에서 빠져나오기

```
root@de98b3c4d0e8:/# exit
exit
```

  - exit 또는 Ctrl+D : 컨테이너 빠져나오면서, 컨테이너 정지
  - Ctrl+P,Q : 컨테이너 정지 하지 않고 빠져 나오기

도커 컨테이너 다루기

- Centos7 이미지 내려받기

```
# docker pull centos:7
7: Pulling from library/centos
d9aaf4d82f24: Pull complete
Digest:
sha256:4565fe2dd7f4770e825d4bd9c761a81b26e49cc9e3c9631c58cfc3188be9505a
Status: Downloaded newer image for centos:7
```

- 이미지 목록 확인

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              7                   d123f4e55e12        6 hours ago         197MB
ubuntu              14.04               dea1945146b9        7 weeks ago         188MB
```

- 컨테이너 생성 하기

```
# docker create -i -t --name mycentos centos:7
250c54187b22d9f177435099cd8613581f24429b07809c71fc4f96e16a982d7d
```

  - create : 컨테이너 생성 (생성만 되고 실행은 되지 않음)
  - --name : 컨테이너 이름 지정 옵션

도커 컨테이너 다루기

- 컨테이너 시작 및 들어가기

```
# docker start mycentos
mycentos

[root@docker1 ~]# docker attach mycentos
[root@250c54187b22 /]#
```

  - start : 컨테이너 시작
  - attach : 컨테이너 들어가기

- 컨테이너 ID 사용

```
# docker start 250c54
250c54

# docker attach 250c54
[root@250c54187b22 /]#
```

  - 고유 ID 이름을 사용하여 컨테이너 관리 가능
  - ID 값중 구분이 가능한 길이만 입력 후 컨테이너 명령 실행

도커 컨테이너 다루기

- 컨테이너 목록 확인

```
# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
250c54187b22        centos:7            "/bin/bash"         25 minutes ago      Up 3 minutes                            mycentos

# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
250c54187b22        centos:7            "/bin/bash"         28 minutes ago      Up 6 minutes                                    mycentos
de98b3c4d0e8        ubuntu:14.04        "/bin/bash"         41 minutes ago      Exited (0) 36 minutes ago                       festive_kare
```

  - ps : 현재 실행중인 목록 출력
  - ps -a : 모든 컨테이너 목록 출력
    - IMAGE : 컨테이너 생성시 사용된 이미지 이름
    - COMMAND : 컨테이너 시작시 실행될 명령어 (기본 내장된 명령어 /bin/bash)
    - CREATED : 컨테이너가 생성된 이후 시간
    - STATUS : 컨테이너 상태 (UP : 실행중, Exited : 중지됨, Pause : 일시중지)
    - PORTS : 컨테이너가 오픈한 포트와 호스트에 연결 상태
    - NAMES : 컨테이너 고유 이름, 중복 불가능, 변경가능

- 컨테이너 이름변경

```
# docker rename mycentos yourcentos
```

도커 컨테이너 다루기

- 컨테이너 삭제

```
# docker ps -a
CONTAINER ID        IMAGE             COMMAND             CREATED             STATUS                      PORTS               NAMES
250c54187b22        centos:7          "/bin/bash"         28 minutes ago      Up 6 minutes                                    mycentos
de98b3c4d0e8        ubuntu:14.04      "/bin/bash"         41 minutes ago      Exited (0) 36 minutes ago                       festive_kare

# docker rm festive_kare
festive_kare

# docker ps -a
CONTAINER ID        IMAGE             COMMAND             CREATED             STATUS              PORTS               NAMES
250c54187b22        centos:7          "/bin/bash"         41 minutes ago      Up 19 minutes                           yourcentos

# docker rm yourcentos
Error response from daemon: You cannot remove a running container
250c54187b22d9f177435099cd8613581f24429b07809c71fc4f96e16a982d7d. Stop the container before attempting removal or force
remove

# docker stop yourcentos
yourcentos


# docker rm yourcentos
```

  - rm : 컨테이너 삭제 (중지된 컨테이너만 삭제됨)
  - rm -f : 실행중인 컨테이너 삭제
  - stop : 컨테이너 중지

도커 컨테이너 다루기

- 중지된 모든 컨테이너 삭제

```
# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
250c54187b22d9f177435099cd8613581f24429b07809c71fc4f96e16a982d7d

Total reclaimed space: 0B
```

- 모든 컨테이너 정지 및 삭제

```
# docker ps -a -q
56e89dd10229
5d0ef0ce7510
dc973f626abd
```

  - ps -a : 상태 관계없이 모든 컨테이너 출력
  - ps -q : 컨테이너 ID만 출력

```
# docker stop $(docker ps -a -q)

# docker rm $(docker ps -a -q)
```

  - $(docker ps -a -q) : 컨테이너 ID값을 실행 명령의 입력값으로 전달

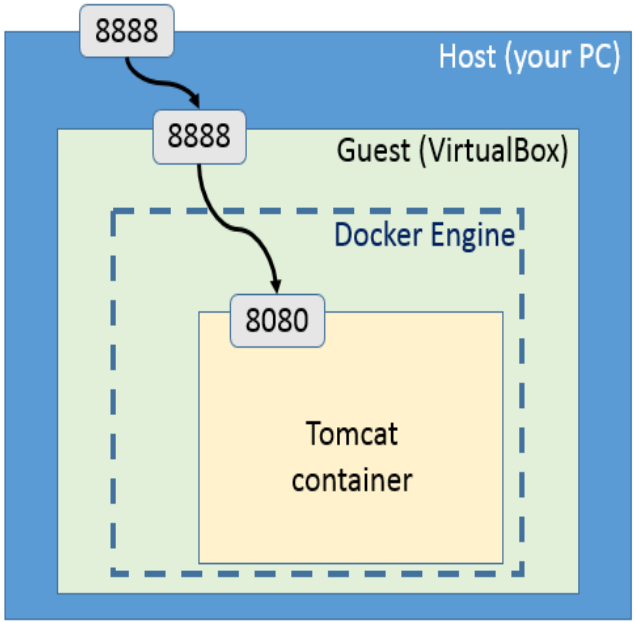
도커 컨테이너 다루기

- 컨테이너 네트워크 상태 확인

```
#docker run -i -t --name network_test ubuntu:14.04

root@f0db180e6ca4:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:508 (508.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```



- 컨테이너 생성시 외부 포트 연결

```
# docker run -i -t --name myserver -p 80:80 ubuntu:14.04

# docker run -i -t --name webserver -p 3306:3306 -p 192.168.35.51:8888:8080 ubuntu:14.04
```

  - -p : 호스트 포트 와 컨테이너 포트 연결 옵션

    특정 호스트 IP:포트 와 컨테이터 포트 연결 가능

컨테이너 애플리케이션 구축

- 서비스 컨테이너 화
  - 컨테이너에 하나의 애플리케이션만 실행
  - 컨테이너간 독립성 보장으로 버전관리 및 소스모듈화 등이 쉬움
  - 한 컨테이너에 프로세스 하나만 실행 = 도커 철학

- 워드프레스 블로그 서비스 구축
  - 데이타베이스 와 워드프레스 웹서버 컨테이너 생성 및 연동

  - Mysql 컨테이너 생성 및 실행

```
# docker run -d \
> --name wordpressdb \
> -e MYSQL_ROOT_PASSWORD=password \
> -e MYSQL_DATABASE=wordpress \
> mysql:5.7
```

  - -d : Detached 모드, 백그라운드 에서 동작하는 애플리케이션으로 실행
  - -e : 내부 환경 변수 설정

컨테이너 애플리케이션 구축

- 워드프레스 블로그 서비스 구축
  - Wordpress 웹 컨테이너 생성

```
# docker run -d \
> -e WORDPRESS_DB_PASSWORD=password \
> --name wordpress \
> --link wordpressdb:mysql \
> -p 80 \
> wordpress
```

  - --link : 컨테이너간 접근시 별명으로 접근 가능하도록 설정

    wordpressdb 컨테이너를 mysql 별명으로 접근가능

    주의사항 : --link 에 입력된 컨테이너가 중지 또는 존재하지 않으면 실행 불가능

  - -p 80 : 호스트의 특정포트와 컨테이너의 80포트를 연결

```
# docker exec wordpress /usr/bin/apt-get update
# docker exec wordpress /usr/bin/apt-get install iputils-ping -y
# docker exec wordpress ping -c 2 mysql
PING mysql (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: icmp_seq=0 ttl=64 time=0.076 ms
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.062 ms
--- mysql ping statistics ---
2 packets transmitted, 2 packets received, 0% packet lo
```

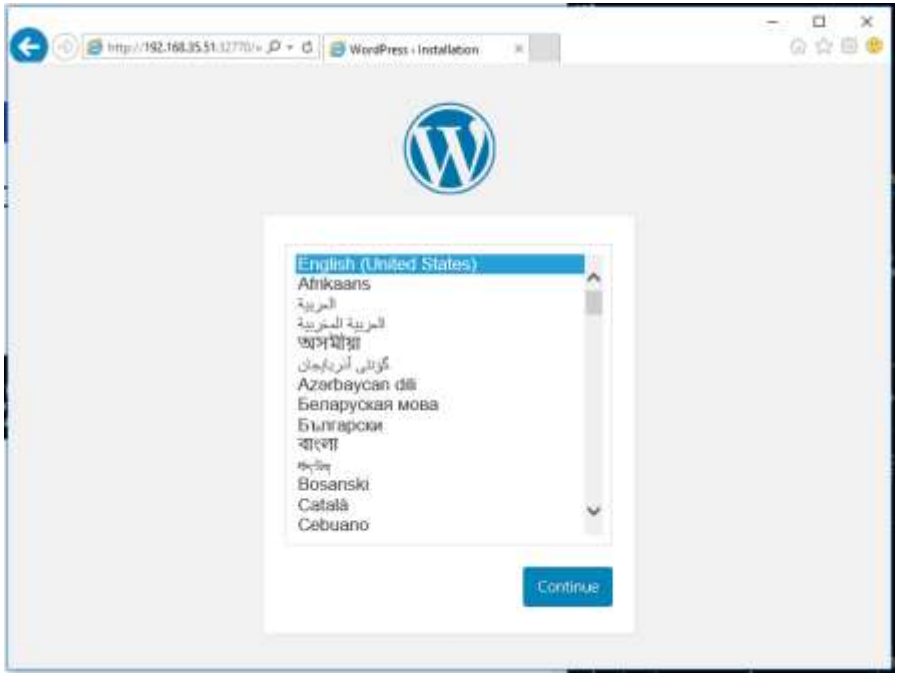
  - exec : 컨테이너 내부에서 명령어를 실행한 뒤 그 결과값을 반환

컨테이너 애플리케이션 구축

- 워드프레스 블로그 서비스 구축
  - 컨테이너 확인

```
# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
8ba76563b8e1        wordpress           "docker-entrypoint..."   4 minutes ago       Up 4 minutes        0.0.0.0:32770->80/tcp  wordpress
5cfe97da35cb        mysql:5.7           "docker-entrypoint..."   6 minutes ago       Up 6 minutes        3306/tcp               wordpressdb
```

  - wordpress 웹서버 80포트는 호스트의 32770 포트로 연결됨



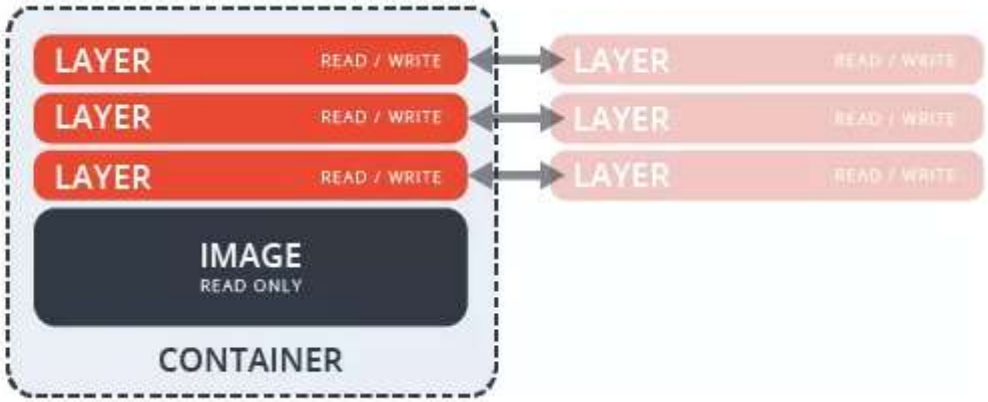
도커 볼륨

- 컨테이너 레이어
  - 이미지로 컨테이너를 생성하면 이미지는 읽기전용
  - 컨테이너 변경정보는 변경사항만 별도로 저장, 컨테이너 정보 보존

    예) mysql 의 DB 저장 데이터는 컨테이너 레이어 공간에 저장됨
  - 컨테이너 레이어의 데이터는 컨테이너 삭제와 함께 삭제되어 복구 불가능

Docker Container

is comprised of a base image with layers that can be swapped out so it's not necessary to replace the entire VM when updating an application



- 볼륨
  - 컨테이너 데이터를 영구적으로 보관 가능
  - 호스트볼륨을 공유 또는 볼륨 컨테이너 이용가능

도커 볼륨

- 호스트 공유 볼륨 공유
  - 호스트 공유 볼륨을 사용한 Mysql 컨테이너 생성

```
# docker run -d \
> --name wordpressdb_hostvolume \
> -e MYSQL_ROOT_PASSWORD=password \
> -e MYSQL_DATABASE=wordpress \
> -v /home/wordpress_db:/var/lib/mysql \
> mysql:5.7
```

    - -v : 호스트의 디렉토리 및 파일을 컨테이너의 디렉토리 및 파일과 공유
      리눅스 시스템 관점 : 파일 및 디렉토리를 마운트하는 구조
      호스트 /home/wordpress_db =공유= 컨테이너 /var/lib/mysql

  - 호스트 /home/wordpress_db 폴더확인

```
# ls /home/wordpress_db
auto.cnf    ca.pem          client-key.pem  ibdata1     ib_logfile1  mysql              private_key.pem  server-cert.pem  sys
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile0  ibtmp1       performance_schema  public_key.pem   server-key.pem
wordpress
```

  - wordpressdb_hostvolume 컨테이너의 mysql 마운트 정보

```
# docker exec wordpressdb_hostvolume mount | grep mysql
/dev/mapper/centos-root on /var/lib/mysql type xfs (rw,relatime,attr2,inode64,noquota)
```

도커 볼륨

- 볼륨 컨테이너 공유
  - -v 옵션으로 볼륨을 사용하는 컨테이너를 다른 컨테이너와 공유
  - 호스트 공유 볼륨을 사용한 Mysql 컨테이너 생성

```
# docker run -i -t \
> --name volumes_from_container \
> --volumes-from wordpressdb_hostvolume \
> ubuntu:14.04
```

  - --volumes-from : 다른 컨테이너의 볼륨을 공유
  - 새로 성성된 컨테이너의 공유된 폴더 확인

```
# ls /var/lib/mysql/
auto.cnf    ca.pem          client-key.pem  ib_logfile0  ibdata1  mysql              private_key.pem  server-cert.pem  sys
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1   performance_schema  public_key.pem   server-key.pem   wordpress

# mount | grep mysql
/dev/mapper/centos-root on /var/lib/mysql type xfs (rw,relatime,attr2,inode64,noquota)
```

도커 볼륨

- 도커 자체 제공 볼륨 기능
  - 도커 볼륨 생성

```
# docker volume create --name myvolume
myvolume
```

  - 새로 생성된 볼륨 확인

```
# docker volume ls
DRIVER              VOLUME NAME
local               myvolume
```

  - myvolume_1 볼륨 컨테이너 생성 후 파일 생성

```
# docker run -i -t --name myvolume_1 \
> -v myvolume:/root/ \
> ubuntu:14.04
root@0259f65f9603:/#  echo hello, volume! >> /root/volume
```

  - Myvolume_2 볼륨 컨테이너 생성 후 볼륨 파일 확인

```
# docker run -i -t --name myvolume_2 \
> -v myvolume:/root/ \
> ubuntu:14.04
root@493dae2bc70b:/# cat /root/volume
hello, volume!
```

도커 볼륨

- 볼륨 정보
  - 도커 엔진에서 볼륨은 디렉토리에 상응하는 단위
  - 볼륨은 다양한 스토리지 백엔드 플러그인 드라이버 사용가능
  - 기본적으로 제공되는 드라이버는 local

볼륨 정보 확인

```
# docker inspect --type volume myvolume
[
    {
        "CreatedAt": "2017-11-04T20:00:59+09:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvolume/_data",
        "Name": "myvolume",
        "Options": {},
        "Scope": "local"
    }
]
```

  - inspect : 컨테이너, 이미지, 볼륨 등 도커 구성 단위의 정보 확인 가능
  - -- type : 정보를 확인할 종류를 명시 (image 또는 volume)

도커 볼륨

- 컨테이너 생성시 볼륨 생성
  - volume_auto 볼륨 컨테이너 생성

```
# docker run -i -t --name volume_auto \
> -v /root \
> ubuntu:14.04
```

    - -v /root : 컨테이너 생성시 /root 폴더용 볼륨이 자동 생성됨

  - 볼륨 정보 확인

```
# docker volume ls
DRIVER              VOLUME NAME
local               c3fd49eb43304d610d2cc4528b2fef1594f9dcb52b5f04772932294f2948465a
local               myvolume
```

    - c2fd49 로 시작하는 ID 의 볼륨이 생성됨

  - 컨테이너 정보 확인

```
# docker container inspect volume_auto | grep c3fd49
            "Name": "c3fd49eb43304d610d2cc4528b2fef1594f9dcb52b5f04772932294f2948465a",
            "Source":
"/var/lib/docker/volumes/c3fd49eb43304d610d2cc4528b2fef1594f9dcb52b5f04772932294f2948465a/_data",
```

도커 볼륨

! 에러발생

- 볼륨 디렉토리 쓰기권한 설정
  - 쓰기 제한 볼륨 컨테이너 생성

```
# docker run  -i -t --name datavol1 \
> -v /root/data1:z \
> -v /root/data2:Z \
> ubuntu:14.04 \
> bash

# docker run  --name datavol2 \
> --volumes-from=datavol1 \
> -d ubuntu:14.04 \
> touch /root/data2 \
touch: cannot touch '/data2/mydata': Permission denied
```

  - -v [볼륨명]:Z : 다른 컨테이너가 볼륨을 쓰지 못하도록 쓰기권한 제어
  - datavol2 컨테이너는 /root/data1 는 쓰기 가능, /root/data2 쓰기 불가능

도커 볼륨

- 볼륨 디렉토리 쓰기권한 설정
  - RO/RW 권한 볼륨 컨테이너 생성

```
# docker run  -i -t --name datavol1 \
> -v /home/data1:/root/data1:rw \
> -v /home/data2:/root/data2:ro \
> ubuntu:14.04 \
> bash

# docker run  -i -t --name datavol2 \
> --volumes-from=datavol1 \
> ubuntu:14.04 \
> touch /root/data2/mydata \
touch: cannot touch '/root/data2/mydata': Read-only file system
```

  - -v [호스트볼륨]:[볼륨명]:rw : 볼륨 읽기 쓰기 권한 제공
  - -v [호스트볼륨]:[볼륨명]:ro : 볼륨 읽기 권한만 제공
  - datavol2 컨테이너는 /root/data1 는 쓰기 가능, /root/data2 쓰기 불가능

도커 볼륨

- 볼륨 삭제
  - 볼륨은 컨테이너를 삭제 해도 자동으로 삭제 되지 않는다.
  - 사용하지 않은 볼륨 한번에 삭제

```
# docker volume prune
WARNING! This will remove all volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
myvolume
c3fd49eb43304d610d2cc4528b2fef1594f9dcb52b5f04772932294f2948465a

Total reclaimed space: 245.1MB
```

  - 스테이트리스(stateless) 컨테이너 : 데이터를 외부 볼륨에 저장하는 방식
  - 스테이트 풀(stateful) 컨테이너 : 데이터를 컨테이너 내부에 저장 하는 방식

컨테이너 로깅

- json-file 로그 사용
  - 표준출력(StdOut) 과 에러(StdErr) 로그를 별도의 메타데이터 파일로 저장
  - 로그파일 : /var/lib/docker/containers/컨테이너ID/컨테이너ID-json.log

로그 확인

```
# docker logs mysql
Initializing database
2017-11-05T10:20:38.822037Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please
use --explicit_defaults_for_timestamp server option (see documentation for more details).
..
```

  - logs : 컨테이너 로그확인

특정 시간 이후 로그 확인 (Unix 타임스템프 사용)

```
# docker logs --since 1509877247 mysql
2017-11-05T10:20:47.502681Z 0 [Note] InnoDB: Shutdown completed; log sequence number 12169513
2017-11-05T10:20:47.503756Z 0 [Note] InnoDB: Removed temporary tablespace data file: "ibtmp1"
..
```

로그 스트림 출력

```
# docker logs -f -t mysql
..
```

컨테이너 로깅

- 로그 드라이버
  - 컨테이너 로그를 저장하는 다양한 로그 드라이버 제공
  - 대표적인 로그 드라이버 : syslog, journald, fluentd, awslogs
- syslog 로그 사용
  - 컨테이너 로그를 syslog로 보내 저장
    - syslogtest 컨테이너 생성

```
# docker run -d --name syslog_container \
> --log-driver=syslog \
> ubuntu:14.04 \
> echo syslogtest
```

    - 호스트의 message 로그에 기록된 로그 확인

```
# cat /var/log/messages | grep syslogtest
Nov  5 19:39:54 docker1 314d3db5ee77[946]: syslogtest
```

컨테이너 로깅

- 원격 syslog 서버 저장방법
  - syslog 원격 서버 설치, 로그 정보를 원격서버로 전달
  - 원격 로그 저장방법 rsyslog 사용,
    - rsyslog 서버 컨테이너 생성 및 설정

```
# docker run -i -t \
> -h rsyslog \
> --name rsyslog_server \
> -p 514:514 -p 514:514/udp \
> ubuntu:14.04

root@rsyslog:/# vi /etc/rsyslog.conf
..
# provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514
..
# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
..
```

  - -h : 컨테이너 호스트 이름 지정
    - Rsyslog 서비스 재시작

```
root@rsyslog:/# service rsyslog restart
```

컨테이너 로깅

- 원격 syslog 서버 저장방법
  - 클라이언트 컨테이너 생성 및 로그생성

```
# docker run -i -t \
> --log-driver=syslog \
> --log-opt syslog-address=tcp://192.168.35.51:514 \
> --log-opt tag="mylog" \
> ubuntu:14.04

root@599eebe7568c:/# echo test
test
```

    - --log-opt : 로그 드라이버에 추가할 옵션

      syslog-address = 로그 서버 주소

      tag = 로그 저장시 사용될 태그 정보, 로그 분류 용도

  - Rsyslog 서버 syslog 확인

```
root@rsyslog:/# cat -f /var/log/syslog
Nov  5 20:01:09 192.168.35.51 mylog[946]: #033]0;root@599eebe7568c: /#007root@599eebe7568c:/# echo
est#010 #010#010 #010#010 #010test#015
Nov  5 20:01:09 192.168.35.51 mylog[946]: test#015
```

도커 이미지

- 도커 이미지
  - 도커 이미지는 도커 허브(Docker Hub)라는 중앙 이미지 저장소에서 다운로드함
  - docker create, docker run, docker pull 명령어로 이미지 다운로드 가능
  - 도커 계정 생성후 이미지를 업로드/다운로드 가능
  - 도커 허브 비공개 저장소는 요금을 지불 해야 사용 가능
  - 이미지 저장소를 직접 구축해 사용 가능 = 도커 사설 레지스트리

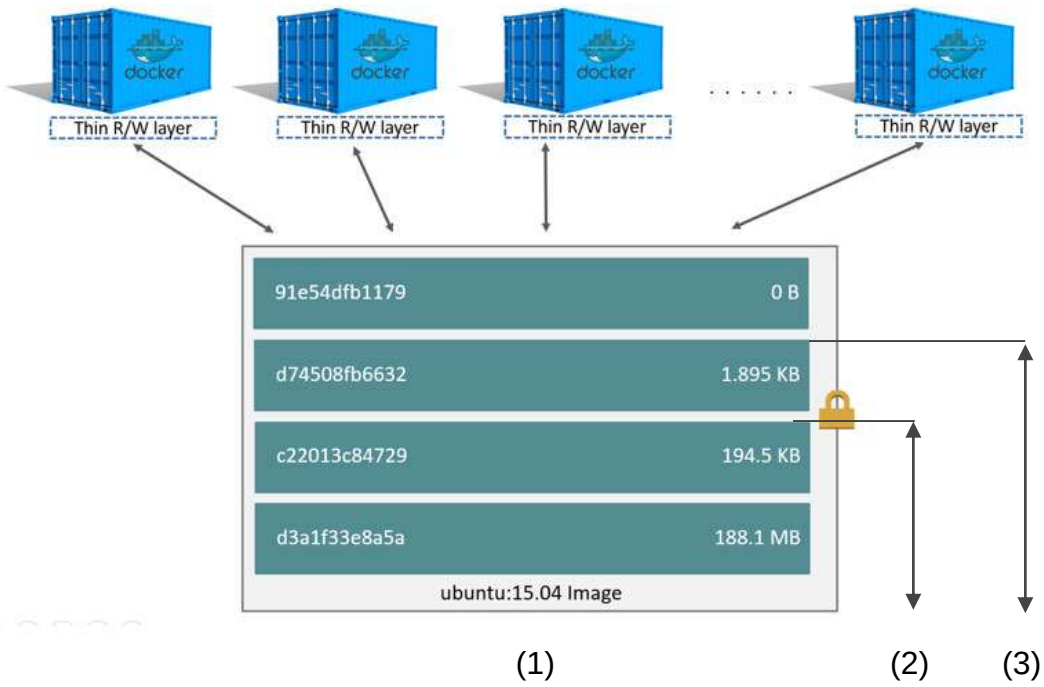
- 도커 허브 이미지 검색

```
# docker search ubuntu
NAME                              DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                            Ubuntu is a Debian-based Linux operating s...   6770      [OK]
dorowu/ubuntu-desktop-lxde-vnc    Ubuntu with openssh-server and NoVNC            141                  [OK]
rastasheep/ubuntu-sshd            Dockerized SSH service, built on top of of...   115                  [OK]
ansible/ubuntu14.04-ansible       Ubuntu 14.04 LTS with ansible                   88                   [OK]
ubuntu-upstart                    Upstart is an event-based replacement for ...   80        [OK]
```

  - STARS : 도커 사용자로 부터 즐기찾기 수

도커 이미지

. 이미지 구조 이해



(1) ubuntu:15.04 (2) commit_test:first (3) commit_test:second

도커 이미지

- 도커 이미지 생성
  - 이미지 생성을 위한 컨테이너 생성

```
# docker run -i -t --name commit_test ubuntu:14.04
root@423213a9e410:/# echo test_first! >> first
```

  - docker commit 명령을 사용, first commit 이미지 생성

```
# docker commit \
> -a "user1" -m "my first commit" \
> commit_test \
> commit_test:first
sha256:175f54ed8eb03cbd3eb52dcf0fd9af84b099abfe00f85007c65424b7bbf513d4
```

    - Docker commit [option] CONTAINER [REPOSITORY[:TAG]]
    - -a : author (이미지 작성자) 메타데이터를 이미지에 저장
    - -m : 커밋 메세지를 뜻, 이미지 설명 입력

  - 이미지 생성 확인

```
[root@docker1 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
commit_test         first               175f54ed8eb0        3 minutes ago       188MB
ubuntu              14.04               dea1945146b9        7 weeks ago         188MB
```

도커 이미지

- 도커 이미지 생성
  - second 이미지 생성을 위한 컨테이너 생성

```
# docker run -i -t --name commit_test2 commit_test:first
root@77edfe2e5e69:/# echo test_second! >> second
```

  - docker commit 명령을 사용, second commit 이미지 생성

```
# docker commit \
> -a "user1" -m "my second commit" \
> commit_test2 \
> commit_test:second
sha256:c87fc1137ca81f04246608adc68efb47cfd0c5c37ca5989335eea6a93ad14c50
```

  - 이미지 생성 확인

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
commit_test         second              c87fc1137ca8        54 seconds ago      188MB
commit_test         first               175f54ed8eb0        29 minutes ago      188MB
ubuntu              14.04               dea1945146b9        7 weeks ago         188MB
```

도커 이미지

- 도커 이미지 생성
  - 이미지 정보 확인

```
# docker inspect ubuntu:14.04
          "Layers": [
..
               "sha256:7fb9ba64f896b3a7001af9604a44243cfa663c84e414cd298ee8bc754feb5aa1",
..

# docker inspect commit_test:first
          "Layers": [
..
               "sha256:7fb9ba64f896b3a7001af9604a44243cfa663c84e414cd298ee8bc754feb5aa1",
               "sha256:3d40b70326a382e5d8664d65bf92d2e1fd97192a2038db41dfdc40336d6945ad"
..

# docker inspect commit_test:second
          "Layers": [
..
               "sha256:7fb9ba64f896b3a7001af9604a44243cfa663c84e414cd298ee8bc754feb5aa1",
               "sha256:3d40b70326a382e5d8664d65bf92d2e1fd97192a2038db41dfdc40336d6945ad",
               "sha256:11c4899c1d01be18ecb766770e927d3edc2bbfafc9366e2af6ad5d5d08ad2f9e"
..
```

  - docker inspect : 이미지 레이어의 아이디 값 확인
  - commit 실행으로 새로운 이미지 생성시 마다 레이어 값이 추가됨

도커 이미지

- 도커 이미지 생성
  - 이미지 히스토리 확인

```
# docker history commit_test:second
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
c87fc1137ca8        9 minutes ago       /bin/bash                                       13B
my second commit
175f54ed8eb0        37 minutes ago      /bin/bash                                       12B
my first commit
dea1945146b9        7 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           7 weeks ago         /bin/sh -c mkdir -p /run/systemd && echo '...   7B
<missing>           7 weeks ago         /bin/sh -c sed -i 's/^#\s*\(deb.*universe\...   2.75kB
<missing>           7 weeks ago         /bin/sh -c rm -rf /var/lib/apt/lists/*          0B
<missing>           7 weeks ago         /bin/sh -c set -xe   && echo '#!/bin/sh' >...   195kB
<missing>           7 weeks ago         /bin/sh -c #(nop) ADD file:8f997234193c2f5...   188MB
```

    - docker history [이미지명] : 이미지 생성 히스토리 출력

도커 이미지

- 도커 이미지 삭제
  - commit_test:first 이미지 삭제

```
# docker rmi commit_test:first
Error response from daemon: conflict: unable to remove repository reference "commit_test:first" (must
force) - container 77edfe2e5e69 is using its referenced image 175f54ed8eb0
```

    - 이미지를 사용 중인 컨테이너 존재, 이미지 레이어 삭제 할 수 없음
    - docker rm -r 로 삭제 가능, 이미지 레이어는 삭제되지 않고 이름만 삭제됨

  - 컨테이너 삭제 후 이미지 삭제

```
# docker stop commit_test2 && docker rm commit_test2
commit_test2

# docker rmi commit_test:first
Untagged: commit_test:first
```

    - Untagged : 이미지 레이어에 부여된 이름만 삭제, 실제 레이어 삭제 안됨
      commit_test:second 레이어가 참조하고 있기 때문

도커 이미지

- 도커 이미지 삭제
  - commit_test:second이미지 삭제

```
# docker rmi commit_test:second
Untagged: commit_test:second
Deleted: sha256:c87fc1137ca81f04246608adc68efb47cfd0c5c37ca5989335eea6a93ad14c50
Deleted: sha256:8f201d21712daecc4b9357cfa191e072f400e8c6c446fb99a52613277c9ebab7
Deleted: sha256:175f54ed8eb03cbd3eb52dcf0fd9af84b099abfe00f85007c65424b7bbf513d4
Deleted: sha256:1081a3cb494cf37f1821d0f410582e5939cdbeaa90244b2e569690686adde3f0
```

    - Untagged : 이름이 삭제 되었고,
    - Deleted : 실제 이미지 레이어 삭제되었음을 의미함
      이미지 삭제는 부모 레이어가 존재 하지 않을때 삭제됨
    - Ubuntu:14.04 이미지는 삭제 되지 않음

  - 이름만 지워진 댕글링(Dangling) 이미지 확인

```
# docker images -f dangling=true
```

  - 댕글링(Dangling) 이미지 한꺼번에 삭제

```
# docker image prune
```

도커 이미지

- 도커 이미지 추출
  - Ubuntu14.04 이미지 추출

```
# docker save -o ubuntu_14_04.tar ubuntu:14.04
# ls ubuntu_14_04.tar
ubuntu_14_04.tar
```

    - docker save : 컨테이너 커맨드, 이미지 이름과 태그, 메타데이터 포함 이미지 추출
    - - o : 추출될 파일명 지정

  - 이미지 로드

```
# docker rmi ubuntu:14.04
Untagged: ubuntu:14.04
Untagged: ubuntu@sha256:6e3e3f3c5c36a91ba17ea002f63e5607ed6a8c8e5fbbddb31ad3e15638b51ebc
Deleted: sha256:dea1945146b96542e6e20642830c78df702d524a113605a906397db1db022703
..

# docker load -i ubuntu_14_04.tar
c47d9b229ca4: Loading layer [==================================================>]  196.9MB/196.9MB
..
Loaded image: ubuntu:14.04
```

    - docker load : save 명령어로 추출된 이미지 로드
      기존 이미지 정보를 모두 포함하므로 동일하게 이미지가 생성됨

도커 이미지

- 도커 이미지 추출
  - docker save 명령어로 이미지를 만들면 컨테이너 설정 정보도 함께 저장됨
    ex) 컨테이너 변경사항, detached 모드, 컨테이너 커맨드 등
  - 컨테이너 정보 없이 파일시스템만 추출

```
# docker export -o rootFS.tar mycontainer
# docker import rootFS.tar myimage:0:0
```

  - docker export : 컨테이너 이미지를 파일로 추출
  - docker import : export 명령어로 추출된 이미지 파일을 새로운 이미지로 저장

  - 이미지를 파일로 추출하면 개수 만큼 디스크 공간을 차지함

도커 이미지

- 도커 이미지 배포
  - 파일배포

    추출한 이미지 파일을 복사 후 저장

    파일용량이 크고 도커엔진이 많을때 배포가 어려움
  - 도커허브

    이미지 클라우드 저장소

    회원 가입을 통한 Public 무료저장소 와 Private 유료 저장소 사용가능
  - 사설 레지스트리

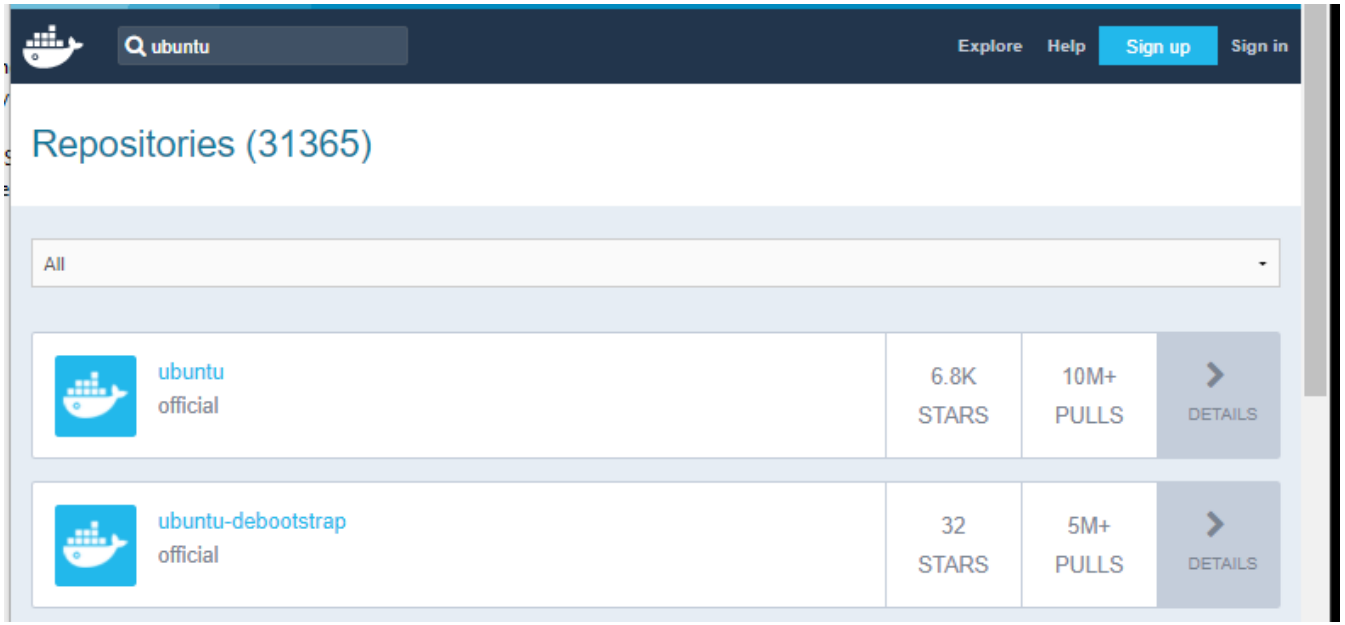
    사용자가 직접 도커 이미지 저장소(Docker Private Registry)를 직접 구성

    저장소 서버, 저장공간을 사용자가 직접 관리 해야함

    회사 사내망 환경에서 이미지 배포시 좋은방법

도커 이미지

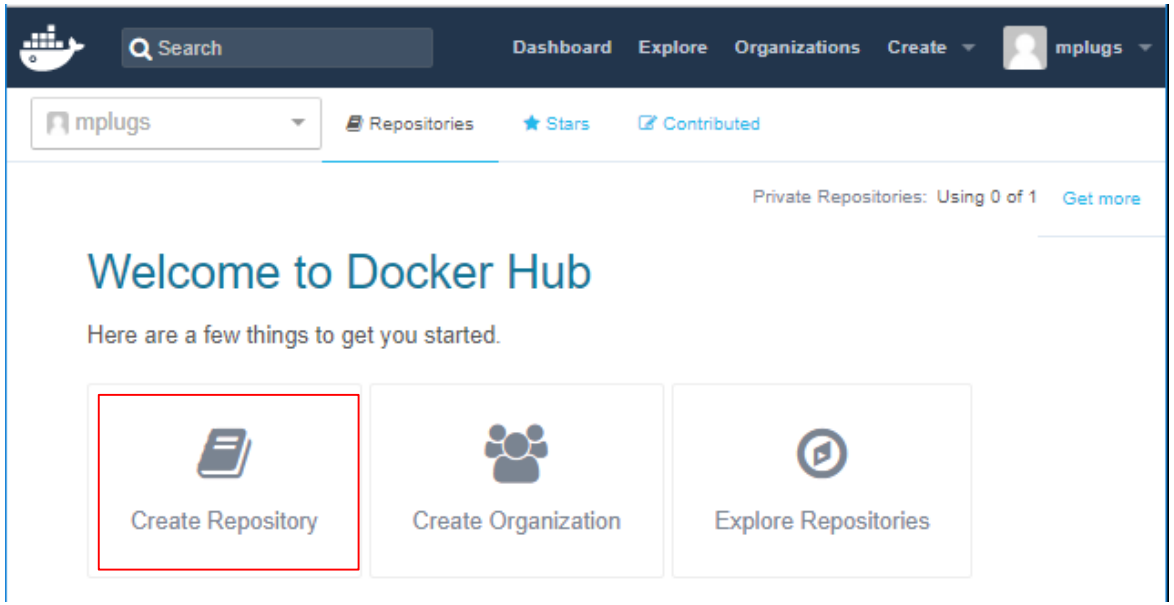
- 도커 허브 (https://hub.docker.com/)
  - ubuntu 이미지 검색결과



  - Sign up 클릭 후 계정생성

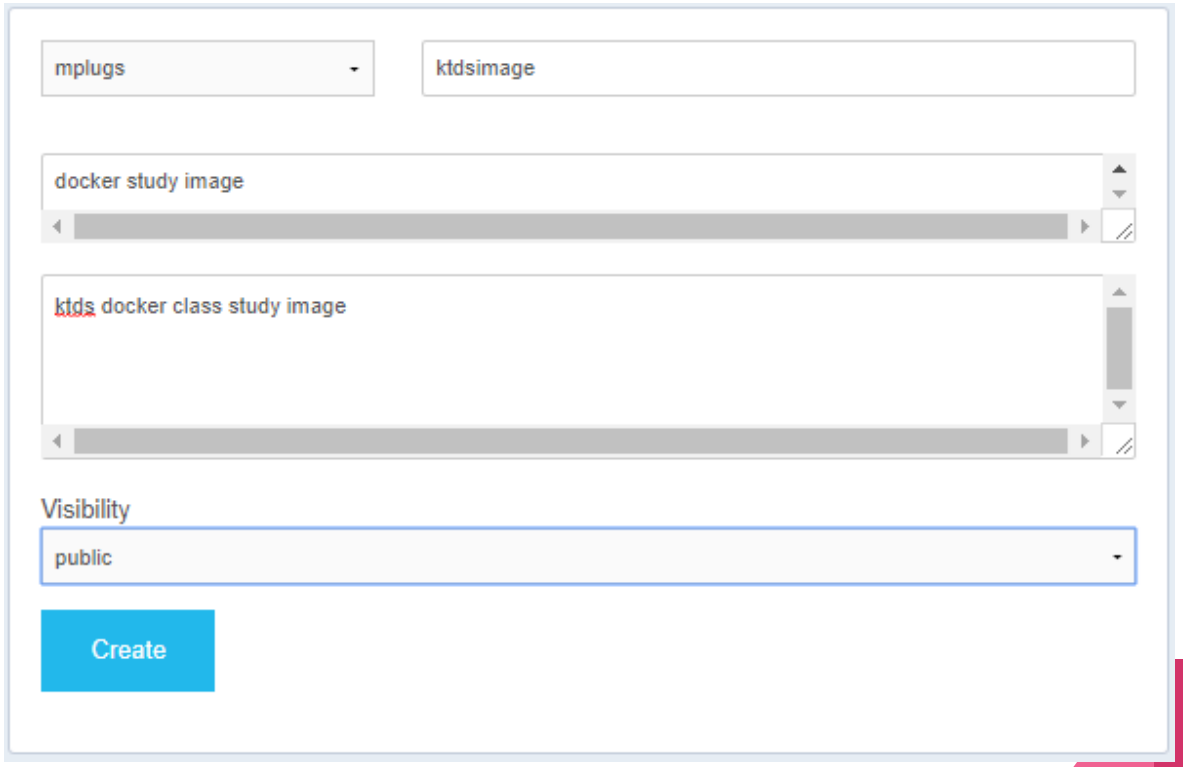
도커 이미지

- 이미지 저장소 생성
  - Create Repository 클릭



도커 이미지

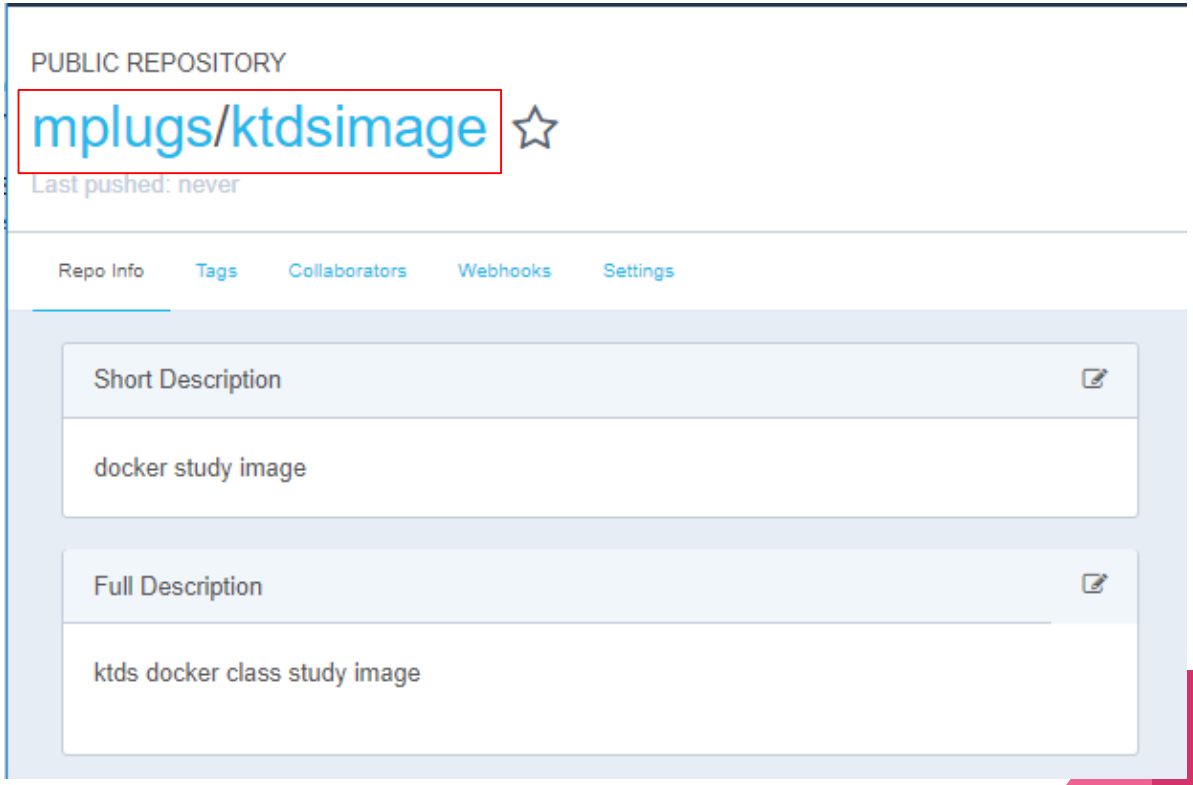
- 이미지 저장소 생성
  - 이미지 저장소 정보 입력



  - Visibiliy : Public(공개) , Private(비공개) 선택
  - Private 저장소는 1개만 무료

도커 이미지

- 이미지 저장소 생성
  - 저장소 이름 확인



  - 저장소 이름 : mplugs/ktdsimage (계정이름 : mplugs , 저장될 이미지 이름 : ktdsimage)

도커 이미지

- 저장소에 이미지 올리기
  - 컨테이너 생성 후 이미지 만들기

```
# docker run -i -t --name commit_container1 ubuntu:14.04
root@18d4a15f3473:/# echo my first push >> test

# docker commit commit_container1 ktdsimage:0.0
sha256:cc9784b889dde92473229a1d4dff0b64584a3004640b61783d2b77ab047e055c
```

    - ktdsimage:0.0 이미지(레이어) 생성됨
  - 이미지에 이름 추가 하기

```
# docker tag ktdsimage:0.0 mplugs/ktdsimage:0.0

# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mplugs/ktdsimage    0.0                 cc9784b889dd        8 minutes ago       188MB
ktdsimage           0.0                 cc9784b889dd        8 minutes ago       188MB
```

    - docker tag [기존 이미지 이름] [새롭게 생성될 이름]
    - ktdsimag:0.0 이미지에 mplugs/ktdsimage:0.0 이름을 추가
    - tag : 기존 이미지에 이름만 추가하는 명령, 기존 이미지는 삭제 되지 않음

도커 이미지

- 저장소에 이미지 올리기
  - 도커 허브 로그인

```
# 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: mplugs
Password:
Login Succeeded
```

    - 인터넷과 연결이 되어 있어야 함
    - 접속 계정 과 패스워드 입력

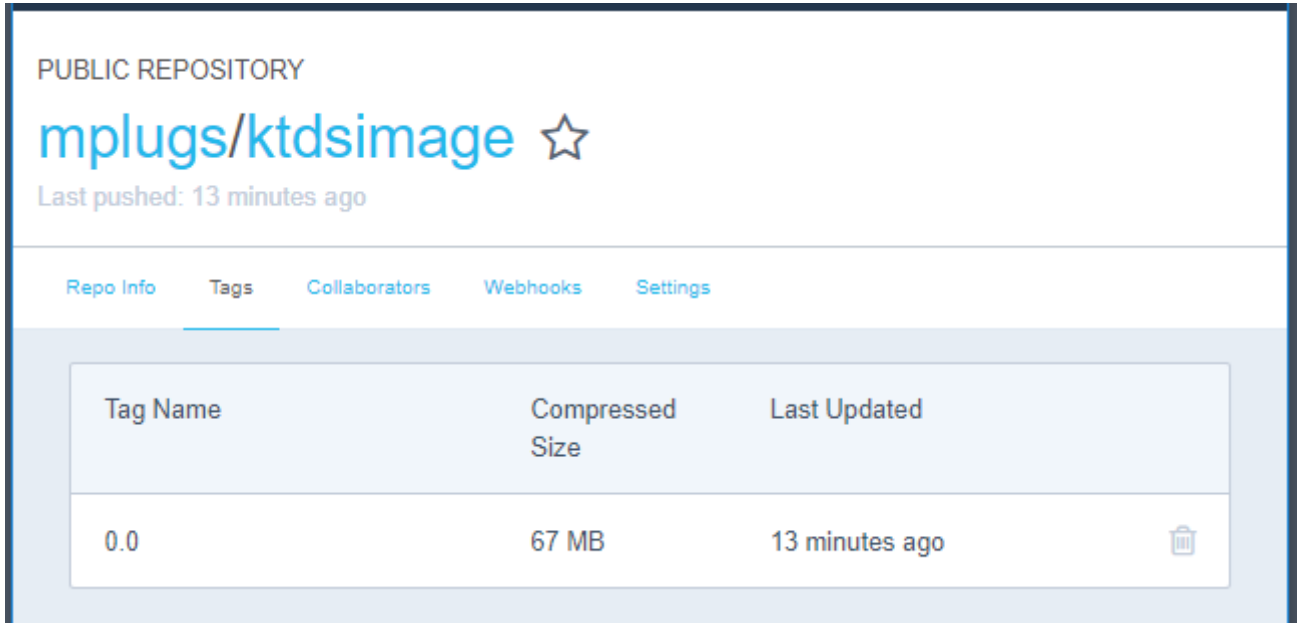
  - 이미지에 이름 추가 하기

```
# docker push mplugs/ktdsimage:0.0
The push refers to a repository [docker.io/mplugs/ktdsimage]
e9efe767c47f: Pushed
7fb9ba64f896: Mounted from library/ubuntu
..
0.0: digest: sha256:5e2c9c48869c62f05d5d0af48334f0ca286fefbab98e6d7689115aa50f18681f size: 1566
```

    - docker push : 이미지 저장소에 이미지 업로드
    - 이미지는 하나만 업로드됨, ubuntu14.04 이미지는 도커허브에 이미 존재하기 때문

도커 이미지

- 저장소에 이미지 올리기
  - 도커 허브 저장소 이미지 업로드 확인



  - Tag 버튼을 클릭하면 이미지 업로드 이미지를 확인할 수 있음

도커 이미지

- 저장소에서 이미지 내려받기
  - 컨테이너 중지, 삭제 후 이미지 삭제

```
# docker stop commit_container1
commit_container1

# docker rmi mplugs/ktdsimage:0.0
Untagged: mplugs/ktdsimage:0.0
Untagged: mplugs/ktdsimage@sha256:5e2c9c48869c62f05d5d0af48334f0ca286fefbab98e6d7689115aa50f18681f
```

  - 이미지 리스트 확인

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ktdsimage           0.0                 cc9784b889dd        18 minutes ago      188MB
nginx               latest              40960efd7b8f        6 days ago          108MB
centos              7                   d123f4e55e12        7 days ago          197MB
..
```

    - 현재 도커엔진에는 mplugs/ktimage:0.0 이미지는 삭제되어 보이지 않음

도커 이미지

- 저장소에서 이미지 내려받기
  - 도커허브로 부터 이미지 다운로드

```
# docker pull mplugs/ktdsimage:0.0
0.0: Pulling from mplugs/ktdsimage
Digest: sha256:5e2c9c48869c62f05d5d0af48334f0ca286fefbab98e6d7689115aa50f18681f
Status: Downloaded newer image for mplugs/ktdsimage:0.0
```

    - docker pull [이미지주소] : 도커허브로 부터 이미지 다운로드

  - 이미지 리스트 확인

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ktdsimage           0.0                 cc9784b889dd        18 minutes ago      188MB
mplugs/ktdsimage    0.0                 cc9784b889dd        18 minutes ago      188MB
nginx               latest              40960efd7b8f        6 days ago          108MB
centos              7                   d123f4e55e12        7 days ago          197MB
..
```

도커 이미지

- 사설 레지스트리 저장소 생성
  - 사설 레지스트리 컨테이너 생성

```
# docker run -d --name myregistry \
> -p 5000:5000 \
> --restart=always \
> registry:2.6
Unable to find image 'registry:2.6' locally
2.6: Pulling from library/registry
49388a8c9c86: Pull complete
..
Digest: sha256:d837de65fd9bdb81d74055f1dc9cc9154ad5d8d5328f42f57f273000c402c76d
Status: Downloaded newer image for registry:2.6
eee2cb731c384e4102a20f0d69722a222134c3c31c80470fc67a2c023252f115
```

  - --restart=always : 컨테이너가 정지되면 다시 시작

    도커 엔진을 재시작하면 컨테이너도 재시작 됨

  - --restart=on-failure : 컨테이너 종료코드가 0이 아닐때 5번까지 재시작 시도
  - --restart=unless-stopped : 컨테이너를 stop 정지했다면, 도커 엔진을 재시작 해도 컨테이너가 재시작 되지 않도록 설정

도커 이미지

- 사설 레지스트리 저장소 생성
  - Centos7 docker-distribution 설치

```
# yum install docker-distribution
# systemctl enable docker-distribution
# systemctl start docker-distribution
```

  - tcp6 사용중지 커널 부팅 옵션 변경

```
# vi /etc/default/grub
add ipv6.disable=1 at line 6,like:
GRUB_CMDLINE_LINUX="ipv6.disable=1 ..."

#grub2-mkconfig -o /boot/grub2/grub.cfg
#reboot
```

    - 레지스트리 포트 5000번이 TCP6만 열리는 증상발생
    - ipv6 사용중지를 커널 부팅옵션으로 변경 후 재부팅
  - tcp 5000번 포트 확인

```
# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
..
tcp        0      0 0.0.0.0:5000            0.0.0.0:*               LISTEN      2202/docker-proxy
```

도커 이미지

- 사설 레지스트리 이미지 생성
  - 사설 레지스트리 접속 테스트

```
# curl localhost:5000/v2/
{}
```

    - 레지스트리 컨테이너는 기본적으로 5000번 포트를 사용

  - 사설 레지스트리 업로드 이미지 Tag 생성

```
# docker tag ktdsimage:0.0 192.168.35.51:5000/ktdsimage:0.0
```

  - 이미지 리스트 확인

```
# docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
192.168.35.51:5000/ktdsimage    0.0                 cc9784b889dd        About an hour ago   188MB
ktdsimage                       0.0                 cc9784b889dd        About an hour ago   188MB
```

도커 이미지

- 사설 레지스트리 이미지 업로드
  - 사설 레지스트리에 이미지 Push 하기

```
# docker push 192.168.35.51:5000/ktdsimage:0.0
The push refers to a repository [192.168.35.51:5000/ktdsimage]
Get https://192.168.35.51:5000/v2/: http: server gave HTTP response to HTTPS client
```

    - 도커 데몬은 기본적으로 https를 통한 레지스트리 접근만 허용

  - 도커 엔진 옵션 변경

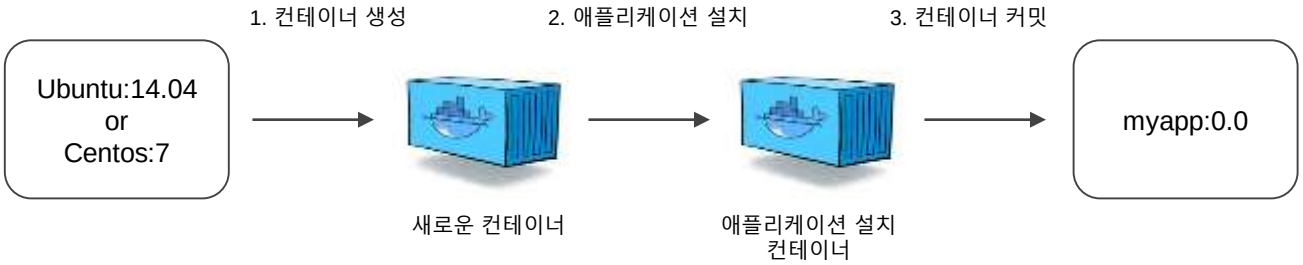
```
# vi /usr/lib/systemd/system/docker.service
..
ExecStart=/usr/bin/dockerd $DOCKER_OPTS
..
# DOCKER_OPTS="--insecure-registry=192.168.35.51:5000"
```

  - 이미지 업로드 명령 실행

```
# docker push 192.168.35.51:5000/ktdsimage:0.0
The push refers to a repository [192.168.35.51:5000/ktdsimage]
e9efe767c47f: Pushed
..
0.0: digest: sha256:5e2c9c48869c62f05d5d0af48334f0ca286fefbab98e6d7689115aa50f18681f size: 1566
0.0                 cc9784b889dd        About an hour ago   188MB
```

Dockerfile

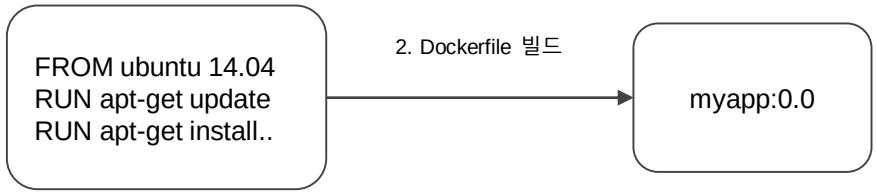
- 컨테이너로 이미지 생성 방법
  1. 기본 OS 이미지로 컨테이너 생성
  2. 애플리케이션 설치 및 환경설정, 소스코드 복제
  3. 컨테이너 이미지 커밋(commit)



  - 애플리케이션 설치 환경구성을 위한 매뉴얼작업이 필요
  - 애플리케이션 구동 이미지로 커밋하기 때문에 이미지 동작을 보장

Dockerfile

- Dockerfile로 이미지 생성 방법
  1. 매뉴얼 작업을 기록한 Dockerfile 생성
  2. 빌드 명령어가 Dockerfile 을 읽어 이미지를 생성



  - 이미지를 직접 생성 또는 커밋 해야 하는 수고스러움을 줄여줌
  - 애플리케이션 빌드를 자동화
  - 도커 허브의 신뢰할 수 있는 이미지를 바탕으로 쉽게 이미지 배포 가능

Dockerfile

- Dockerfile 작성
  - 컨테이너 빌드에 필요한 작업 명령이 저장된 특수 파일
  - 도커 엔진은 현재 디렉토리의 “Dockerfile” 이라는 이름의 파일을 참조

시나리오 : 웹서버를 설치하고, 로컬에 있는 test.html -> 컨테이너 /var/www/html 복사

- 로컬 test.html 파일 생성

```
# echo test >> test.html
```

- 아파치 웹서버가 설치된 이미지를 빌드하는 Dockerfile 생성

```
# echo test >> test.html

# vi Dockerfile
FROM ubuntu:14.04
MAINTAINER teacher
LABEL "purpose"="practice"
RUN apt-get update
RUN apt-get install apache2 -y
ADD test.html /var/www/html
WORKDIR /var/www/html
RUN ["/bin/bash", "-c", "echo hello >> test2.html"]
EXPOSE 80
CMD apachectl -DFOREGROUND
```

Dockerfile

- Dockerfile 작성
  - 한줄이 하나의 명령어
  - 명령어는 대소문자 상관 없으나, 일반적으로 대문자를 사용

- FROM : 베이스가 될 이미지 정의
- MAINTAINER : 이미지를 생성한 개발자 정보, 도커 1.13.0 버전 이후 사용하지 않음
- LABEL : 이미지에 메타데이터 추가, “키:값” 형태로 정의
  docker inspect 명령어로 이미지 메타데이터 정보 확인가능
- RUN : 이미지를 만들기 위해 컨테이너 내부에서 명령어 실행
  명령어의 옵션/인자 값은 배열형태로 전달
  Dockerfile 명령어는 쉘을 사용하지 않기 때문에 쉘을 정의해야한다
  예) RUN [“sh”, “-c”, “echo $MY_ENV”]
- ADD : Dockerfile 이 위치한 디렉토리의 파일 -> 이미지에 추가
- WORKDIR : 명령어를 실행할 디렉토리 정의, cd 명령과 같은기능
- EXPOSE : 생성한 이미지에서 노출할 포트 정의
- CMD : 컨테이너가 시작될때 실행되는 명령설정, 한번만 사용가능

Dockerfile

- Dockerfile 빌드
  - 이미지 생성

```
# docker build -t mybuild:0.0 ./
Sending build context to Docker daemon  3.072kB
Step 1/10 : FROM ubuntu:16.04
 ---> dd6f76d9cc90
Step 2/10 : MAINTAINER teacher
 ---> Running in 72ed646689bf
 ---> bdcec47ac282
..
```

    - docker build : Dockerfile 을 이용한 이미지 생성 명령
      -t : 생성할 이미지 이름 정의 옵션
      이름을 정의 하지 않으면 16진수 형태로 이름이 저장됨
  - 생성된 이미지 확인

```
# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
mybuild                       0.0                 8df4c18a7a0c        25 seconds ago      260MB
192.168.35.51:5000/ktdsimage  0.0                 cc9784b889dd        13 hours ago        188MB
..
```

Dockerfile

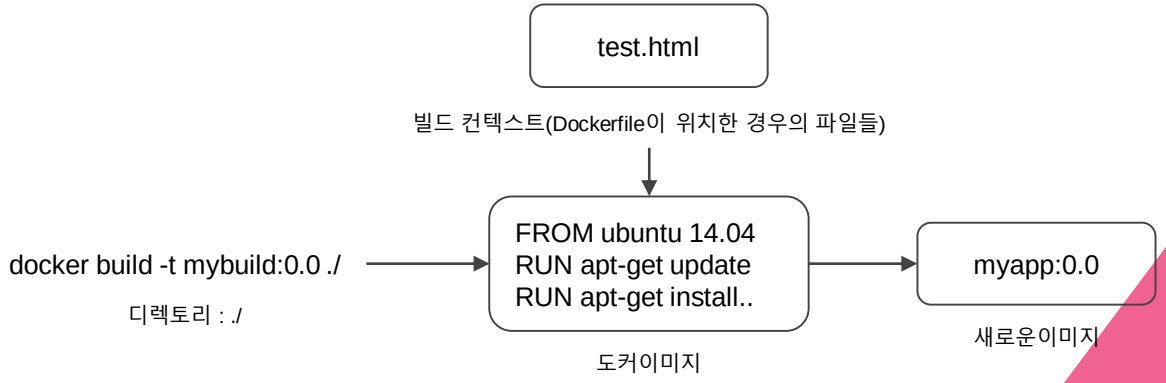
- Dockerfile 빌드
  - 생성된 이미지로 컨테이너 실행

```
# docker run -d -P --name myserver mybuild:0.0
2e33d49d5935be9b91653f926b9842238363302b6a4f419ad1576e9a2451c0c1
```

    - -P : EXPOSE 로 노출된 포트를 호스트에서 사용가능한 포트에 차례로 연결
  - 컨테이너와 연결된 호스트 포트 확인

```
# docker port myserver
80/tcp -> 0.0.0.0:32768
```

    - docker port [컨테이너] : 컨테이너 포트와 호스트 포트 연결정보 출력



Dockerfile

- 빌드 컨텍스트(Context)
  - docker build 실행 첫번째 로그

```
# docker build -t mybuild:0.0 ./
Sending build context to Docker daemon  3.072kB
```

  - 이미지 생성시 ./ 디렉토리의 컨텍스트 파일이 전송됨
  - 컨텍스트 파일은 명령어 마지막에 지정하는 위치의 파일 및 디렉토리 전부 포함
  - 불필요한 파일은 .dockerignore 파일에 정의필요

```
# vi .dockerignore
test2.html
*.html
*/*.html
!test.htm?
```

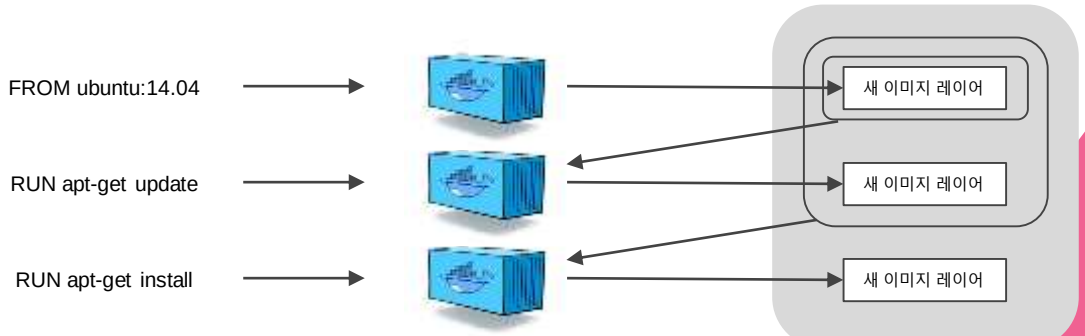
    - ! : 컨텍스트 제외 하지 않을 파일 지정

Dockerfile

- Dockerfile을 이용한 컨테이너 생성과 커밋
  - 이미지가 만들어지는 과정

```
Sending build context to Docker daemon  3.072kB
Step 1/10 : FROM ubuntu:16.04
 ---> dd6f76d9cc90
Step 2/10 : MAINTAINER teacher
 ---> Running in 72ed646689bf
 ---> bdcec47ac282
Removing intermediate container 72ed646689bf
Step 3/10 : LABEL "purpose" "practice"
 ---> Running in 06db3b26f9fb
 ---> 71723b26562b
...
```

  - ADD, RUN 등의 명령어가 실행될 때마다 새로운 컨테이너 레이어 생성
  - 최종 이미지 생성까지 임시 컨테이너 레이어 생성 후 삭제



Dockerfile

- 캐시를 이용한 이미지 빌드
  - Dockerfile2 파일 생성

```
# vi Dockerfile2
FROM ubuntu:14.04
MAINTAINER teacher
LABEL "purpose"="practice"
RUN apt-get update
```

  - Dockerfile2 파일을 이용한 이미지 빌드

```
# docker build -f Dockerfile2 -t mycache:0.0 ./
Sending build context to Docker daemon  4.096kB
Step 1/10 : FROM ubuntu:16.04
 ---> dd6f76d9cc90
Step 2/10 : MAINTAINER teacher
 ---> Using cache
 ---> bdcec47ac282
..
Successfully built 8df4c18a7a0c
```

    - -f : docker build 에 사용할 Dockerfile 지정
    - 이전에 빌드했던 Dockerfile 과 같은 내용이 있다면 이전 이미지를 활용

Dockerfile

- 캐시를 이용한 이미지 빌드
  - 캐시로 사용할 이미지를 직접 지정하여 빌드

```
# docker build --cache-from nginx my_extend_nginx:0.0 .
```

    - --cache-from : 특정 이미지의 Dockerfile 캐시 이용
    - nginx:latest 이미지를 빌드하는 Dockerfile 에 일부 내용을 추가해 활용

  - 이미 존재하는 캐시를 사용하지 않을 경우

```
# docker build --no-cache -t mycache:0.0 .
```

    - --no-cache : 기존 빌드에 사용된 캐시를 사용하지 않고 Dockerfile을 첨부터 다시 이미지 레이어를 생성함

Dockerfile

- Dockerfile 기타 명령어
  - ENV : 도커에서 사용할 환경 변수 설정

```
# vi Dockerfile
FROM ubuntu:14.04
ENV test /home
WORKDIR $test
RUN touch $test/mytouchfile
```

    - test 변수에 /home 값을 설정
  - myenv 이미지 빌드

```
# docker build -t myenv:0.0 ./
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM ubuntu:14.04
 ---> dea1945146b9
..
Successfully built 9a64ed22c0fa
Successfully tagged myenv:0.0
```

  - 컨테이너 생성 후 변수 확인

```
# docker run -i -t --name env_test myenv:0.0 /bin/bash
root@1dd86a895239:/home# echo $test
/home
```

Dockerfile

- Dockerfile 기타 명령어
  - -e 옵션으로 ENV 설정값 덮어쓰기

```
# docker run -i -t --name env_test_override \
> -e test=myvalue \
> myenv:0.0 /bin/bash
root@5bdbdd8f3dd2:/home# echo $test
myvalue
```

    - test 변수값이 /home -> myvalue 로 변경됨
  - 환경변수 설정된 경우, 설정되지 않은 경우 확인

```
# vi Dockerfile
FROM ubuntu:14.04
ENV my_env my_value
RUN echo ${my_env:-value} / ${my_env:+value} / ${my_env2:-value} / ${my_env2:+value}

# docker build ./
Sending build context to Docker daemon  4.096kB
..
Step 3/3 : RUN echo ${my_env:-value} / ${my_env:+value} / ${my_env2:-value} / ${my_env2:+value}
 ---> Running in a1153a71fa0c
my_value / value / value /
..
```

Dockerfile

- Dockerfile 기타 명령어
  - VOLUME : 호스트와 공유할 컨테이너 내부의 디렉토리 설정

```
# vi Dockerfile
FROM ubuntu:14.04
ENV my_env my_value
RUN mkdir /home/volume
RUN echo test >> /home/volume/testfile
VOLUME /home/volume
```

    - /home/volume 디렉토리를 호스트와 공유
  - volume_test 이미지 빌드 후 컨테이너 생성

```
# docker build -t myvolume:0.0 ./
..

# docker run -i -t -d --name volume_test myvolume:0.0
6cfadd4c0b4bd0baefc4fa13821ea70ce5e9a19b0b363e70a07ea85ef7ecdc61
```

  - 볼륨 리스트 확인

```
# docker volume ls
DRIVER              VOLUME NAME
local               01c9539670ad5991eff1bcc7ca4200bfd7ff0167c1d79f6bc18b847eba852b17
```

Dockerfile

- Dockerfile 기타 명령어
  - ARG : build 명령어를 실행할 때 추가로 입력 받아 Dockerfile 내 사용된 변수값 설정

```
# vi Dockerfile
FROM ubuntu:14.04
ARG my_arg
ARG my_arg_2=value2
RUN touch ${my_arg}/mytouch
```

    - my_arg 는 build 명령 실행시 입력, my_arg_2 는 Dockerfile 에서 설정
  - myarg 이미지 빌드 실행

```
# docker build --build-arg my_arg=/home -t myarg:0.0 ./
..
```

  - 볼륨 리스트 확인

```
# docker run -i -t --name arg_test myarg:0.0
root@ca4abb4ef31e:/# ls /home/mytouch
/home/mytouch
```

Dockerfile

- Dockerfile 기타 명령어
  - USER : USER로 사용자 계정을 설정하면, 그 아래 명령은 해당 사용자 권한으로 실행

```
..
RUN groupadd -r author && useradd -r -g author user1
USER user1
..
```

    - user1 사용자 계정으로 하위 명령어 실행 됨

  - ONBUILD : 빌드된 이미지를 기반으로 하는 다른 이미지가 Dockerfile 로 실행될때 실행할 명령어를 추가

```
# vi Dockerfile
FROM ubuntu:14.04
RUN echo “this is onbuild test”
ONBUILD RUN echo “onbuild!” >> /onbuild_file

# docker build ./ -t onbuild_test:0.0

# docker run -i --rm onbuild_test:0.0 ls /
bin boot dev etc home lib lib64 media mnt opt
proc root run sbin srv sys tmp usr var
```

    - onbuild_test 이미지 생성 후 컨테이너 실행
    - /onbuild_file 파일이 확인되지 않음

Dockerfile

- Dockerfile 기타 명령어
  - ONBUILD 가 적용된 이미지를 기반으로 하는 Dockerfile

```
# vi Dockerfile2
FROM onbuild_test:0.0
RUN echo “this is child image!”
```

  - ONBUILD 가 적용된 이미지를 기반으로 하는 Dockerfile

```
# docker build -f ./Dockerfile2 ./ -t onbuild_test:0.1
Sending build context to Docker daemon  4.096kB
Step 1/2 : FROM onbuild_test:0.0
# Executing 1 build trigger...
Step 1/1 : RUN echo "onbuild!" >> /onbuild_file
 ---> Running in 50d56b5426b1
 ---> 3bb26a906dda
..
```

  - ONBUILD 가 적용된 이미지를 기반으로 하는 Dockerfile

```
# docker run -i -t --rm onbuild_test:0.1 ls /
bin   dev  home  lib64    mnt               opt   root  sbin    sys  usr
boot  etc  lib             media       onbuild_file  proc  run   srv          tmp  var
```

    - ONBUILD 적용된 이미지로 부터 컨테이너 생성
    - ONBUILD 실행 명령어가 적용되어 onbuild_file 확인됨

Dockerfile

- Dockerfile 기타 명령어
  - STOPSIGNAL : 컨테이너가 정지될 때 사용될 시스템 콜의 종류 지정

```
# vi Dockerfile
FROM ubuntu:14.04
STOPSIGNAL SIGKILL
```

  - stopsignal 이미지 빌드 후 컨테이너 생성하기

```
# docker build . -t stopsignal:0.0
Sending build context to Docker daemon  4.096kB
..
Step 2/2 : STOPSIGNAL SIGKILL
..
Successfully tagged stopsignal:0.0

# docker run -itd --name stopsignal_container stopsignal:0.0
a349b4bf3cf4ae50b2d1e324c5a9eacfc54d36dd055718fdd564de0c51e5b0ae
```

  - 컨테이너 정보 확인

```
# docker inspect stopsignal_container | grep Stop
            "StopSignal": "SIGKILL"
```

    - StopSignal 값이 SIGKILL 로 설정됨 확인
    - 아무 값도 설정하지 않으면, 기본값은 SIGTERM

Dockerfile

- Dockerfile 기타 명령어
  - HEALTHCHECK : 이미지로 부터 생성된 컨테이너의 애플리케이션 상태 체크 설정
    애플리케이션 프로세스는 살아있으나, 동작하지 않는 상태 방지

```
# vi Dockerfile
FROM nginx
RUN apt-get update -y && apt-get install curl -y
HEALTHCHECK --interval=1m --timeout=3s --retries=3 CMD curl -f http://localhost || exit 1
```

    - --interval : 컨테이너 상태 체크 주기
    - --timeout : 설정한 시간을 초과하면 상태 체크 실패 간주
    - --retries : 설정 횟수만큼 상태 체크에 실패시 unhealth 생태로 설정
    - 1분에 한번씩 curl 명령 실행, 3초응답 지연이 3회 발생시 unhealth 로 상태변경
  - nginx:healthcheck 이미지 빌드 하기

```
docker build ./ -t nginx:healthcheck
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM nginx
..
Successfully built bc27a8263d1d
Successfully tagged nginx:healthcheck
```

Dockerfile

- Dockerfile 기타 명령어
  - healthcheck 컨테이너 생성

```
# docker run -d -P nginx:healthcheck
d4061b732e91acfec1098581b3b3e3859a1c1e70a96e30bd5b00cb271da0a3ed
```

    - -P : 컨테이너 포트를 호스트의 랜덤포트로 노출
  - 컨테이너 상태 확인

```
# docker ps | grep nginx
d4061b732e91        nginx:healthcheck   "nginx -g 'daemon ..."   3 minutes ago       Up 3 minutes
(healthy)   0.0.0.0:32768->80/tcp    stupefied_bose
```

  - 컨테이너 정보 확인

```
# docker inspect d4061b | grep -B 3 -A 6 health
            "StartedAt": "2017-11-14T14:31:06.962028619Z",
            "FinishedAt": "0001-01-01T00:00:00Z",
            "Health": {
                "Status": "healthy",
                "FailingStreak": 0,
                "Log": [
                    {
                        "Start": "2017-11-14T23:33:07.011529837+09:00",
                        "End": "2017-11-14T23:33:07.055825695+09:00",
                        "ExitCode": 0,
..
```

Dockerfile

- Dockerfile 기타 명령어
  - SHELL : 이미지 빌드중 명령 실행을 위한 셸 지정

    기본값 [Linux : /bin/sh -c , Windows : cmd /S /C]

```
# vi Dockerfile
FROM node
RUN echo hello, node!
SHELL ["/usr/local/bin/node"]
RUN -v
```

  - nodetest 이미지 빌드 하기

```
# docker build ./ -t nodetest
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM node
..
v9.1.0
..
Successfully built 1fad63a47199
Successfully tagged nodetest:latest
```

    - /usr/local/bin/node 의 버전 출력 확인

Dockerfile

- Dockerfile 기타 명령어
  - COPY 와 ADD 차이점
    - COPY : 로컬 디렉토리에서 읽어 들인 컨텍스트를 이미지 파일에 복사
      사용 형식은 ADD 와 같음

```
COPY test.html /home/
COPY [“test.html”, “/home/”]
```

      - COPY 는 파일만 이미지에 복사 가능
    - ADD : 로컬파일, URL, tar 파일 등도 복사 가능

```
ADD http://ftp.daumkakao.com/centos/timestamp.txt /home

ADD test.tar /home
```

      - tar 파일은 자동으로 해제해서 추가 됨

Dockerfile

- Dockerfile 기타 명령어
  - ENTRYPOINT 와 CMD 차이점
  - ENTRYPOINT : CMD 와 동일하게 컨테이너가 시작될 때 수행할 명령 실행
    커맨드를 인자로 사용할 수 있는 **스크립트의 역활**을 할 수 있음
    - CMD 명령 실행시 /bin/bash

```
# docker run -i -t --name no_entropoint ubuntu:14.04 /bin/bash
root@760b8d745ecc:/#
```

  - /bin/bash 명령을 실행
    - ENTRYPOINT 명령 실행시 /bin/bash

```
# docker run -i -t --entrypoint="echo" --name yes_entrypoint ubuntu:14.04 /bin/bash
/bin/bash
```

  - echo 명령의 인자값으로 “/bin/bash” 사용, 결국 echo 명령 실행
    - CMD 와 ENTRYPOINT 둘다 설정되지 않으면 이미지 빌드시 에러발생

Dockerfile

- Dockerfile 기타 명령어
  - entrypoint 를 이용한 스크립트 실행

```
# docker run -i -t --name entrypoint_sh --entrypoint=”/test.sh” ubuntu:14.04 /bin/bash
```

    - 실행할 스크립트는 컨테이너 내부에 존재 해야함
    - COPY 혹은 ADD 명령을 이용해 이미지 빌드시 복사 필요

  - ENTRYPOINT 와 --entrypoint 의 우선순위
    - Dockerfile 에 정의한 ENTRYPOINT 는 Docker run 명령에서 --entrypoint 옵션으로 재정의 된 명령으로 덮어 쓰입니다.
  - ENTRYPOINT 사용 예

```
# vi Dockerfile
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install apache2 -y
ADD entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/bin/bash", "/entrypoint.sh"]
```

    - [“bin/bash”, “entrypoint.sh”] : JSON 형태 배열로 명령어 정의 가능

Dockerfile

- Dockerfile 기타 명령어
  - JSON 배열 형태와 일반 형식의 차이점
    - 일반형식 사용의 예

```
CMD echo test
# -> /bin/sh -c echo test

ENTRYPOINT /entrypoint.sh
# -> /bin/sh -c /entrypoint.hs
```

      - 명령 실행시 “/bin/sh -c” 가 기본값으로 실행됨
    - JSON 배열 형식 사용의 예

```
CMD [“echo”, “test”]
# -> echo test

ENTRYPOINT [“/bin/bash”, “/entrypoint.sh”]
# -> /bin/bash /entrypoint.sh
```

      - 배열로 선언된 실제 명령만 실행됨

수고하셨습니다.