Docker & Kubernetes

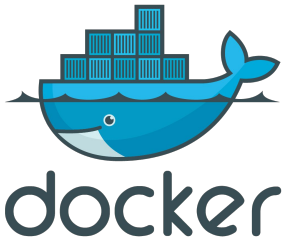
강사 : 김정석

목차

1장 도커란 ?

- 가상머신 과 도커 컨테이너
- 도커엔진 설치
- 도커 데몬

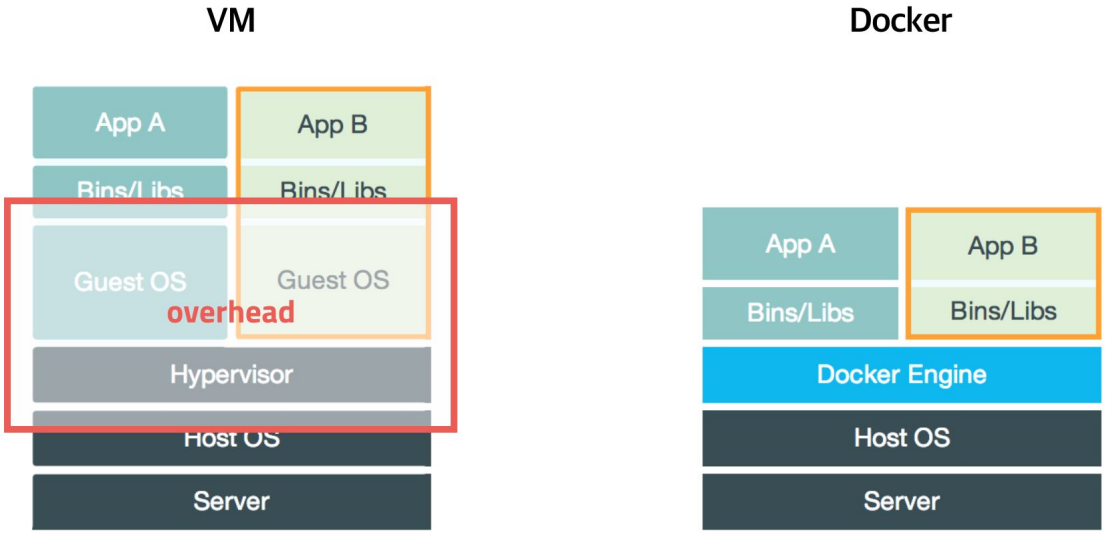
도커(Docker)



- 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/

리눅스 도커엔진 설치

- 도커 설치 지원 플랫폼 참조

  https://docs.docker.com/engine/installation/#time-based-release-schedule

- 실습 환경
  - Centos7 최신버전 설치
  - Ubuntu 16.04 최신버전
- 리눅스 커널 버전 확인
  - 리눅스 커널 버전 3.10 이상 확인
- Centos7

```
# uname -r
3.10.0-514.el7.x86_64
```

- Ubuntu 16.04

```
# uname -r
4.4.0-116-generic
```

Centos 7 설치후 기본 설정변경

- Selinux 설정 비활성화

```
# vi /etc/selinux/config
SELINUX=disabled
# reboot
```

- NetworkManager 서비스 중지

```
# systemctl stop NetworkManager
# systemctl disable NetworkManager
```

- 방화벽 서비스 중지

```
# systemctl stop firewalld
# systemctl disable firewalld
# iptables -L
```

Centos7 도커엔진 설치

- Old 도커엔진 버전 삭제

```
# yum remove docker docker-common docker-selinux docker-engine
```

- Yum Util 도구 Device-Mapper 드라이버 설치

```
yum install -y device-mapper-persistent-data lvm2
```

- 최신 도커엔진 설치용 리포지터리 추가

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

- 최신 도커엔진 설치

```
# yum install docker-ce
```

- 도커엔진 시작

```
# systemctl start docker
```

- 도커엔진 테스트

```
# docker run hello-world
```

Ubuntu16.04 도커엔진 설치

- 최신 버전 업데이트 및 패치적용

```
# apt-get update && apt-get upgrade -y
```

- 최신 도커엔진 설치

```
# apt install docker.io -y
```

- 도커엔진 시작

```
# systemctl start docker
```

- 도커엔진 테스트

```
# docker run hello-world
```

도커 데몬

- 도커 구조
  - ➢ 도커 명령어

```
# which docker
/usr/bin/docker
```

    - docker : 클라이언트 CLI 명령어
  - ➢ 도커 프로세스

```
# ps aux | grep docker
avahi     706  0.0  0.0  30204  1816 ?        Ss   05:45   0:00 avahi-daemon: running [docker1.local]
root      923  0.3  2.2 675780 41280 ?        Ssl  05:45   0:21 /usr/bin/dockerd --insecure-registry 192.168.35.51:5000

# ls /var/run/docker.sock
/var/run/docker.sock
```

    - dockerd : 도커 엔진의 프로세스는 dockerd 파일로 실행
    - docker 클라이언트는 /var/run/docker.sock 유닉스 소켓을 통해 API 명령 호출

도커 데몬

- 도커 데몬 실행
  - ➢ 데커 데몬 서비스 실행

```
# systemctl start docker

# systemctl stop docker
```

  - ➢ 도커 데몬 서비스 자동 실행 설정

```
# systemctl enable docker
```

- 도커 데몬 직접 실행
  - ➢ 데커 데몬 실행

```
# dockerd --help

Usage:        dockerd COMMAND

A self-sufficient runtime for containers.

Options:
      --add-runtime runtime                Register an additional OCI compatible runtime (default [])
..
```

    - ○ docker 서비스는 dockerd 명령을 참조해 서비스를 시작

도커 데몬

- 도커 데몬 실행 옵션 설정
  - ➢ dockerd 명령에 옵션 설정

```
# dockerd -D -H tcp://0.0.0.0:2375 --insecure-registry=192.168.100.99:5000 --tls=false
```

  - ➢ 도커 데몬 서비스에서 옵션 변경

```
# vi /etc/systemd/system/multi-user.target.wants/docker.service
[Service]
..
ExecStart=/usr/bin/dockerd --insecure-registry 192.168.35.51:5000
..
```

    - ○ systemd 서비스의 실행 옵션을 변경 후 적용
    - ○ systemd 버전 업그레이스시 덮어씌워져 변경 될 수 있음

  - ➢ docker 서비스용 옵션 설정 파일 생성 후 변경

```
# mkdir -p /etc/systemd/system/docker.service.d
# vi /etc/systemd/system/docker.service.d/docker.conf
[Service]
..
ExecStart=/usr/bin/dockerd --insecure-registry 192.168.35.51:5000
```

    - ○ systemd 버전 업그레이드와 관계없이 영구 적용
    - ○ docker 서비스 재시작 후 적용확인

도커데몬

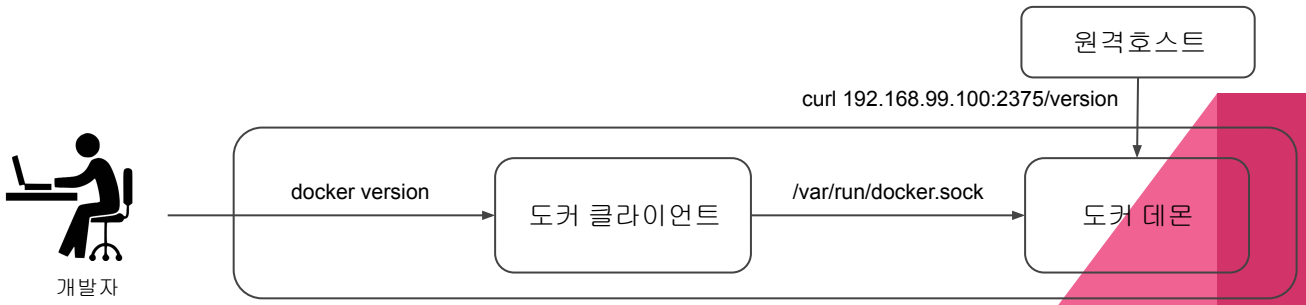
- 도커 데몬 실행 옵션 설정
  - ➢ 도커데몬 원격 API 사용가능 옵션

```
# dockerd -D -H tcp://192.168.99.100:2375
```

  - ○ -H : 도커 데몬 API 접근 통신 포트 설정
  - ○ 기본 API 값 : -H unix:///var/run/docker.sock (로컬 유닉스 소켓 사용)
  - ○ -H tcp://192.168.99.100:2375 (URL 로 http 요청을 보내 데몬 API 접근)
  - ➢ docker 클라이언트의 docker 데몬 주소 설정

```
[client]  # export DOCKER_HOST="tcp://192.168.199.100:2375"
[client]  # docker version

[client]  # docker -H tcp://192.168.100.2375 version
```

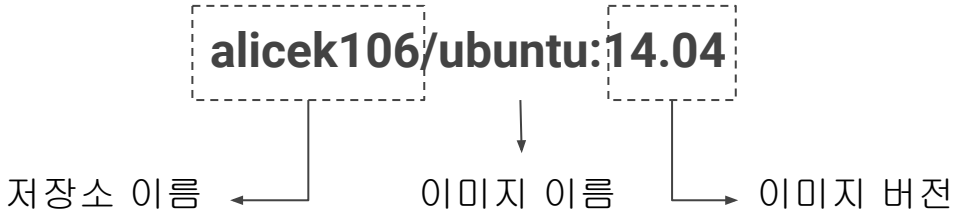


2장 도커 엔진

- 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



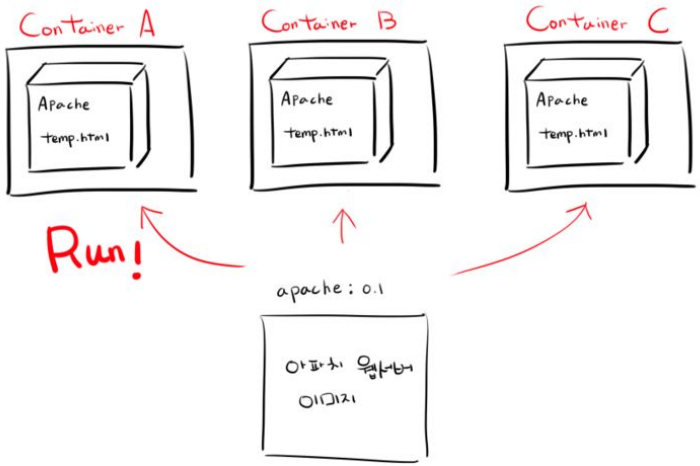
  - 저장소 이름 : 이미지가 저장된 장소, 이름이 없으면 도커 허브(Docker Hub)로 인식
  - 이미지 이름 : 이미지의 역할을 나타낼 이름, 생략 불가능
  - 이미지 버전 : 이미지 버전정보, 생략하면 latest 로 인식

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 : 컨테이너와 상호 입출력 가능 옵션
  - -t : 셸을 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

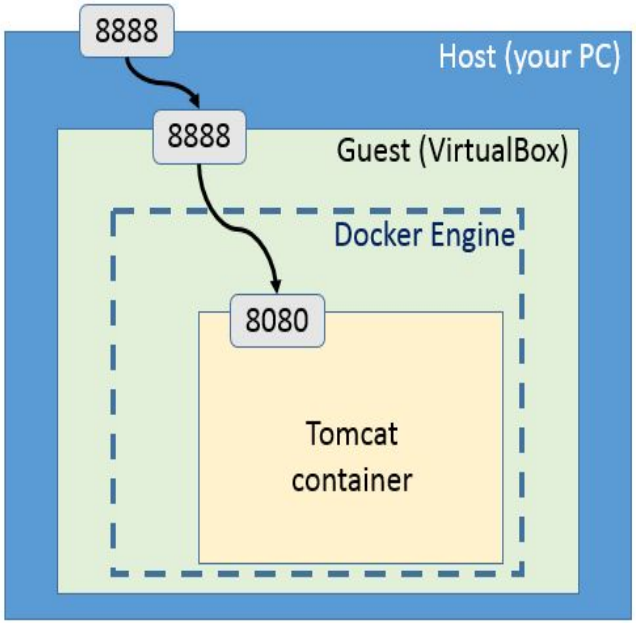
도커 컨테이너 다루기

- 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 컨테이너 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
  - 데이터베이스 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
```

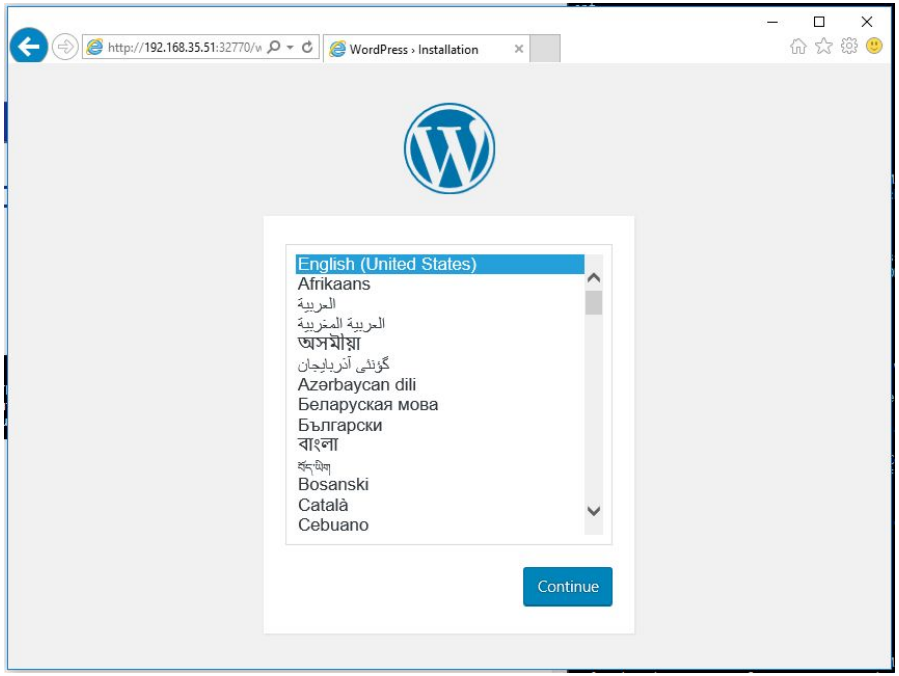
  - exec : 컨테이너 내부에서 명령어를 실행한 뒤 그 결과값을 반환

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

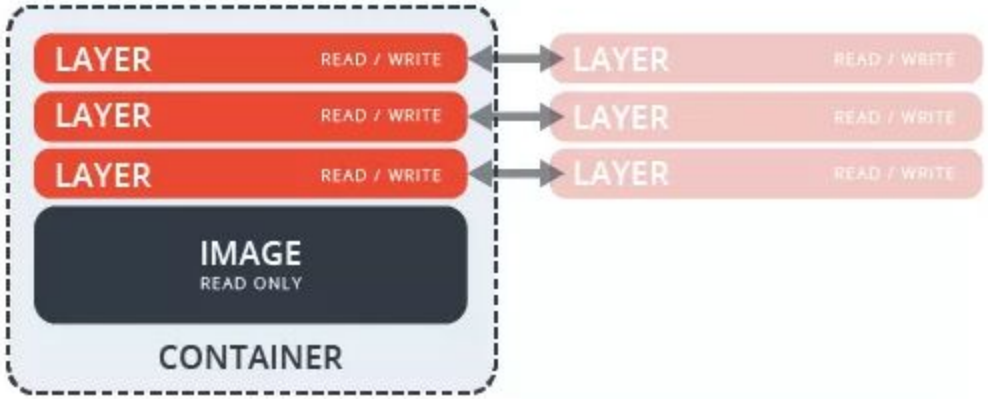


도커 볼륨

- 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

➢ 컨테이너의 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 생성된 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 --name datavol1 \
 -v /home/data1:/root/data1:rw \
 -v /home/data2:/root/data2:ro \
 ubuntu:14.04

# 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.

➢ 볼륨 리스트 확인

```
# docker volume ls
DRIVER          VOLUME NAME
local           df68dd10e59ddf5bb183ede2fab5e70bec90ef417aab703576a0cf72d18b5b39
local           f3ae91d2af119336e17cc1d25f908559bc90bf8767badf70defd96c7c5166a31
local           myvolume
```

➢ myvolume 볼륨 삭제

```
# docker volume rm myvolume
myvolume

# docker volume ls
DRIVER          VOLUME NAME
local           df68dd10e59ddf5bb183ede2fab5e70bec90ef417aab703576a0cf72d18b5b39
local           f3ae91d2af119336e17cc1d25f908559bc90bf8767badf70defd96c7c5166a31
```

도커 볼륨

- 볼륨 삭제
  - 사용하지 않은 볼륨 리스트는 한번에 삭제가능

➢ 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

도커 네트워크

- 네트워크 구조
  - 컨테이너 내부 IP는 순차적으로 할당
  - 컨테이너 생성, 재시작시 변경될 수 있음
  - 호스트에 veth 디바이스가 생성됨
  - 컨테이너 마다 외부 통신을 위해 호스트에 가상 네트워크 인터페이스(veth) 생성

➢ 컨테이너 네트워크 정보

```
root@d286b8157298:/#
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
..
```

➢ 호스트 네트워크 정보

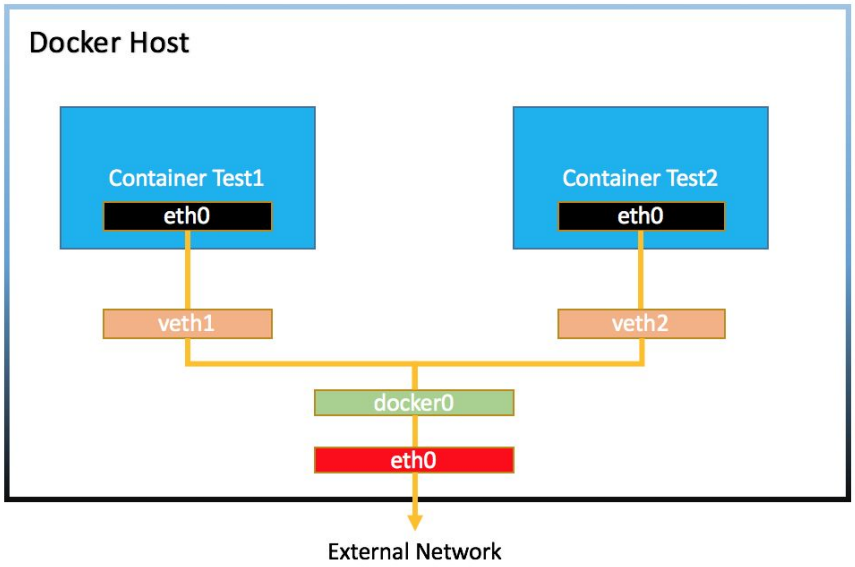
```
# ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
..
veth95fa667: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::dc85:72ff:fe5e:6285  prefixlen 64  scopeid 0x20<link>
..
```

도커 네트워크

- 네트워크 구조
  - 호스트 veth 디바이스는 docker0 브리지에 바인딩됨

➢ 호스트 docker0 브리지 정보

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.024252d9f225       no              veth95fa667
```



도커 네트워크

- 네트워크 기능
  - docker0 는 기본제공 브리지
  - 다양한 네트워크 드라이버 제공 (bridge, host, none, container, overlay)
  - 플러그인 및 솔루션 (weave, flannel, openvswitch)

➢ 호스트 도커 네트워크 리스트

```
# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c5a1645cca4e        bridge              bridge              local
1fcd84e8dc17        docker_gwbridge     bridge              local
6ed0fb9a6fff        host                host                local
61c80b79c426        none                null                local
```

➢ 호스트 도커 브리지 설정 정보

```
# docker inspect bridge
[
    {
        "Name": "bridge",

                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
..
```

도커 네트워크

- 브리지 생성하기
  - ➢ 호스트에 mybridge 네트워크 생성

```
# docker network create --driver bridge mybridge
23daa7904394dfccd836c56f1ccdc6ab0919e87d5e5fb84f4574148f384d4cae
```

  - ➢ mybridge 네트워크 사용 컨테이너 생성

```
# docker run -i -t --name mynetwork_container \
 --net mybridge \
 ubuntu:14.04
root@57dd1662ce3d:/#

root@57dd1662ce3d:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:13:00:02
          inet addr:172.19.0.2  Bcast:0.0.0.0  Mask:255.255.0.0
```

    - ○ 컨테이너에 새로운 ip 가 할당됨 확인

  - ➢ ip 할당을 임의로 설정한 네트워크 생성

```
# docker network create --driver bridge \
 --subnet=172.72.0.0/16 \
 --ip-range=172.72.0.0/24 \
 --gateway=172.72.0.1 \
 my_custom_network
a4709da563a5b2808d598b0adae5941ced4eadbb791502276ce59329f50212be
```

도커 네트워크

- 호스트(host) 네트워크
  - 컨테이너에 호스트 네트워크를 그대로 사용가능
  - 네트워크를 별도로 만들 필요 없음

➢ 호스트 네트워크를 사용한 network_host 컨테이너 생성

```
# docker run -i -t --name network_host \
 --net host \
 ubuntu:14.04
root@docker1:/#
```

  - 컨테이너 이름이 호스트 도메인명 과 동일함
  - 별도의 포트 연결 없이 컨테이너 애플리케이션 서비스 가능

- 논(none) 네트워크
  - 네트워크를 사용하지 않는 구조

➢ 호스트 네트워크를 사용한 network_host 컨테이너 생성

```
# docker run -i -t --name network_none \
 --net none \
 ubuntu:14.04
```

도커 네트워크

- 컨테이너(Container) 네트워크
  - 다른 컨테이너 네트워크를 공유
  - IP, MAC, NIC 의 속성을 공유함

➢ 컨테이너 네트워크를 사용한 network_container 컨테이너 생성

```
# docker run -i -t -d --name network_container_1 ubuntu:14.04
6a175d467e5a37e504c9a6b994efa89c451317bd917744488a3ce0f45a7613e8

# docker run -i -t -d --name network_container_2 \
 --net container:network_container_1 \
 ubuntu:14.04
bb0574edfb6c99c970caed8d0c78545b1863956254c55b830decb7111a8c77ef
```

  - --net container:[다른 컨테이너 이름/ID] : 컨테이너 네트워크로 연결
  - -i -t -d : 옵션을 함께 사용하면 내부 쉘을 실행 하지만 내부로 들어가지 않고 컨테이너도 종료 되지 않음, 테스트 용도의 컨테이너 생성시 유용

➢ container_1 과 container_2 네트워크 확인

```
# docker exec network_container_1 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0

# docker exec network_container_2 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
```

도커 네트워크

- 브리지 네트워크와 --net-alias
  - 동일한 호스트 이름으로 여러 개의 컨테이너에 접근 가능

➢ service 호스트 이름으로 연결가능한 3개 컨테이너 생성

```
# docker run -i -t -d --name network_alias_container1 \
 --net mybridge \
 --net-alias service ubuntu:14.04
8b1b201bc29d93c1a35f541647c87d2379749c0b1d3e3240a5f71129862e24f1

# docker run -i -t -d --name network_alias_container2 \
 --net mybridge \
 --net-alias service ubuntu:14.04
eb5596c7a23596280debfb5c32afb2871c3644914516458707ab862584fd4730

# docker run -i -t -d --name network_alias_container3 \
 --net mybridge \
 --net-alias service ubuntu:14.04
2832a48f2e3531d28cd5e37c3ee1dc3df870d760896c2a632c0ab7a080431d08
```

  - --net-alias [호스트 이름] : 컨테이너에 접근 가능한 호스트이름 설정

➢ network_alias_container1 컨테이너의 IP 확인

```
# docker inspect network_alias_container1 | grep IPAddress
        "SecondaryIPAddresses": null,
        "IPAddress": "",
            "IPAddress": "172.19.0.3"
```

도커 네트워크

- 브리지 네트워크와 --net-alias
  - ➢ alias_ping 컨테이너 생성 후 service 호스트 ping 테스트

```
# docker run -i -t --name network_alias_ping \
 --net mybridge \
 ubuntu:14.04

root@0c42c30fa6bb:/# ping -c 1 service
PING service (172.19.0.4) 56(84) bytes of data.
64 bytes from network_alias_container2.mybridge (172.19.0.4): icmp_seq=1 ttl=64 time=0.056 ms

root@0c42c30fa6bb:/# ping -c 1 service
PING service (172.19.0.5) 56(84) bytes of data.
64 bytes from network_alias_container3.mybridge (172.19.0.5): icmp_seq=1 ttl=64 time=0.059 ms

root@0c42c30fa6bb:/# ping -c 1 service
PING service (172.19.0.3) 56(84) bytes of data.
64 bytes from network_alias_container1.mybridge (172.19.0.3): icmp_seq=1 ttl=64 time=0.059 ms
```

    - ○ 도커 엔진 내장 DNS 가 --net-alias 옵션으로 service 이름에 대해 IP를 전달

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
```

컨테이너 자원

- 컨테이너 자원 할당 제한
  - 컨테이너 생성시 자원제한 값을 할당 하지 않으면 모든 자원을 제한 없이 사용
  - 자원 할당 제한으로 호스트의 자원을 관리해야 함

➢ 컨테이너 자원 제한 사용량 확인 방법

```
# docker inspect rsyslog_server
“HostConfig” : {
..
           "DiskQuota": 0,
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": false,
            "PidsLimit": 0,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0
..
```

  - Update 옵션을 이용 컨테이너 자원제한 변경 방법

    # docker update (변경할 자원 제한) (컨테이너 이름)

    ex) # docker update --cpuset-cpus=1 centos ubuntu

컨테이너 자원

- 컨테이너 메모리 제한
  - ➢ 메모리 제한 컨테이너 생성

```
# docker run -d \
 --memory="1g" \
 --name memeory_1g \
 nginx
```

    - --memory : 컨테이너 메모리 용량 제한 설정

  - ➢ 메모리 설정 값 확인

```
# docker inspect memory_1g | grep \"Memory\"
            "Memory": 1073741824,
```

  - ➢ 스왑 메모리값 제한 컨테이너 생성

```
# docker run -it --name swap_500m \
 --memory=200m \
 --memory-swap=500m \
 ubuntu:14.04
```

컨테이너 자원

- 컨테이너 CPU 제한
  - ➢ CPU 제한 컨테이너 생성

```
# docker run -i -t --name cpu_share \
 --cpu-shares 2048 \
 ubuntu:14.04
```

    - --cpu-shares : 상대적인 값을 지정 (1024=cpu 할당비율 1)
      2048 값은 일반 컨테이너 보다 CPU 할당 시간이 2배

  - ➢ stress 패키지 설치 및 cpu 부하 발생

```
# apt-get update
# apt-get install stress
# stress --cpu 1
```

  - ➢ 호스트의 cpu 사용량 확인

```
# ps aux | grep stress
..
root       3624 99.6  0.0   7316    96 pts/0    R+   21:03   3:12 stress --cpu 1
..
root       3753 44.4  0.0   7316   100 pts/0    R+   21:04   2:08 stress --cpu 1
```

컨테이너 자원

- 컨테이너 CPU 제한
  - ➢ 특정 CPU 코어 사용 제한

```
# docker run -i -t --name cpuset_2 \
 --cpuset-cpus=2 \
 ubuntu:14.04

root@46667cb2da51:/# stress --cpu 1
```

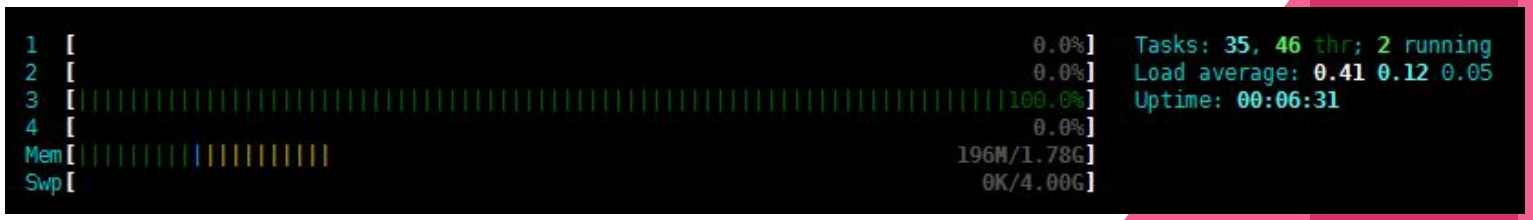
    - --cpuset-cpu : 컨테이너가 특정 CPU만 사용하도록 설정

  - ➢ htop 패키지 설치 (CPU 코어 사용 확인)

```
### Centos7 ###
# yum -y install epel-release
# yum -y install htop

### Ubuntu 16.04 ###
# apt-get install htop
```

  - ➢ htop 명령어 실행, 3번째 코어 CPU 사용량 확인



컨테이너 자원

- 컨테이너 CPU 제한
  - ➢ 특정 CPU 스케줄 시간 조절 (--cpu-period , --cpu-quota)

```
# docker run -i -t --name quota_1_4 \
 --cpu-period=100000 \
 --cpu-quota=25000 \
 ubuntu:14.04
```

    - CFS(Completely Fair Scheduler) 주기는 기본값 100ms =100000
    - 기본값 100000 중 25000 (¼) 를 할당
    - CPU 할당 스케쥴 시간이 ¼ 로 줄었기 때문에 성능도 ¼ 로 줄어듬

  - ➢ 직관적 CPU 사용량 설정 (--cpus)

```
# docker run -i -t --name cpus_container \
 --cpus=0.5 \
 ubuntu:14.04
```

    - --cpu-share=512 또는 --cpu-period=100000 --cpu-quota=50000 과 동일
    - 호스트 CPU 중 50% 사용

컨테이너 자원

- Block I/O 제한
  - 컨테이너가 파일을 읽고 쓰는 대역폭을 제한
  - Direct I/O 의 경우만 블록 입출력이 제한 됨, Buffered I/O 는 제한되지 않음

➢ 초당 쓰기 1mb 제한 컨테이너 생성

```
# docker run -i -t \
 --device-write-bps /dev/mapper/centos-root:1mb \
 ubuntu:14.04
root@4f1f9af26e72:/#
```

  - --device-write-bps, --device-read-bps : 쓰고 읽는 작업의 초당 제한 설정
  - --device-write-iops, --device-read-iops : 상대값을 이용한 쓰고 읽는 작업 속도 제한
  - kb, mb, gb 단위로 제한 가능

➢ 블록 쓰기 테스트

```
root@4f1f9af26e72:/# dd if=/dev/zero of=test.out bs=1M count=10 oflag=direct
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 10.009 s, 1.0 MB/s
```

컨테이너 자원

- Block I/O 제한
  - ➢ 5mb 제한 컨테이너 에서 블록 쓰기 테스트

```
docker run -i -t --device-write-bps /dev/mapper/centos-root:5mb ubuntu:14.04
root@ba1391a7498c:/# dd if=/dev/zero of=test.out bs=1M count=10 oflag=direct
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 2.00846 s, 5.2 MB/s
```

  - ➢ iops 옵션을 통한 블록 쓰기 제한 컨테이너 생성

```
# docker run -i -t --device-write-iops /dev/mapper/centos-root:5 ubuntu:14.04
root@80ad9f23ba27:/# dd if=/dev/zero of=test.out bs=1M count=10 oflag=direct
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 4.00267 s, 2.6 MB/s

# docker run -i -t --device-write-iops /dev/mapper/centos-root:10 ubuntu:14.04
root@4d1eb087a06b:/# dd if=/dev/zero of=test.out bs=1M count=10 oflag=direct
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 2.00278 s, 5.2 MB/s
```

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

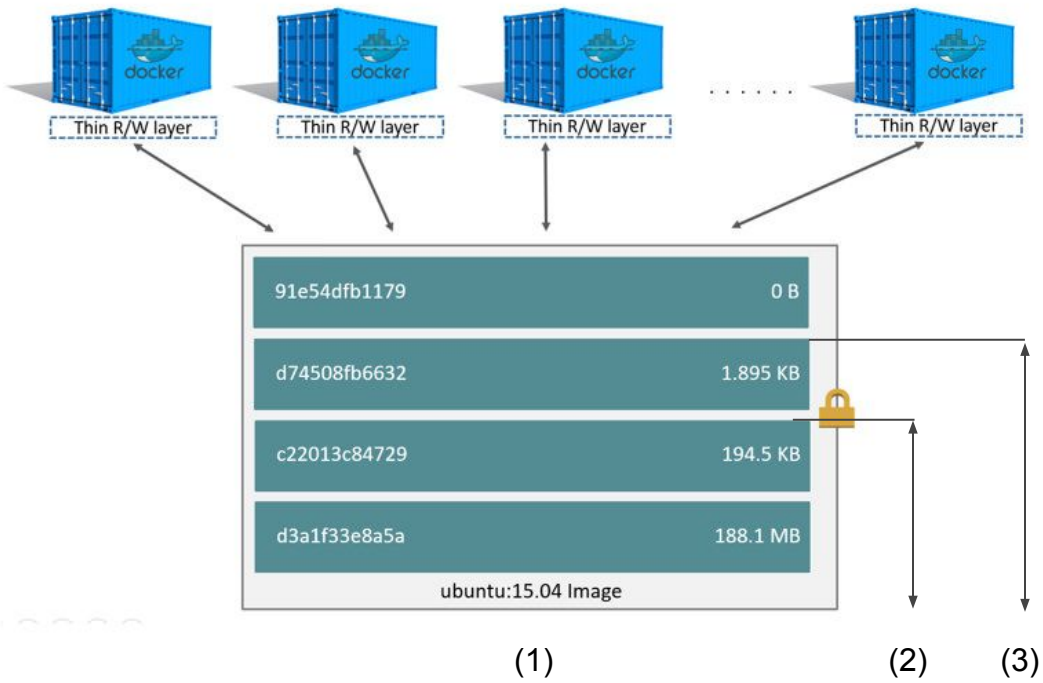
➢ 도커 허브 이미지 검색

```
# 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 즐겨찾기 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 메시지 입력, 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
```

    - 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

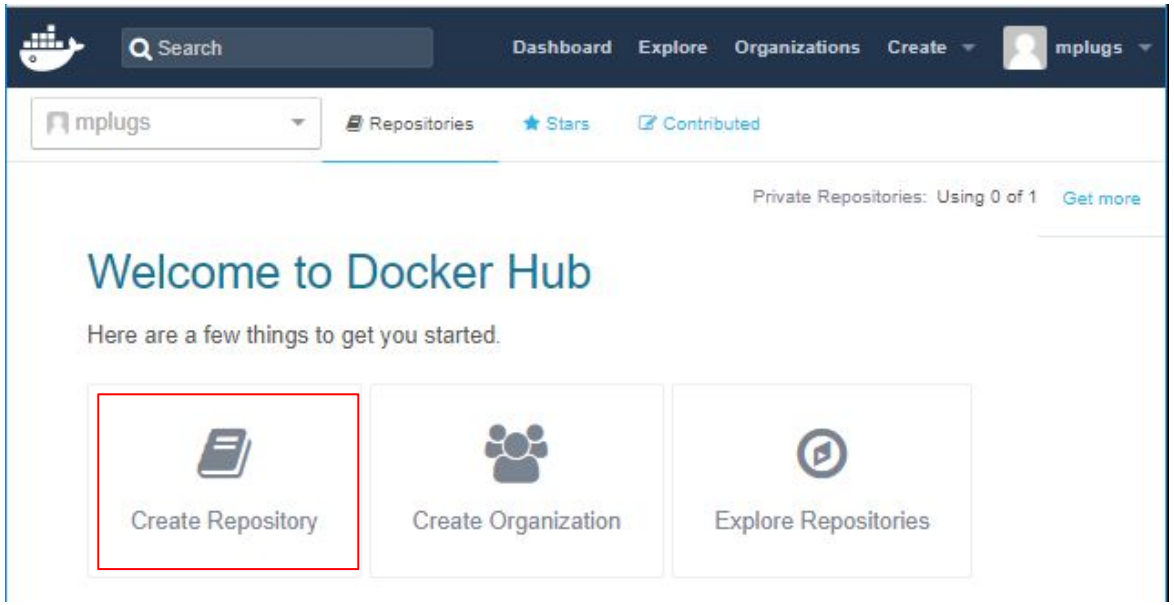
도커 이미지

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

  - Sign up 클릭 후 계정생성

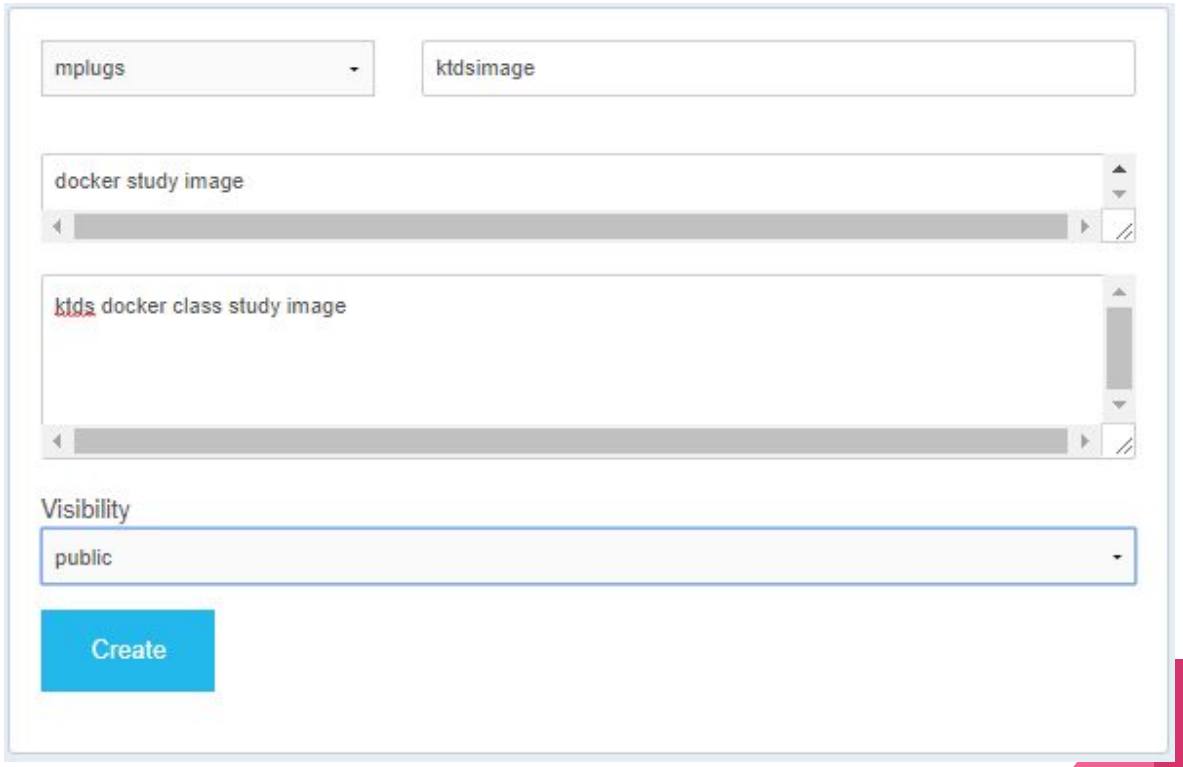
도커 이미지

- 이미지 저장소 생성
  - Create Repository 클릭



도커 이미지

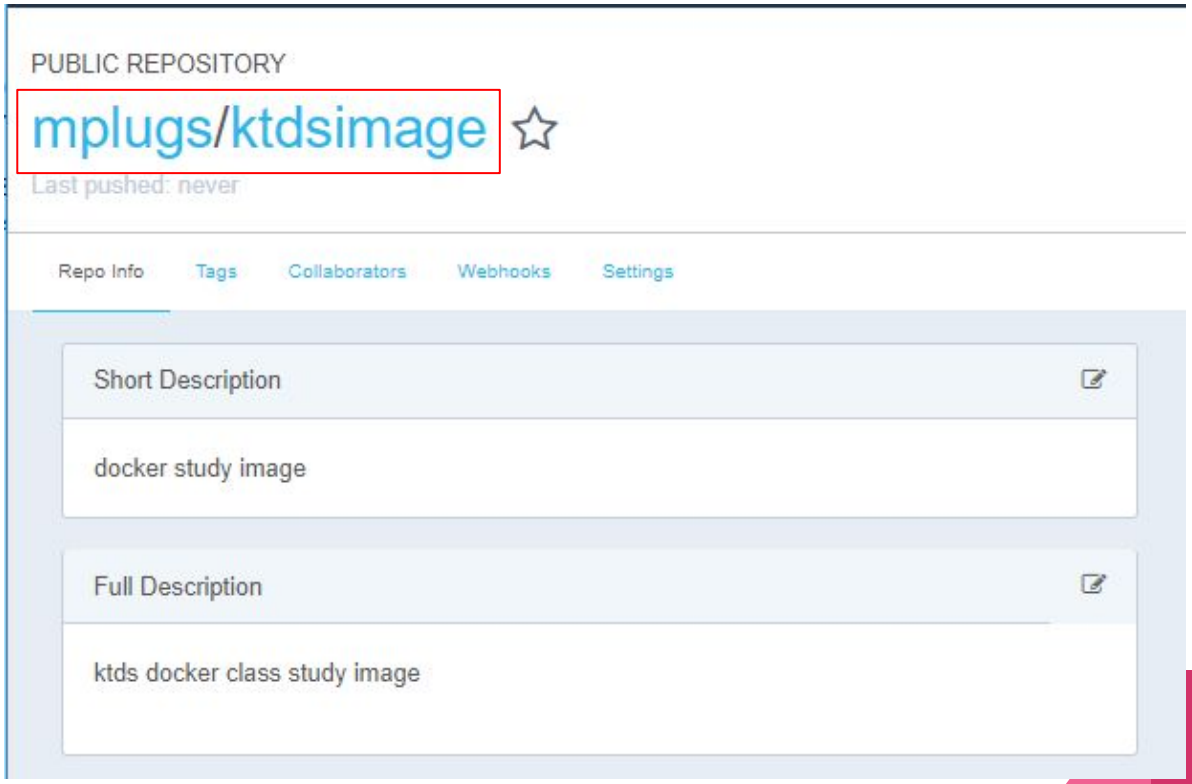
- 이미지 저장소 생성
  - 이미지 저장소 정보 입력



  - 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

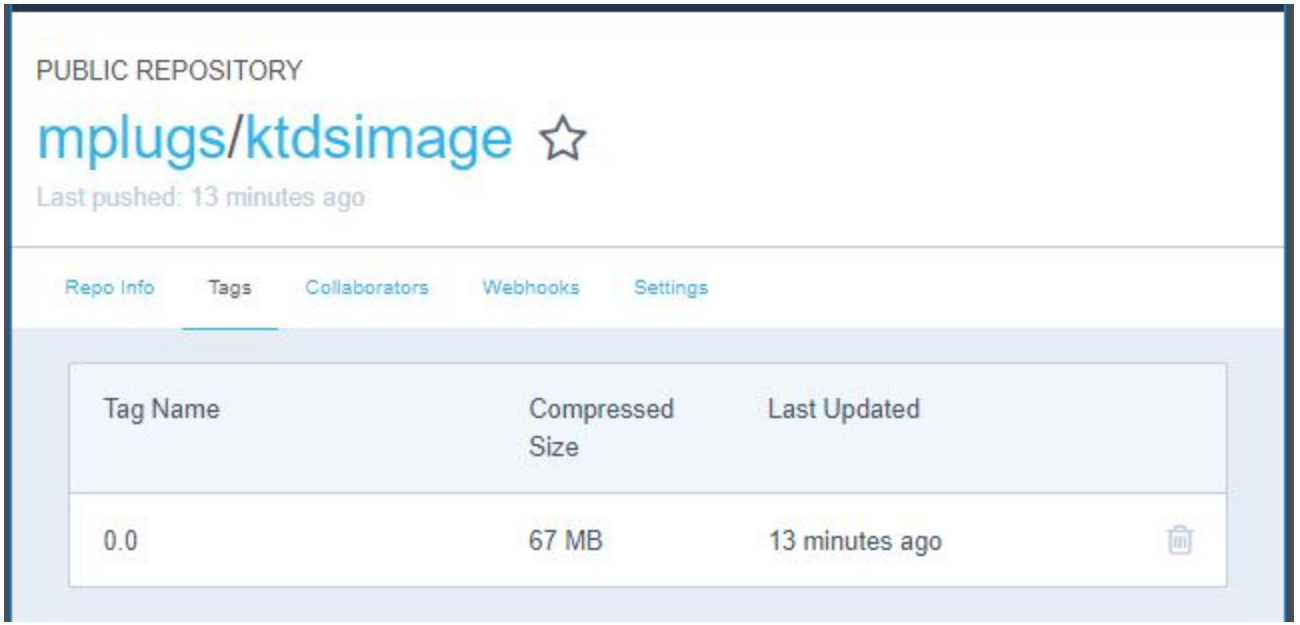
  - ➢ 이미지에 이름 추가 하기

```
# 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

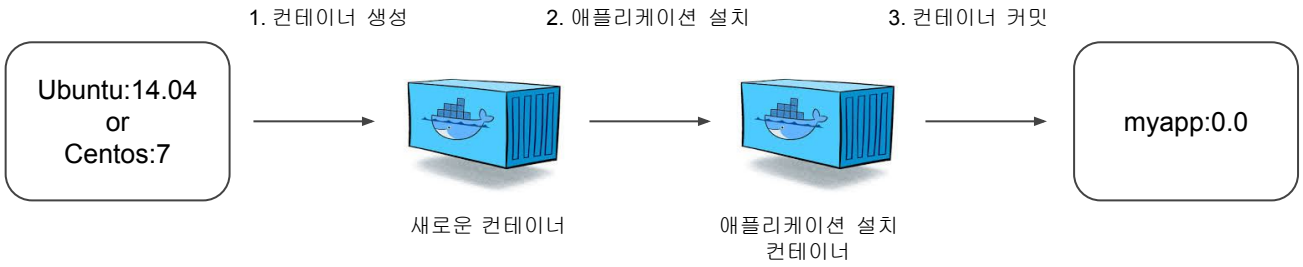
```
# vi /usr/lib/systemd/system/docker.service
..
ExecStart=/usr/bin/dockerd $DOCKER_OPTS
..
# DOCKER_OPTS="--insecure-registry=192.168.35.51:5000"
```

  - ➢ 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

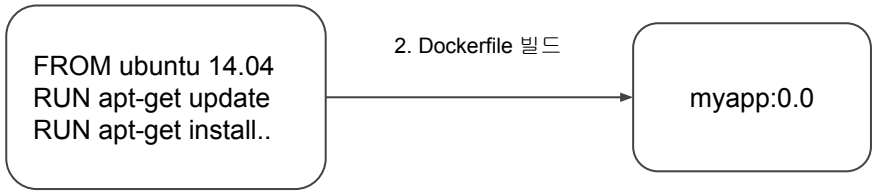
- 컨테이너로 이미지 생성 방법
  1. 기본 OS 이미지로 컨테이너 생성
  2. 애플리케이션 설치 및 환경설정, 소스코드 복제
  3. 컨테이너 이미지 커밋(commit)



  - 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

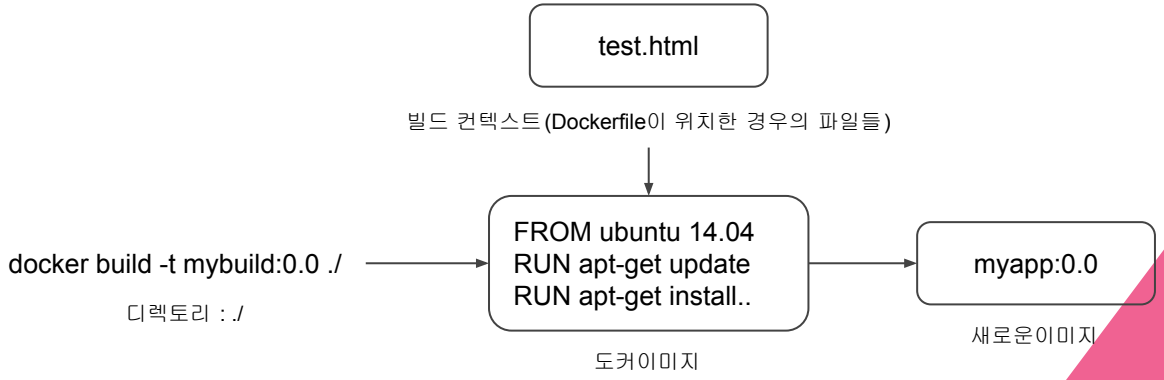
- Dockerfile 빌드
  - ➢ 생성된 이미지로 컨테이너 실행

```
# 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?
```

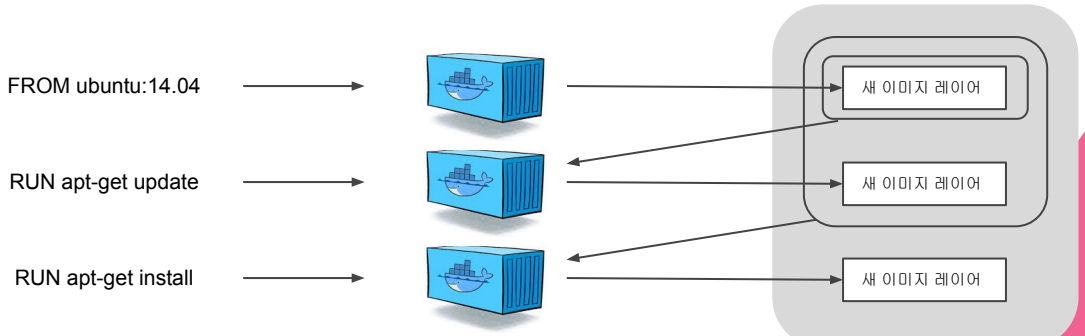
    - ○ ! : 컨텍스트 제외 하지 않을 파일 지정

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 사용될 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 역할**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

3장 도커 스웜

- 도커 스웜 과 도커 스웜 모드
- 도커 스웜 설치 와 노드관리
- 도커 스웜 서비스
- 도커 스웜 네트워크
- 도커 스웜 볼륨

도커 스웜

- 도커 스웜 사용 이유
  - 서버 자원이 부족한 경우 서버 자원 확장 필요
  - Scale out 확장을 위한 병렬 클러스터 구성 필요
  - 여러 대의 서버를 하나의 풀로 관리 가능
  - 새로운 서버를 추가됐을 때 발견 하고 관리
  - 컨테이너를 서버에 할당 할 때 스케줄러와 로드밸런싱 문제 해결
  - 대표적인 도커 제공 클러스터 엔진

도커 스웜 모드 와 도커 스웜

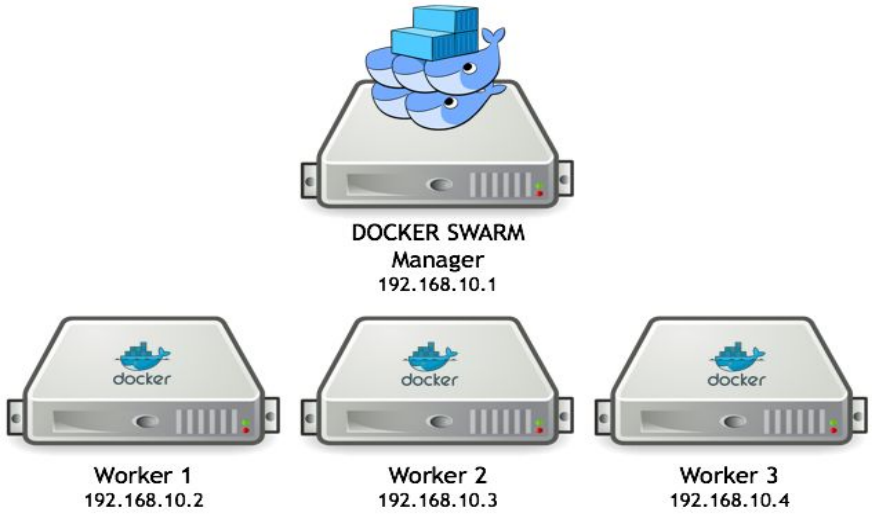
- 도커 스웜 모드
  - 도커 버전 1.12 버전 이후 부터 사용 가능
  - 스웜 에이전트가 도커 자체에 내장
  - 분산 코디네이터 설치 필요 없음

- 도커 스웜
  - 도커 버전 1.6 이후 부터 사용 가능
  - 스웜 에이전트가 컨테이너로서 별도로 존재
  - 분산 코디네이터를 외부에 별도로 구성 필요

✓ 도커 클러스터를 구성하 위한 필수 도구
분산 코디네이터 : 각종 정보를 저장하고 동기화
클러스터 매니저 : 클러스터 내 서버 관리
에이전트 : 클러스터 내 서버 제어

도커 스웜 모드

- 도커 스웜 모드 특징
  - 웹서비스 컨테이너를 다루기위한 클러스터링 구성에 적합
  - 매니저 노드와 워커 노드로 구성
  - 매니저 노드는 1개 이상 , 전체 노드는 홀수 개수로 구성 권장
  - 워커 노드가 없어도 매니저노드가 워커 노드 역활 포함
  - 매니저 노드의 절반 이상에 장애 발생시 복구될 때 까지 클러스터의 운영 중단



도커 스웜 모드 설치

- 스웜 모드 설치
  - ➢ 도커 버전 및 데몬 상태 확인
    - centos7 도커 버전 확인

```
# docker -v
Docker version 18.03.1-ce, build 9ee9f40


# docker info | grep -i swarm
Swarm: inactive
WARNING: devicemapper: usage of loopback devices is strongly discouraged for production use.
         Use `--storage-opt dm.thinpooldev` to specify a custom block storage device.
```

    - ubuntu 16.04 도커 버전 확인

```
# docker -v
Docker version 1.13.1, build 092cba3

# docker info | grep -i swarm
WARNING: No swap limit support
Swarm: inactive
```

도커 스웜 모드 설치

- 스웜 모드 설치
  - ➢ 클러스터 구성을 위한 3대의 도커 서버 준비
    - 호스트 정보

```
swarm-manager 192.168.35.100
swarm-worker1 192.168.35.101
swarm-worker2 192.168.35.102
```

  - ➢ 매니저 역할 서버에서 스웜 모드 클러스터 시작

```
root@swarm-manager:~# docker swarm init --advertise-addr 192.168.35.100
Swarm initialized: current node (plwgqgyn9vaw2p41vjo86lmnm) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-ebbfmu36un4267lqfkxecsjlw \
    192.168.35.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

    - --advertise-addr : 워커 노드가 매니저 노드에 접근 가능한 IP 지정

도커 스웜 모드 설치

- 스웜 모드 설치
  - ➢ 워커 노드 추가하기

```
root@swarm-worker1:~# docker swarm join \
   --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-ebbfmu36un4267lqfkxecsjlw \
   192.168.35.100:2377
This node joined a swarm as a worker.

root@swarm-worker2:~# docker swarm join \
   --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-ebbfmu36un4267lqfkxecsjlw \
   192.168.35.100:2377
This node joined a swarm as a worker.
```

    - --token : 새로운 워커 노드를 클러스터에 추가할때 사용될 인증키

  - ➢ 워커 노드 추가 확인

```
root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
ric1vaxsnfb1i8oq18vskqdnb     swarm-worker1  Ready   Active
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
```

    - * : 현재 명령을 실행한 서버를 표시

도커 스웜 모드 설치

- 스웜 모드 설치
  - ➢ 워커 노드 추가하기

```
root@swarm-worker1:~# docker swarm join \
   --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-ebbfmu36un4267lqfkxecsjlw \
   192.168.35.100:2377
This node joined a swarm as a worker.

root@swarm-worker2:~# docker swarm join \
   --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-ebbfmu36un4267lqfkxecsjlw \
   192.168.35.100:2377
This node joined a swarm as a worker.
```

    - --token : 새로운 워커 노드를 클러스터에 추가할때 사용될 인증키

  - ➢ 워커 노드 추가 확인

```
root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
ric1vaxsnfb1i8oq18vskqdnb     swarm-worker1  Ready   Active
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
```

도커 스웜 모드 설치

- 매니저 토큰 관리
  - ➢ 매니저 노드 추가를 위한 토큰 확인

```
root@swarm-manager:~# docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-c38ywqyd72polcwmck8phonfs \
    192.168.35.100:2377
```

    - --token : 새로운 워커 노드를 클러스터에 추가할때 사용될 인증키

  - ➢ 매니저 토큰 갱신

```
root@swarm-manager:~# docker swarm join-token --rotate manager
Successfully rotated manager join token.

To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2is2v64o5qu6zsofauhv356lffyo73yni0v1gt6rle5slbxjkt-aserm7oh4qlsavwwpbbff41op \
    192.168.35.100:2377
```

도커 스웜 모드 설치

- 스웜 클러스터 정보 확인
  - ➢ docker info 명령을 이용한 스웜 클러스터 정보 확인

```
# docker info
..
Images: 0
Server Version: 1.13.1
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: active
 NodeID: plwgqgyn9vaw2p41vjo86lmnm
 Is Manager: true
 ClusterID: wtr0zbtv98fnr4uhao6zbi8in
 Managers: 1
 Nodes: 3
 Orchestration:
  Task History Retention Limit: 5
..
```

도커 스웜 노드 관리

- 워크 노드 삭제
  - ➢ 워크 노드 스웜 모드 해제

```
root@swarm-worker1:~# docker swarm leave
Node left the swarm.
```

    - 삭제 할 워커 노드에서 docker swarm leave 명령 실행

  - ➢ 클러스터 정보 확인

```
root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
ric1vaxsnfb1i8oq18vskqdnb     swarm-worker1  Down    Active
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
```

    - 스웜 모드 해제 워커 노드의 상태 Down 확인

  - ➢ 워커 노드 삭제

```
root@swarm-manager:~# docker node rm swarm-worker1
swarm-worker1

root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
```

도커 스웜 노드 관리

- 워커 노드를 매니저 노드로 변경

```
root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Active

root@swarm-manager:~# docker node promote swarm-worker1

root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Active        Reachable
```

- 매니저 노드 를 워커 노드로 변경

```
root@swarm-manager:~# docker node demote swarm-worker1
Manager swarm-worker1 demoted in the swarm.
root@swarm-manager:~# docker node promote swarm-worker1

root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Active
```

도커 스웜 모드 서비스

- 스웜 모드 서비스 개념
  - 도커 명령어의 제어 단위는 컨테이너
  - 스웜 모드의 제어 단위는 서비스 (Service)
  - 서비스는 1개 이상의 컨테이너 집합
  - 서비스 내의 컨테이너를 태스크 (Task) 라고 명칭
  - 컨테이너들은 워커 노드와 매니저 노드에 할당됨

  - 서비스를 생성할때 컨테이너를 3개로 설정했다고 가정
  - 함께 생성된 컨테이너를 리플리카(replica) 라고 함

도커 스웜 모드 서비스

- 컨테이너 리플리카 (replica)
  - 노드 다운 발생시 서비스에 정의된 리플리카의 수만큼 컨테이너 생성
  - 컨테이너 중 일부가 작동을 멈춰 정지한 경우 리플리카 수만큼 컨테이너 생성
  - 스웜 매니저는 클러스터에서 컨테이너를 감시, 리플리카 수만큼 컨테이너 생성

- 롤링 업데이트 (Rolling Update)
  - 서비스 내 컨테이너 이미지를 일괄적으로 업데이트 할 경우
  - 컨테이너를 순차적으로 새로운 이미지로 재생성, 서비스 다운타임 없이 업데이트

도커 스웜 서비스

- 스웜 서비스 생성
  - ➢ 서비스 생성

```
root@swarm-manager:~# docker service create \
 ubuntu:14.04 \
 /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

  - ➢ 서비스 리스트 확인

```
root@swarm-manager:~# docker service ls
ID            NAME           MODE        REPLICAS  IMAGE
xorlrsv98v14  pensive_kirch  replicated  1/1       ubuntu:14.04
```

  - ➢ 서비스 상세 정보 확인

```
root@swarm-manager:~# docker service ps pensive_kirch
ID            NAME             IMAGE         NODE           DESIRED STATE  CURRENT STATE          ERROR  PORTS
x4svxz6x1k2l  pensive_kirch.1  ubuntu:14.04  swarm-manager  Running        Running 2 minutes ago
```

    - 컨테이너 목록, 상태, 컨테이너가 할당된 노드 위치 확인 가능

도커 스웜 서비스

- 리플리카 정의 서비스 생성
  - ➢ nginx 웹 서버 서비스 생성

```
root@swarm-manager:~# docker service create --name myweb \
 --replicas 2 \
 -p 80:80 \
 nginx
qyko35yja1loo8sdigwms3x1t
```

    - --replicas : 컨테이너 리플리카 셋 설정
    - -p : 컨테이너 포트 와 각 노드의 포트 연결

  - ➢ 서비스 상세 정보 확인

```
root@swarm-manager:~# docker service ps myweb
ID            NAME     IMAGE          NODE           DESIRED STATE  CURRENT STATE           ERROR  PORTS
q1uaoc0mpa8b  myweb.1  nginx:latest   swarm-worker1  Running        Running 29 seconds ago
zpsn2obnjcmj  myweb.2  nginx:latest   swarm-worker2  Running        Running 28 seconds ago
```

도커 스웜 서비스

- 리플리카 설정 변경
  - ➢ nginx 웹 서버 서비스 scale up

```
root@swarm-manager:~# docker service scale myweb=4
myweb scaled to 4

root@swarm-manager:~# docker service ps myweb
ID            NAME     IMAGE         NODE           DESIRED STATE  CURRENT STATE           ERROR
PORTS
q1uaoc0mpa8b  myweb.1  nginx:latest  swarm-worker1  Running        Running 6 minutes ago
zpsn2obnjcmj  myweb.2  nginx:latest  swarm-worker2  Running        Running 6 minutes ago
om3y71jvwwvz  myweb.3  nginx:latest  swarm-manager  Running        Running 2 seconds ago
mnk912vq1w1k  myweb.4  nginx:latest  swarm-worker2  Running        Running 12 seconds ago
```

    - 리플리카 요청 개수에 맞게 컨테이너 생성
    - 노드의 포트는 컨테이너 포트에 라운드 로빈 (round-robin) 방식으로 연결

도커 스웜 서비스

- 글로벌 서비스 생성
  - 스웜 클러스터 내에서 사용할 수 있는 모든 노드에 컨테이너를 하나씩 생성
  - 스웜 클러스터를 모니터링하기 위한 에이전트 컨테이너 생성시 유용

➢ nginx 웹 서버 서비스 생성

```
root@swarm-manager:~# docker service create --name global_web \
 --mode global \
 nginx
0uow6m8dbatvn2fcqn870arhe

root@swarm-manager:~# docker service ls
ID            NAME          MODE        REPLICAS  IMAGE
0uow6m8dbatv  global_web    global      3/3       nginx:latest
qyko35yja1lo  myweb         replicated  4/4       nginx:latest
xorlrsv98v14  pensive_kirch  replicated  1/1       ubuntu:14.04
```

➢ 서비스 상세 정보 확인

```
root@swarm-manager:~# docker service ps global_web
ID            NAME                                    IMAGE         NODE           DESIRED STATE  CURRENT STATE
ERROR  PORTS
n5vdyeekpci2  global_web.plwgqgyn9vaw2p41vjo86lmnm  nginx:latest  swarm-manager  Running        Running 22 seconds ago
g97hyaab1ufj  global_web.vz7prsi133nzubvy65wydtxzf  nginx:latest  swarm-worker2  Running        Running 22 seconds ago
tpvwsqf230y8  global_web.ywnkgswoukp0mjis1wwuy1jv2  nginx:latest  swarm-worker1  Running        Running 22 seconds ago
```

도커 스웜 서비스

- 스웜 모드 장애 복구
  - 리플리카 정의로 생성된 컨테이너가 정지되거나, 노드가 다운되면 컨테이너 자동 복구
- 컨테이너 정지 발생시
  - ➢ 리플리카로 생성된 컨테이너 강제 삭제

```
root@swarm-manager:~# docker rm -f myweb.3.om3y71jvwwvzj989odtkxa3xo
myweb.3.om3y71jvwwvzj989odtkxa3xo
```

  - ➢ 중지된 컨테이너와 새로 생성된 컨테이너 확인

```
root@swarm-manager:~# docker service ps myweb
ID             NAME          IMAGE          NODE           DESIRED STATE  CURRENT STATE           ERROR
PORTS
q1uaoc0mpa8b   myweb.1       nginx:latest   swarm-worker1  Running        Running 8 hours ago
zpsn2obnjcmj   myweb.2       nginx:latest   swarm-worker2  Running        Running 8 hours ago
7b4g1aiha0gr   myweb.3       nginx:latest   swarm-manager  Running        Running 37 seconds ago
om3y71jvwwvz    \_ myweb.3   nginx:latest   swarm-manager  Shutdown       Failed 42 seconds ago   "task: non-zero exit
(137)"
mnk912vq1w1k   myweb.4       nginx:latest   swarm-worker2  Running        Running 8 hours ago
```

도커 스웜 서비스

- 특정 노드 다운 발생시
  - ➢ 워커 노드 도커 데몬 중지

```
root@swarm-worker1:~# service docker stop
```

  - ➢ 워커 노드 상태 확인

```
root@swarm-manager:~# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Down    Active
```

  - ➢ 복구된 컨테이너 확인

```
root@swarm-manager:~# docker service ps myweb
ID            NAME         IMAGE         NODE          DESIRED STATE  CURRENT STATE           ERROR
PORTS
h1kceu12nf8v  myweb.1      nginx:latest  swarm-manager  Running        Running 2 minutes ago
q1uaoc0mpa8b   \_ myweb.1  nginx:latest  swarm-worker1  Shutdown       Running 9 hours ago
zpsn2obnjcmj  myweb.2      nginx:latest  swarm-worker2  Running        Running 9 hours ago
..
```

도커 스웜 서비스

- 리플리카 와 글로벌 서비스 차이
  - 노드가 복구되면 글로벌 서비스는 원래 노드에 컨테이너가 복구됨
  - 리플리카 서비스 컨테이너는 scale 명령으로 컨테이너 수를 줄이고, 다시 늘려야 함

➢ 노드 복구후 글로벌 서비스 확인

```
root@swarm-manager:~# docker service ps global_web
ID            NAME                                    IMAGE         NODE            DESIRED STATE  CURRENT STATE
ERROR                              PORTS
o6dr0sr7rs9e  global_web.ywnkgswoukp0mjis1wwuy1jv2  nginx:latest  swarm-worker1  Running        Running about a minute
ago
n5vdyeekpci2  global_web.plwgqgyn9vaw2p41vjo86lmnm  nginx:latest  swarm-manager  Running        Running 8 hours ago
g97hyaab1ufj  global_web.vz7prsi133nzubvy65wydtxzf  nginx:latest  swarm-worker2  Running        Running 8 hours ago
tpvwsqf230y8  global_web.ywnkgswoukp0mjis1wwuy1jv2  nginx:latest  swarm-worker1  Shutdown       Failed about a minute
ago   "No such container: global_web..."
```

➢ 리플리카 컨테이너 재균등 생성

```
root@swarm-manager:~# docker service scale myweb=1
myweb scaled to 1
root@swarm-manager:~# docker service scale myweb=4
myweb scaled to 4
```

도커 스웜 서비스

- 서비스 롤링 업데이트
  - 컨테이너 이미지의 순차적 업데이트 적용
  - 서비스 중단 없는 업데이트 가능

➢ 롤링 업데이트 테스트를 위한 서비스 생성

```
root@swarm-manager:~# docker service create --name myweb2 \
 --replicas 3 \
 nginx:1.10
xftzwtswuu9i3isnnwk001b3h
```

➢ 이미지 업데이트 실행

```
root@swarm-manager:~# docker service update \
 --image nginx:1.11 \
 myweb2
myweb2
```

➢ 서비스 확인

```
root@swarm-manager:~# docker service ps myweb2
ID            NAME          IMAGE       NODE           DESIRED STATE  CURRENT STATE            ERROR
PORTS
x6juwkzrx2fm  myweb2.1      nginx:1.11  swarm-manager  Running        Preparing 3 seconds ago
1q3vrmpqxfd5   \_ myweb2.1  nginx:1.10  swarm-manager  Shutdown       Shutdown 2 seconds ago
..
```

도커 스웜 서비스

- 서비스 롤링 업데이트
  - 서비스 생성시 업데이트를 위한 정책 설정 가능
    업데이트 컨테이너수, 업데이트 딜레이 시간, 업데이트 실패시 동작

➢ 롤링업데이트 정책 설정

```
root@swarm-manager:~# docker service create \
 --replicas 4 \
 --name myweb3 \
 --update-delay 10s \
 --update-parallelism 2 \
 --update-failure-action continue \
 nginx:1.10
tvib4nxegyimqa9r90mq6cq36
```

  - --update-delay : 업데이트 적용 딜레이 시간
  - --update-parallelism : 동시 업데이트 할 컨테이너 수
  - --update-failure-action : 업데이트 실패시 계속 진행 여부, 기본값 pause

도커 스웜 서비스

- 서비스 롤링 업데이트
  - ➢ 롤링 업데이트 정책 확인

```
root@swarm-manager:~# docker service inspect --pretty myweb3

ID:          lew84tiw8g54f1jkbyenxqj6u
Name:        myweb3
Service Mode:Replicated
 Replicas:   4
Placement:
UpdateConfig:
 Parallelism:2
 Delay:            10s
 On failure: continue
 Max failure ratio: 0
ContainerSpec:
 Image:
nginx:1.10@sha256:6202beb06ea61f44179e02ca965e8e13b961d12640101fca213efbfd145d7575
Resources:
Endpoint Mode:     vip
```

도커 스웜 네트워크

- 도커 스웜 네트워크 구조
  - 컨테이너를 여러노드에 분산 할당하기 때문에 네트워크가 하나로 묶인 네트워크 풀 필요
  - 서비스를 외부로 노출했을때 어느 노드의 서비스로든 연결 가능한 라우팅 기능 필요
  - 스웜 모드 자체적으로 지원 하는 네트워크 드라이버 사용
- 네트워크 리스트 확인

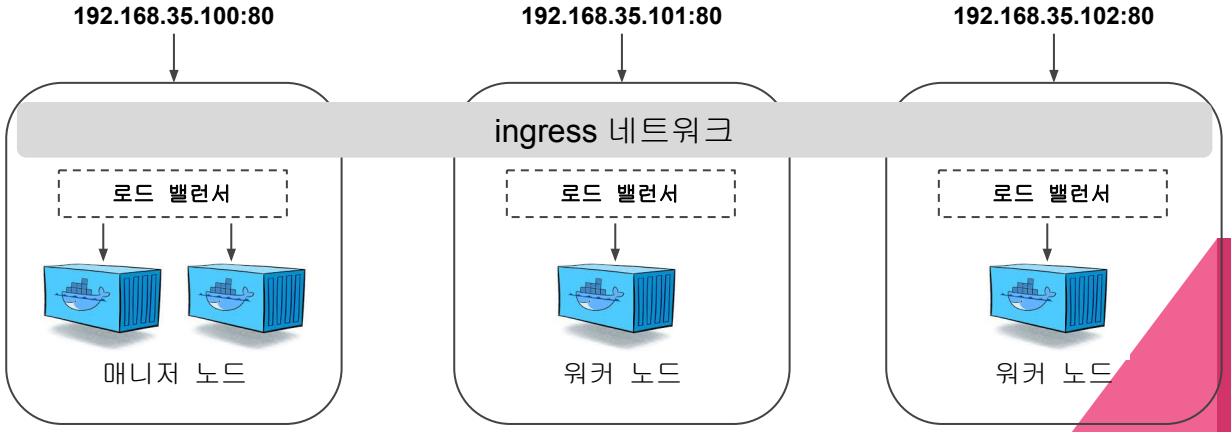
```
root@swarm-manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
052cdc72eaf0        bridge              bridge              local
f8d483f685c6        docker_gwbridge     bridge              local
ff725c832972        host                host                local
kueyo6017adt        ingress             overlay             swarm
791b469cfde4        none                null                local
```

  - docker_gwbridge : 스웜에서 오버레이 (overlay) 네트워크 필요시 사용
  - ingress : 로드밸런싱 과 라우팅 메시 (Routing Mesh) 에 사용

도커 스웜 네트워크

- ingress 네트워크
  - 스웜 클러스터 생성시 자동 등록 되는 네트워크
  - 스웜 모드 사용시만 유효, SCOPE 항목이 swarm 으로 설정됨
  - 매니저 노드 뿐 아니라 스웜 클러스터에 등록된 노드 전부 ingress 네트워크 생성됨
- ingress 네트워크 확인

```
root@swarm-manager:~# docker network ls | grep ingress
kueyo6017adt        ingress             overlay             swarm
```



ingress 네트워크 구조

도커 스웜 네트워크

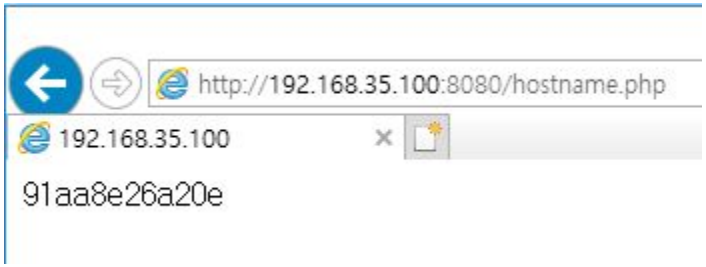
- ingress 네트워크 서비스 생성

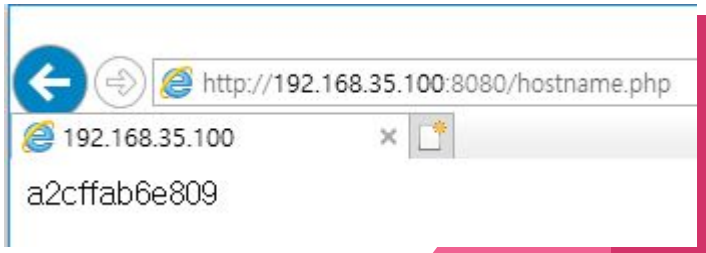
```
root@swarm-manager:~# docker service create --name hostname \
 -p 8080:80 \
 --replicas=4 \
 ubuntu:apache-php
iw57u71wmi9shpeyym96y5s1e
```

- ingress 네트워크 서비스 리스트 확인

```
root@swarm-manager:~# docker service ps hostname
ID            NAME        IMAGE              NODE           DESIRED STATE  CURRENT STATE           ERROR
PORTS
xzjir7324wb1  hostname.1  ubuntu:apache-php  swarm-manager  Running        Running 11 seconds ago
wmx8tj6ncf8s  hostname.2  ubuntu:apache-php  swarm-worker1  Running        Running 14 seconds ago
annfweg7lkrf  hostname.3  ubuntu:apache-php  swarm-worker2  Running        Running 11 seconds ago
6dbcesrgaqjn  hostname.4  ubuntu:apache-php  swarm-manager  Running        Running 11 seconds ago
```

- 노드 ip:port 를 통한 접속 확인





도커 스웜 네트워크

- 컨테이너 Ingress 네트워크
  - ➢ 컨테이너 Ingress 네트워크 확인

```
root@swarm-manager:~# docker ps --format "table {{.ID}}\t{{.Status}}\t{{.Image}}"
CONTAINER ID        STATUS              IMAGE
c3a4b06998da        Up 28 minutes       ubuntu:apache-php
a2cffab6e809        Up 28 minutes       ubuntu:apache-php
d593ba523ae3        Up 28 minutes       ubuntu:apache-php

root@swarm-manager:~# docker exec c3a4b06998da ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0a:ff:00:0d
          inet addr:10.255.0.13  Bcast:0.0.0.0  Mask:255.255.0.0
..

eth1      Link encap:Ethernet  HWaddr 02:42:ac:12:00:05
          inet addr:172.18.0.5  Bcast:0.0.0.0  Mask:255.255.0.0
..

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

  - eth0 : ingress 네트워크
  - eth1 : 컨테이너 네트워크

도커 스웜 네트워크

- 오버레이 네트워크
  - ➢ 사용자 정의 오버레이 네트워크 생성

```
root@swarm-manager:~# docker network create \
 --subnet 10.0.9.0/24 \
 -d overlay \
 myoverlay
kvjm298iffxz68ydadzb4xnoo
```

    - -d : 네트워크 드라이버 지정, overlay 네트워크 드라이버로 생성
  - ➢ 생성된 오버레이 네트워크 확인

```
root@swarm-manager:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
..
kvjm298iffxz        myoverlay           overlay             swarm
..
```

  - ➢ Docker run --net 사용가능 오버레이 네트워크 생성

```
root@swarm-manager:~# docker network create -d overlay \
 --attachable \
 myoverlay2
1bkkpayv9asm0qy30g0cp9ozb
```

    - --attachable : docker run --net 명령으로 이용가능한 overlay 네트워크 생성옵션

도커 스웜 네트워크

- 오버레이 네트워크
  - ➢ 오버레이 네트워크 사용 컨테이너 생성

```
root@swarm-manager:~# docker run -it \
 --net myoverlay2 ubuntu:apache-php
a9cc2d7be876b304c02cee4d46f9421518935ae449c6c671fe4cbc6e63c0774b
```

  - ➢ 컨테이너 네트워크 정보 확인

```
root@swarm-manager:~# docker exec a9cc2d7be876 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0a:00:00:02
          inet addr:10.0.0.2  Bcast:0.0.0.0  Mask:255.255.255.0
..

eth1      Link encap:Ethernet  HWaddr 02:42:ac:12:00:0a
          inet addr:172.18.0.10  Bcast:0.0.0.0  Mask:255.255.0.0
..

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

    - eth0 : myoverlay2 오버레이 네트워크
    - eth1 : 컨테이너 네트워크

도커 스웜 네트워크

- 오버레이 네트워크
  - ➢ 오버레이 네트워크 사용 도커 스웜 서비스 생성

```
root@swarm-manager:~# docker service create --name overlay_service \
 --network myoverlay \
 --replicas 2 \
 ubuntu:apache-php
z966mqlx4v4sakgrrrx5szr4c
```

  - ➢ 서비스 컨테이너 네트워크 정보 확인

```
root@swarm-manager:~# docker exec 69968b60b0a6 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0a:00:09:04
          inet addr:10.0.9.4  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe00:904/64 Scope:Link
..

eth1      Link encap:Ethernet  HWaddr 02:42:ac:12:00:09
          inet addr:172.18.0.9  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe12:9/64 Scope:Link
..
```

    - eth0 : 오버레이 네트워크
    - eth1 : 컨테이너 네트워크

도커 스웜 네트워크

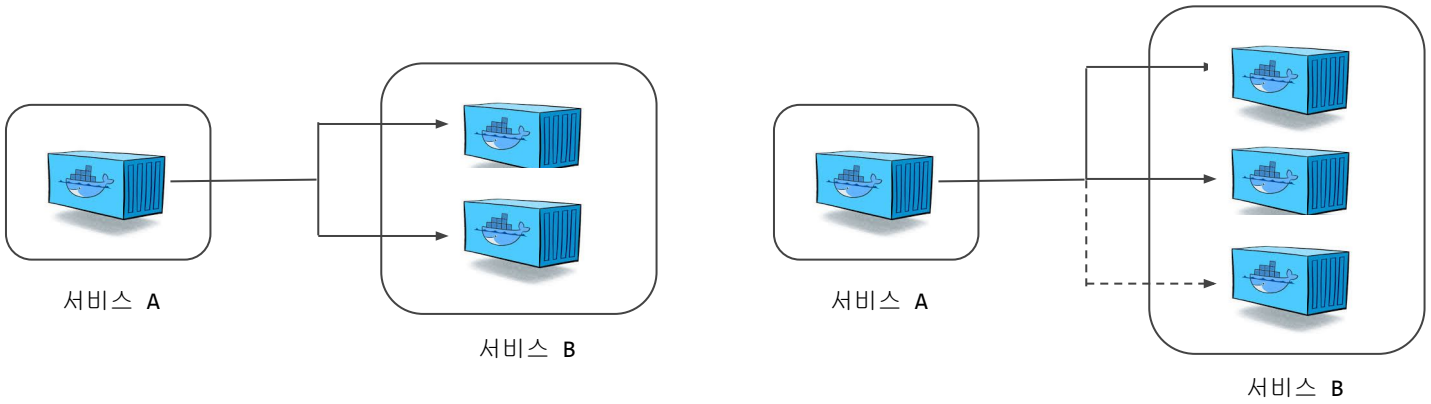
- docker_gwbridge 네트워크
  - 오버레이 네트워크를 사용하지 않는 컨테이너는 기본으로 브리지 (bridge) 네트워크 사용
  - ingress를 포함한 오버레이 네트워크는 docker_gwbridge 네트워크 사용
  - 외부로 나가는 통신 및 오버레이 네트워크의 트래픽 종단점 (VTEP) 역할 담당

➢ docker_gwbridge 네트워크 확인

```
root@swarm-manager:~# docker network ls | grep docker
f8d483f685c6        docker_gwbridge     bridge              local
```

도커 스웜 네트워크

- 서비스 디스커버리
  - 도커 스웜 모드는 서비스 발견 기능을 자체적으로 지원
  - 새로운 컨테이너가 생성 되거나, 없어졌을 경우 컨테이너를 자동 감지
  - 서비스간 통신은 서비스 이름으로 접근



  - A 서비스는 B 서비스의 컨테이너를 상용중
  - B 서비스의 컨테이너를 Scale-out 실행 후 B 서비스 이름으로 접근가능

도커 스웜 네트워크

- 서비스 디스커버리
  - ➢ 서비스 디스커버리 테스트를 위한 오버레이 네트워크 생성

```
root@swarm-manager:~# docker network create -d overlay discovery
a01mmyzhgu11b2i775v687gf9
```

  - ➢ discovery 오버레이 네트워크를 사용한 server 서비스

```
root@swarm-manager:~# docker service create --name server \
 --replicas 2 \
 --network discovery \
 ubuntu:apache-php
q3juytscg999eu9nzjdm2rw7r
```

  - ➢ discovery 오버레이 네트워크를 사용한 서비스 client 생성

```
root@swarm-manager:~# docker service create --name client \
 --network discovery \
 ubuntu:14.04 \
 ping docker.com
rzlj6aki5gakx96zyrpbldcfx
```

  - ➢ client 서비스 컨테이너 위치 확인

```
root@swarm-manager:~# docker service ps client
ID            NAME      IMAGE         NODE           DESIRED STATE  CURRENT STATE           ERROR  PORTS
lb9bu9bqgrf3  client.1  ubuntu:14.04  swarm-worker2  Running        Running 40 seconds ago
```

도커 스웜 네트워크

- 서비스 디스커버리
  - client 서비스 컨테이너 위치 확인 및 들어가기

```
root@swarm-worker2:~# docker ps --format "table {{.ID}}\t{{.Command}}" | grep ping
9a61b942f012        "ping docker.com"

root@swarm-worker2:~# docker exec -it 9a61b942f012 bash
root@9a61b942f012:/#
```

  - client -> server 서비스 접근 테스트

```
root@swarm-worker2:~# docker docker
root@9a61b942f012:/#

root@9a61b942f012:/#
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
84e69d779e1c
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
42b366ff0ca0
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
84e69d779e1c
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
42b366ff0ca0
```

  - service 서비스 컨테이너 Scale-out

```
root@swarm-manager:~# docker service scale server=3
server scaled to 3
```

도커 스웜 네트워크

- 서비스 디스커버리
  - ➢ client -> server 서비스 접근 테스트

```
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
366383f1281a
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
84e69d779e1c
root@9a61b942f012:/# curl -s server/hostname.php && echo ""
42b366ff0ca0
```

  - ➢ server 서비스 vip 확인

```
root@swarm-manager:~# docker service inspect --format {{.Endpoint.VirtualIPs}} server
[{a01mmyzhgu11b2i775v687gf9 10.0.1.2/24}]
```

  - ➢ client -> server ping 확인

```
root@9a61b942f012:/# ping server
PING server (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.026 ms
..

root@9a61b942f012:/# ping server
PING server (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.020 ms
..
```

도커 스웜 볼륨

- 도커 스웜 모드 볼륨 개요
  - ➢ 기본 도커 볼륨 사용은 -v 옵션으로 도커볼륨 또는 호스트 디렉토리 사용

```
### 호스트와 디렉토리를 공유 하는 경우 ###
# docker run -i -t --name host_dir_case -v /root:/root ubuntu:14.04

### 도커 볼륨을 사용하는 경우 ###
# docker run -i -t --name host_dir_case -v myvolume:/root ubuntu:14.04
```

  - ➢ 도커 스웜 모드에서 볼륨 사용시 --mount 옵션을 통한 상세 설정 지정

```
root@swarm-manager:~# docker service create --name ubuntu \
 --mount type=volume,source=myvol,target=/root \
 ubuntu:14.04 \
 ping docker.com
utmdgy0lns72g9u4gjn6jutq6
```

    - type : 도커 볼륨 사용은 volume, 호스티 디렉토리 사용은 bind 로 설정
    - source : 사용할 볼륨명, source 를 지정하지 않으면 16진수의 새로운 볼륨 생성
    - target : 컨테이너 내부에 마운트 될 디렉토리 위치

도커 스웜 볼륨

- 도커 스웜 모드 볼륨 개요
  - ➢ 컨테이너 디렉토리의 파일 복사

    컨테이너의 디렉토리에 파일이 있으면 볼륨으로 복사됨

```
root@swarm-manager:~# docker service create --name ubuntu-copy \
 --mount type=volume,source=test,target=/etc/vim/ \
 ubuntu:14.04 \
 ping docker.com
9q2rqnlegtqxt8pnp72plcab1

root@swarm-manager:~# docker run -i -t --name test \
 -v test:/root \
 ubuntu:14.04

root@49dfc1f73aab:/#  ls root/
vimrc  vimrc.tiny
```

  - ➢ 컨테이너 디렉토리의 파일 복사 방지

    컨테이너의 디렉토리에 파일이 있으면 볼륨으로 복사방지

```
root@swarm-manager:~# docker service create --name ubuntu-nocopy \
 --mount type=volume,source=test,target=/etc/vim/,volume-nocopy \
 ubuntu:14.04 \
 ping docker.com
```

도커 스웜 볼륨

- bind 타입 볼륨 생성
  - ➢ 호스트의 디렉토리를 공유하기 위해서는 반드시 source 옵션을 명시해야 함

```
root@swarm-manager:~# mkdir /root/host
root@swarm-manager:~# touch /root/host/host-file

root@swarm-manager:~# docker service create --name ubuntu-host \
 --mount type=bind,source=/root/host,target=/root/container \
 ubuntu:14.04 \
 ping docker.com
lrnw9n1kme3rtngvcg5z9wl0c
```

도커 스웜 노드 Availability

- 도커 스웜 노드 Availability
  - 컨테이너 할당의 스케줄을 위해 사용 가능

- 도커 스웜 노드 Availability 상태
  - Active : 컨테이너 할당이 가능한 정상적인 상태
  - Drain : 노드에 문제가 발생해 일시적으로 사용하지 않은 상태로 설정시 사용
    실행중인 서비스 컨테이너는 Active 상태의 노드로 변경 할당 됨
  - Pause : 일시적인 중지로 컨테이너 할당이 되지 않지만, 실행중인 컨테이너가 죽거나 다른 Active 노드로 변경 할당 되지 않음

➢ 도커 스웜 노드 Availability 상태 확인

```
root@swarm-manager:~/host# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Active
```

도커 스웜 노드 Availability

- 도커 스웜 노드 Availability 상태
  - ➢ 도커 스웜 노드 Availability 를 Active -> Drain 으로 변경

```
root@swarm-manager:~/host# docker node update \
 --availability drain \
 swarm-worker1
swarm-worker1

root@swarm-manager:~/host# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Active
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Drain
```

  - ➢ 도커 스웜 노드 Availability 를 Active -> Pause 으로 변경

```
root@swarm-manager:~/host# docker node update \
 --availability pause \
 swarm-worker2
swarm-worker2

root@swarm-manager:~/host# docker node ls
ID                            HOSTNAME       STATUS  AVAILABILITY  MANAGER STATUS
plwgqgyn9vaw2p41vjo86lmnm *   swarm-manager  Ready   Active        Leader
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Pause
ywnkgswoukp0mjis1wwuy1jv2     swarm-worker1  Ready   Drain
```

도커 스웜 노드 Label

- 도커 스웜 노드 Label
  - 노드를 분류하기 위해 노드 특성을 키-값 으로 지정
  - 컨테이너 할당시 label 을 명시하여 특정 노드에 할당 가능

➢ 노드 label 추가

```
root@swarm-manager:~/host# docker node update \
 --label-add storage=ssd \
 swarm-worker1
swarm-worker1
```

➢ 노드 label 확인

```
root@swarm-manager:~/host# docker node inspect --pretty swarm-worker1
ID:                     ywnkgswoukp0mjis1wwuy1jv2
Labels:
 - storage = ssd
Hostname:               swarm-worker1
Joined at:              2018-06-09 12:11:23.754000444 +0000 utc
Status:
 State:                 Ready
 Availability:          Drain
 Address:               192.168.35.101
..
```

도커 스웜 서비스 제약

- 도커 스웜 서비스 제약 설정
  - 서비스 컨테이너가 할당될 노드의 종류 선택 기능 제공
  - 컨테이너 할당시 label 을 명시하여 특정 노드에 할당 가능
  - node.labels 제약조건 컨테이너 생성

```
root@swarm-manager:~/host# docker service create --name label-test \
 --constraint 'node.labels.storage == ssd' \
 --replicas=2 \
 ubuntu:14.04 \
 ping docker.com
rg1bzl4e0ss1w12hp0if2tx9a

root@swarm-manager:~/host# docker service ps label-test
ID            NAME          IMAGE         NODE  DESIRED STATE  CURRENT STATE               ERROR
PORTS
xudz7pmld3lr  label-test.1  ubuntu:14.04        Running        Pending about a minute ago
zkxt8lffi1rv  label-test.2  ubuntu:14.04        Running        Pending about a minute ago
```

  - 제약 조건에 해당하는 노드가 없으면 컨테이너 생성이 안됨

도커 스웜 서비스 제약

- 도커 스웜 서비스 제약 설정
  - ➢ node.id 제약조건 컨테이너 생성

    노드의 ID를 명시해 서비스 컨테이너 생성, ID 값은 전체를 입력해야 함

```
root@swarm-manager:~/host# docker node ls | grep swarm-worker2
vz7prsi133nzubvy65wydtxzf     swarm-worker2  Ready   Pause

root@swarm-manager:~/host# docker service create --name label-test2 \
 --constraint 'node.id == vz7prsi133nzubvy65wydtxzf' \
 --replicas=2 \
 ubuntu:14.04 \
 ping docker.com
22n4uf057fipugvxgz5argakz

root@swarm-manager:~/host# docker service ps label-test2
ID            NAME           IMAGE         NODE  DESIRED STATE  CURRENT STATE           ERROR  PORTS
xiooniv9ouon  label-test2.1  ubuntu:14.04        Running        Pending 42 seconds ago
r5kctankbru5  label-test2.2  ubuntu:14.04        Running        Pending 42 seconds ago
```

도커 스웜 서비스 제약

- 도커 스웜 서비스 제약 설정
  - ➢ node.hostname 제약조건 컨테이너 생성

```
root@swarm-manager:~/host# docker service create --name label-test3 \
 --constraint 'node.hostname == swarm-worker1' \
 ubuntu:14.04 \
 ping docker.com
q87ua4u22dshakwmx5bl5tq4y
```

  - ➢ node.role 제약조건 컨테이너 생성

```
root@swarm-manager:~/host# docker service create --name label-test4 \
 --constraint 'node.role != manager' \
 --replicas 2 \
 ubuntu:14.04 \
 ping docker.com
fn01afzj9yi0nvweqq4jawdmp
```

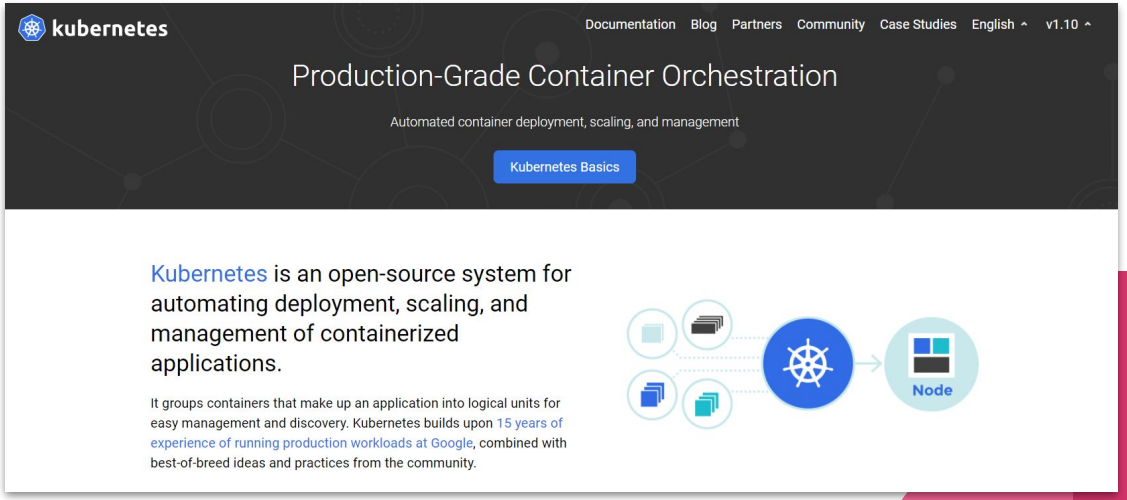
    - != : 조건에 맞지 않은 제약설정

4장 쿠버네티스

- 쿠버네티스 소개
- 쿠버네티스 설치
- 쿠버네티스 컴포넌트
- 쿠버네티스 객체
- 쿠버네티스 네트워크
- 쿠버네티스 스토리지

쿠버네티스 소개

- 쿠버네티스(Kubernetes, 약어로 k8s) 개발 배경
  - 도커는 컨테이너를 규모에 맞게 늘려가도록 배치하는 기능이 부족
  - 컨테이너 애플리케이션을 배포하는 오케스트레이터를 구글이 개발
  - 구글은 내부 서비스를 클라우드와 컨테이너 환경으로 오래전부터 사용
  - 구글이 2014년 6월 오픈소스 프로젝트로 발표
  - 2015년 7월 버전 1.0을 기반으로 CNCF(Cloud Native Computing Foundation)을 설립
  - 레드햇, 이베이, AT&T, 시스코, IBM, 인텔, 트위터, VMware 등 다수 회사참여 개발



https://kubernetes.io

쿠버네티스 설치

- 쿠버네티스 설치 방식
  - 올인원 쿠버네티스 : 마스터와 노드가 동일한 호스트에 실행, 기본 작동 방식을 이해하기 유리하지만, 실제 환경에 적용하기 적절치 못함
  - 쿠버네티스 클러스터 : 하나의 마스터와 최소 두 개 이상의 노드로 구성, 각각 별도의 시스템에서 구동, 필요할 때마다 노드를 추가해 용량을 늘려야 하는 실제 환경에서의 구성 방식

  ➢ 쿠버네티스는 다양한 방식의 설치 방법을 제공, 쿠버네티스 개발 진행을 잘 따라 갈 수 있는 리눅스 환경을 사용하는것을 권장

- 리눅스 환경의 쿠버네티스 구성 방법
  - 쿠버네티스 설치를 위한 VM 생성
  - 물리 서버머신에 직접 설치
  - 가상머신 배치도구(Vagrant) 를 이용한 설치
  - 클라우드 사업자가 제공하는 쿠버네티스 사용

Centos7 쿠버네티스 설치

- Centos7 설치 참조 가이드
  - Centos7 Kubernetes Install
    https://blog.tekspace.io/setup-kubernetes-cluster-on-centos-7/
    https://www.linuxtechi.com/install-kubernetes-1-7-centos7-rhel7/
- Centos7 환경 설정 변경
  - Centos7 Swap disable
    https://www.refmanual.com/2016/01/08/completely-remove-swap-on-ce7/#.WxWsKUiFP-g

```
# sudo swapoff -a
# sudo lvremove -Ay /dev/centos/swap

# sudo vi /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
##GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb quiet"
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root crashkernel=auto rhgb quiet"
GRUB_DISABLE_RECOVERY="true"

# sudo cp /etc/grub2.cfg /etc/grub2.cfg.bak
# sudo grub2-mkconfig >/etc/grub2.cfg
# reboot
```

Centos7 쿠버네티스 설치

- 노드설치 공통작업
  - yum repo 추가

```
# sudo bash -c 'cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF'
```

  - kubeadm, docker 패키지 설치

```
# yum install kubeadm docker -y
```

  - kubeadm, docker 서비스 시작

```
# systemctl restart docker && systemctl enable docker

# systemctl  restart kubelet && systemctl enable kubelet
```

Centos7 쿠버네티스 설치

- Master 노드설치
  - ➢ 호스트 네임 설정

```
# hostnamectl set-hostname kube-master
```

  - ➢ Master 노드 초기화

```
# kubeadm init
..
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join 192.168.35.50:6443 --token 4fx44e.lmdpvef1ioxiy0m2 --discovery-token-ca-cert-hash
sha256:00471908ef45c3fa0323cf6b241c6713b50fb2c4a3c9aa18655aa5b1b5de6d28
```

Centos7 쿠버네티스 설치

- Master 노드설치
  - ➢ Master 노드 환경변수 설정

```
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

  - ➢ Master 노드 상태 확인

```
# kubectl get nodes
NAME           STATUS     ROLES     AGE       VERSION
kuber-master   NotReady   master    3m        v1.10.3
```

  - ➢ Master 설치 컴포넌트 확인

```
[root@kuber-master ~]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY     STATUS    RESTARTS   AGE
kube-system   etcd-kuber-master                      1/1       Running   0          3m
kube-system   kube-apiserver-kuber-master            1/1       Running   0          3m
kube-system   kube-controller-manager-kuber-master   1/1       Running   0          2m
kube-system   kube-dns-86f4d74b45-26qnt              0/3       Pending   0          3m
kube-system   kube-proxy-smf2g                       1/1       Running   0          3m
kube-system   kube-scheduler-kuber-master            1/1       Running   0          3m
```

Centos7 쿠버네티스 설치

- 노드설치
  - ➢ 호스트 네임 설정

```
# hostnamectl set-hostname kube-node1
```

  - ➢ 일반 노드 클러스터 등록

```
#  kubeadm join 192.168.35.50:6443 --token 4fx44e.lmdpvef1ioxiy0m2 --discovery-token-ca-cert-hash
sha256:00471908ef45c3fa0323cf6b241c6713b50fb2c4a3c9aa18655aa5b1b5de6d28
..
[discovery] Cluster info signature and contents are valid and TLS certificate validates against
pinned roots, will use API Server "192.168.35.50:6443"
[discovery] Successfully established connection with API Server "192.168.35.50:6443"

This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

  - ➢ 일반 노드 등록 확인

```
[root@kuber-master ~]# kubectl get nodes
NAME           STATUS     ROLES     AGE       VERSION
kuber-master   NotReady   master    2h        v1.10.3
kuber-node1    NotReady   <none>    59s       v1.10.3
```

Ubuntu16.04 쿠버네티스 설치

- Ubuntu16.04 설치 참조 가이드
  - Ubuntu16.04 Kubernetes Install
    https://blog.tekspace.io/setup-kubernetes-cluster-with-ubuntu-16-04/
    http://iamartin-gh.herokuapp.com/kubernetes-install/

- Ubuntu16.04 환경 설정 변경
  - ➢ Ubuntu16.04 Swap disable
    http://thdev.net/127

```
# sudo swapoff -a

# sudo vi /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=643efedd-98a5-485e-8a79-974ed9693ac0 /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
#UUID=e93e4f6a-5055-42f8-a3d9-f90f95226276 none            swap    sw              0       0

# reboot
```

Ubuntu16.04 쿠버네티스 설치

- 노드설치 공통작업
  - ➢ apt repo 추가

```
# sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
ok

# sudo vi /etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main

# sudo apt-get update
```

  - ➢ kubeadm, docker 패키지 설치

```
# sudo apt install docker.io -y
```

  - ➢ docker 서비스 시작

```
# systemctl restart docker && systemctl enable docker
```

Ubuntu16.04 쿠버네티스 설치

- Master 노드설치
  - ➢ 호스트 네임 설정

```
# hostnamectl set-hostname kube-master
```

  - ➢ Master 노드 패키지 설치

```
# sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

  - ➢ Master 노드 초기화

```
# sudo kubeadm init
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join 192.168.35.50:6443 --token hm6os1.qoi3qn1hecpqx6pl --discovery-token-ca-cert-hash
sha256:efe798a7234e5e3b6a0b92eb3dfe88ec03fea910242c518f551d71088c6a0040
```

Ubuntu16.04 쿠버네티스 설치

- Master 노드설치
  - ➢ Master 노드 환경변수 설정

```
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

  - ➢ Master 노드 상태 확인

```
# kubectl get nodes
NAME           STATUS     ROLES     AGE       VERSION
kuber-master   NotReady   master    3m        v1.10.3
```

  - ➢ Master 설치 컴포넌트 확인

```
# kubectl get pods --all-namespaces
NAMESPACE     NAME                                 READY     STATUS    RESTARTS   AGE
kube-system   coredns-78fcdf6894-2rpr2             0/1       Pending   0          4m
kube-system   coredns-78fcdf6894-ll9rn             0/1       Pending   0          4m
kube-system   etcd-kube-master                     1/1       Running   0          3m
kube-system   kube-apiserver-kube-master           1/1       Running   0          3m
kube-system   kube-controller-manager-kube-master  1/1       Running   0          3m
kube-system   kube-proxy-fw62l                     1/1       Running   0          4m
kube-system   kube-scheduler-kube-master           1/1       Running   0          3m
```

Ubuntu16.04 쿠버네티스 설치

- Master 노드설치
  - ➢ POD 네트워크 플러그인 설치

```
# kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d
'\n')"
serviceaccount/weave-net created
```

  - ➢ Master 설치 컴포넌트 확인

```
# kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY     STATUS    RESTARTS   AGE
kube-system   coredns-78fcdf6894-2rpr2               1/1       Running   0          4m
kube-system   coredns-78fcdf6894-l19rn               1/1       Running   0          4m
kube-system   etcd-kube-master                       1/1       Running   0          3m
kube-system   kube-apiserver-kube-master             1/1       Running   0          3m
kube-system   kube-controller-manager-kube-master    1/1       Running   0          4m
kube-system   kube-proxy-fw62l                       1/1       Running   0          4m
kube-system   kube-scheduler-kube-master             1/1       Running   0          4m
kube-system   weave-net-wjst5                        2/2       Running   0          24s
```

Ubuntu16.04 쿠버네티스 설치

- 노드설치
  - ➢ 호스트 네임 설정

```
# hostnamectl set-hostname kube-node1
```

  - ➢ 일반 노드 패키지 설치

```
# sudo apt-get install -y kubelet kubeadm
```

  - ➢ 일반 노드 클러스터 등록

```
#  kubeadm join 192.168.35.50:6443 --token hm6os1.qoi3qn1hecpqx6pl --discovery-token-ca-cert-hash
sha256:efe798a7234e5e3b6a0b92eb3dfe88ec03fea910242c518f551d71088c6a0040

This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

  - ➢ 일반 노드 등록 확인

```
kubectl get nodes
NAME          STATUS    ROLES     AGE       VERSION
kube-master   Ready     master    21m       v1.11.0
kube-node1    Ready     <none>    1m        v1.11.0
```

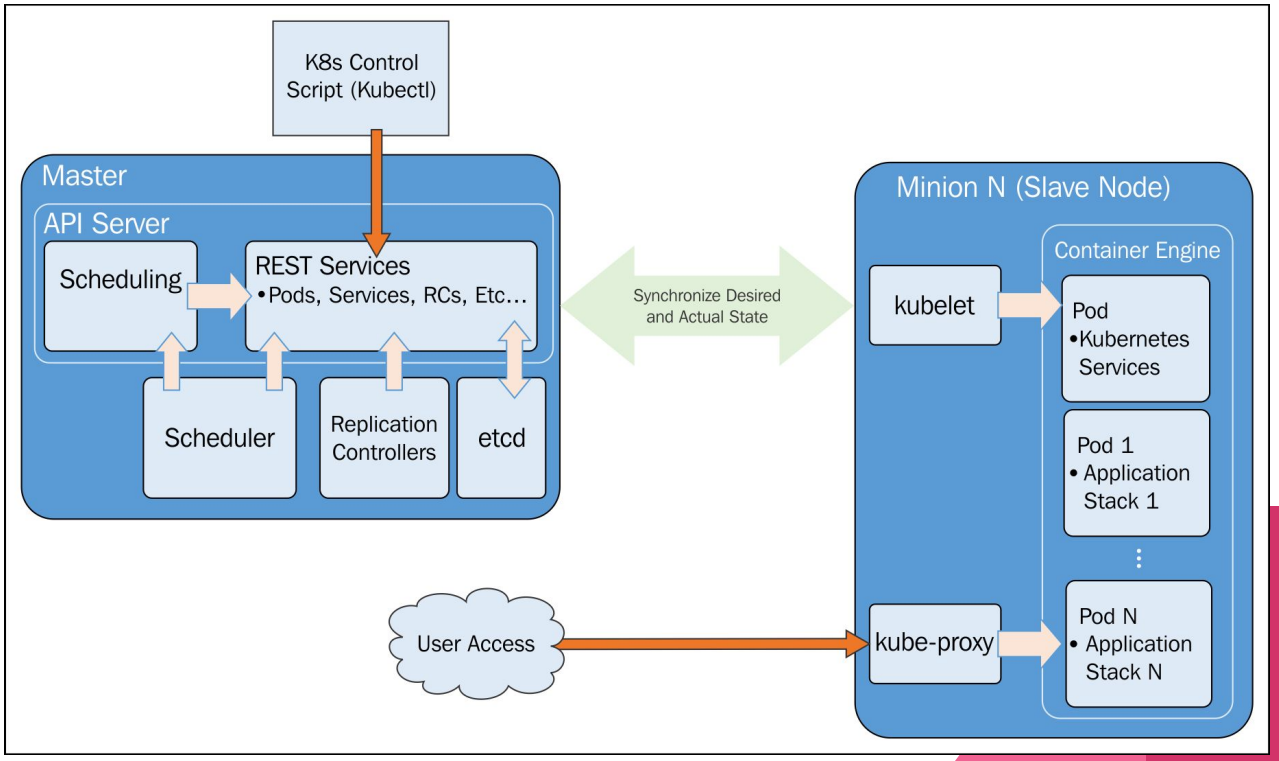
쿠버네티스 설치

- 쿠버네티스 설치 패키지
  - kubeadm
    온프레미스 환경에서도 쿠버네티스를 쉽게 설치할 수 있는 관리 모듈
    각종 설정이나, 컴포넌트를 일일이 다룰 필요 없이 쿠버네티스 클러스터를 간편하게 사용할 수 있는 장점 제공
    마스터와 분산 코디네티너의 다중화 등을 비롯한 여러 세부 기능들이 아직 개발중
    실제 운영환경에 적용하기 적합하지 않을 수 있음
  - kubectl
    쿠버네티스를 관리하기 위해 사용하는 클라이언트 명령으로, 마스터에서 실행
    쿠버네티스에서 관리하는 리소스를 생성하거나 가져오거나, 삭제하는 등의 작업을 수행

✓ 리소스 파일(YAML 또는 JSON)
포드와 리플리케이션 컨트롤러, 서비스를 비롯한 쿠버네티스 리소스를 생성하기 위해 kubectl 명령 실행시 전달해야 할 정보를 YAML 이나 JSON 포맷으로 제공

쿠버네티스 컴포넌트

- 쿠버네티스 구조
  - Master : 포드의 배포 및 관리, 리플리케이션 컨트롤러, 서비스, 노드 등과 같은 쿠버네티스 환경을 구성하는 요소들을 제어하는 컨트롤러 역할 수행
  - Node : 컨테이너가 실제로 구동하는 환경을 제공



쿠버네티스 컴포넌트

- 쿠버네티스 마스터 서비스 데몬
  - etcd
    쿠버네티스 클러스터에 대한 설정 데이터를 저장 하는 분산 데이터 저장소
    watch 기능을 지원으로, 컴포넌트의 변경 사항에 대한 알림을 제공
    대규모 클러스터의 경우 고가용성을 위해 3개에서 많으면 5개 노드의 etcd 클러스터 구성
  - kube-api-server
    각노드에서 보내온 api 요청은 쿠버네트스 api 서버에 전달되고,
    etcd 오브젝트 스토어에 업데이트 하기 전에 검증을 수행 후 모든 데이터를 저장
  - kube-controller-manager
    다양한 매니저를 하나의 바이너리로 통합한 도구
    노드 디스커버리와 모니터링기능,복제컨트롤러, 포트 컨트롤러, 서비스 컨트롤러,
    엔드포인트 컨트롤러 등이 포함
  - kube-scheduler
    아직 스케줄링 되지 않은 포드를 현재 사용가능한 노드에 바인딩 하는 역할 수행
  - dns
    DNS 서비스는 쿠버네티스 1.3부터 표준 쿠버네티스 클러스터에 포함 됐으며,
    일반적인 포트로 스케줄링 된다.
    헤드리스(headless) 서비스를 제외한 나머지 서비스는 dns 이름을 가지며
    포드 역시 dns 이름을 가진다. 이것은 자동 탐색에 매우 유용하다.

쿠버네티스 컴포넌트

- 쿠버네티스 노드 서비스 데몬
  - docker
    컨테이너를 실행하고 관리할 수 있는 기능을 제공
  - kube-proxy
    로드 밸런싱 과 네트워크 프록시 기능을 담당.
    서비스를 지역적으로 반영하고 TCP 및 UDP 포워딩을 수행
    환경변수나 dns 를 통해 클러스터 IP를 찾는다.
    이 작업은 서비스 엔드포인트를 제어하는 방식으로 처리
  - kubelet
    노드에 존재하는 포드의 다양한 부분을 관리하며, 마스터에 있는 API 서버와 통신
    api 서버에서 포트 secret 다운로드
    볼륨 마운트
    포드의 컨테이너 실행 (도커 또는 Rkt)
    노드와 각 포트 상태 보고
    컨테이너 활성여부 조사

쿠버네티스 컴포넌트

- 쿠버네티스 클라이언트
  - kubectl : 쿠버네티스 API와 상호작용하기 위한 CLI 명령어 도구
    포드, 리플리카세트, 서비스 등 대부분의 쿠버네티스 객체를 관리하는데 사용

- 쿠버네티스 상태 확인

➢ 버전확인

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.3", ..
Server Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.3", ..
```

➢ 컴포넌트 확인

```
# kubectl get componentstatuses
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

➢ 노드 리스트 확인

```
# kubectl get nodes
NAME          STATUS    ROLES     AGE       VERSION
kube-master   Ready     master    21d       v1.10.3
kube-node1    Ready     <none>    21d       v1.10.3
```

쿠버네티스 컴포넌트

- 쿠버네티스 상태 확인
  - ➢ 노드 상세 정보 확인

    노드 CPU 정보 와 자원 상태 정보 출력

```
# kubectl describe nodes kube-node1
Name:               kube-node1
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=kube-node1
..
CreationTimestamp:  Wed, 06 Jun 2018 08:01:58 +0900
Taints:             <none>
Unschedulable:      false

Conditions:
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason                       Message
  ----             ------  -----------------                 ------------------                ------                       -------
  OutOfDisk        False   Wed, 27 Jun 2018 11:14:42 +0900   Wed, 06 Jun 2018 08:01:58 +0900   KubeletHasSufficientDisk     kubelet has
sufficient disk space available
  MemoryPressure   False   Wed, 27 Jun 2018 11:14:42 +0900   Wed, 06 Jun 2018 08:01:58 +0900   KubeletHasSufficientMemory   kubelet has
sufficient memory available
  DiskPressure     False   Wed, 27 Jun 2018 11:14:42 +0900   Wed, 06 Jun 2018 08:01:58 +0900   KubeletHasNoDiskPressure     kubelet has no disk
pressure
  PIDPressure      False   Wed, 27 Jun 2018 11:14:42 +0900   Wed, 06 Jun 2018 08:01:58 +0900   KubeletHasSufficientPID      kubelet has
sufficient PID available
  Ready            True    Wed, 27 Jun 2018 11:14:42 +0900   Mon, 25 Jun 2018 09:57:20 +0900   KubeletReady                 kubelet is posting
ready status. AppArmor enabled
Addresses:
  InternalIP:  192.168.35.51
  Hostname:    kube-node1
```

쿠버네티스 컴포넌트

- 쿠버네티스 상태 확인
  - ➢ 노드 자원 가용량 정보 와 소프트웨어 설치 정보 확인

```
Capacity:
 cpu:                4
 ephemeral-storage:  40168028Ki
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             4028132Ki
 pods:               110
Allocatable:
 cpu:                4
 ephemeral-storage:  37018854544
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             3925732Ki
 pods:               110

System Info:
 Machine ID:                 736b7f27fc35a3977b975c6d5b16fe52
 System UUID:                51664D56-C9C1-DF30-44D4-7B7692621657
 Boot ID:                    dfa9a93e-63ef-4879-bb9d-4de84b35bfbe
 Kernel Version:             4.4.0-116-generic
 OS Image:                   Ubuntu 16.04.4 LTS
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://1.13.1
 Kubelet Version:            v1.10.3
 Kube-Proxy Version:         v1.10.3
PodCIDR:                     10.244.1.0/24
ExternalID:                  kube-node1
```

쿠버네티스 컴포넌트

- 쿠버네티스 상태 확인
  - ➢ 노드에서 실행 중인 POD 정보 와 자원 할당량 확인

```
Non-terminated Pods:         (5 in total)
  Namespace                  Name                                CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ---------                  ----                                ------------  ----------  ---------------  -------------
  default                    alpaca-prod-7f94b54866-b4c2q        0 (0%)        0 (0%)      0 (0%)           0 (0%)
  default                    bandicoot-prod-85ddf4c7dd-2pgnc     0 (0%)        0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-flannel-ds-fnbwk               100m (2%)     100m (2%)   50Mi (1%)        50Mi (1%)
  kube-system                kube-proxy-27glw                    0 (0%)        0 (0%)      0 (0%)           0 (0%)
  kube-system                kubernetes-dashboard-7d5dcdb6d9-6h8zz  0 (0%)        0 (0%)      0 (0%)           0 (0%)

Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ------------  ----------  ---------------  -------------
  100m (2%)     100m (2%)   50Mi (1%)        50Mi (1%)
```

쿠버네티스 컴포넌트

- 쿠버네티스 프록시
  - ➢ 쿠버네티스 내 서비스의 로드밸런싱을 위한 네트워크 트래픽을 라우팅

```
# kubectl get daemonSets --namespace=kube-system kube-proxy
NAME         DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
kube-proxy   3         3         3         3            3           <none>          21d
```

- 쿠버네티스 DNS
  - ➢ 쿠버네티스 내 서비스의 이름지정 과 검색 기능을 제공

```
# kubectl get deployments --namespace=kube-system kube-dns
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-dns   1         1         1            1           21d
```

  - ➢ DNS 서버에 대한 로드밸런싱을 수행하는 서비스 확인

```
# kubectl get services --namespace=kube-system kube-dns
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP   22d
```

    - 컨테이너의 /etc/resolv.conf 파일에서 dns 서버 정보 확인 가능

쿠버네티스 객체

- 쿠버네티스 객체
  - 쿠버네티스는 애플리케이션 서비스 관리를 위해 다양한 객체를 제공
  - 애플리케이션 서비스 배포 및 스케줄링, 이중화, Life cycle 관리 등의 기능 제공
- 쿠버네티스 주요 객체
  - 포드 (Pod)
  - 라벨 (Label) 과 애노테이션 (Annotation)
  - 서비스 (Service)
  - 리플리케이션 컨트롤러 (Replication Controller) 와 리플리카셋 (ReplicaSet)
  - 데몬셋 (Daemon Set)
  - 디플로이먼트 (Deployment)
  - 스테이트풀셋 (Stateful Set)
  - 잡 (Job)
  - 컨피그맵 (ConfigMap)

쿠버네티스 객체 - 포드

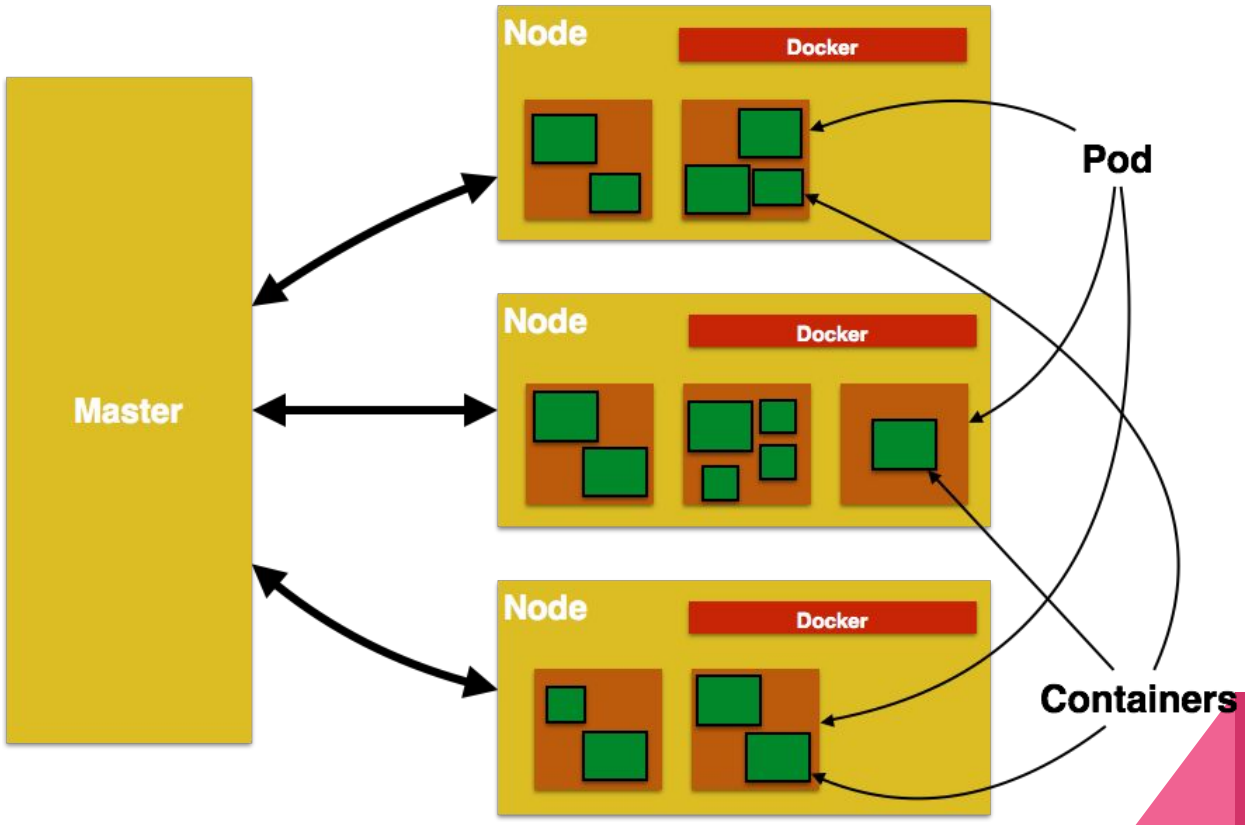
- 포드 (Pod)
  - 쿠버네티스에서 서비스 배포를 위한 기본 단위
  - 애플리케이션 서비스를 위한 컨테이너, 볼륨으로 구성된 집합체
  - 포드에서 동작하는 애플리케이션은 동일한 IP 주소와 포트, 디스크 볼륨을 공유
  - 포드단위로 복제, 스케줄링, 로드밸런싱 가능
  - 포드의 각 컨테이너는 각자의 cgroup 이 정의되지만, 몇가지 리눅스 네임스페이스를 공유 한다.

✓ 네임스페이스
  - lightweight 가상화 솔루션
  - 리눅스 커널을 통해 자원들을 독립된 (고립된) 환경에서 사용가능 하도록 구현
  - UTS, IPC, PID, NS, NET, USER 등의 자원들을 사용할 수 있음

✓ cgroup
  - 리눅스의 프로세스의 자원사용을 제한, 격리하는 리눅스 커널 기능
  - CPU, 메모리, 디스크 입출력, 네트워크 등의 자원을 관리하는 컨트롤러 기능 제공

쿠버네티스 객체 - 포드

- 포드 와 컨테이너 구조



참조 : https://therichwebexperience.com/blog/arun_gupta/2015/07/kubernetes_design_patterns

쿠버네티스 객체 - 포드

- 포드 생성
  - ➢ kubectl run 명령어를 사용한 포드 생성

```
# kubectl run kuard --image=gcr.io/kuar-demo/kuard-amd64:1
deployment.apps "kuard" created
```

  - ➢ 생성된 포드 실행 확인

```
# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
kuard-b75468d67-wh6vl    1/1       Running   0          49s
```

  - ➢ 포드 삭제

```
# kubectl delete pod kuard-b75468d67-wh6vl
pod "kuard-b75468d67-wh6vl" deleted
```

쿠버네티스 객체 - 포드

- 매니페스트(Manifest)를 이용한 포드 생성
  - 매니페스트 : YAML 이나 JSON 형식으로 작성된 POD 생성 참조 파일
    키 필드와 속성의 쌍으로 구성되며, 소스코드 처럼 주석을 추가 할 수 있다.

➢ 매니페스트 작성

```
# vi kuard-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kuard-manifest
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

➢ 매니페스트를 이용한 포드 생성

```
# kubectl apply -f kuard-pod.yaml
pod "kuard-manifest" created
```

쿠버네티스 객체 - 포드

- 매니페스트(Manifest)를 이용한 포드 생성
  - ➢ 생성된 포드 실행 확인

```
# kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
kuard-manifest          1/1       Running   0          23s
```

  - ➢ 포드 상세정보 확인

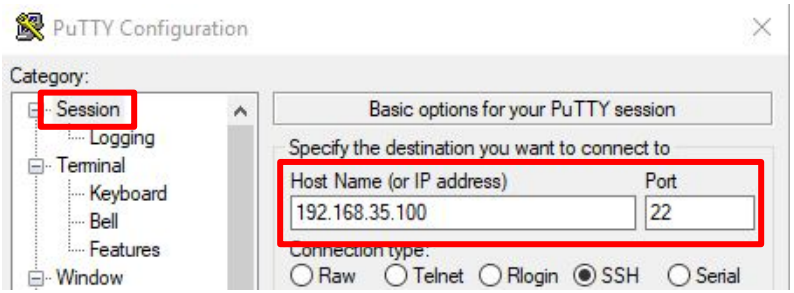
```
# kubectl describe pods kuard-manifest
Name:         kuard-manifest
Namespace:    default
Node:         kube-node2/192.168.35.52
Start Time:   Sat, 30 Jun 2018 01:11:14 +0900
..
Containers:
  kuard:
    Container ID:   docker://b0604dd0f23bc5524c020283a3c20836f37159c06508f663dea5f479173e8a7d
    Image:          gcr.io/kuar-demo/kuard-amd64:1
    Image ID:
docker-pullable://gcr.io/kuar-demo/kuard-amd64@sha256:3e75660dfe00ba63d0e6b5db2985a7ed9c07c3e115faba2
91f899b05db0acd91
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 30 Jun 2018 01:11:25 +0900
..
```

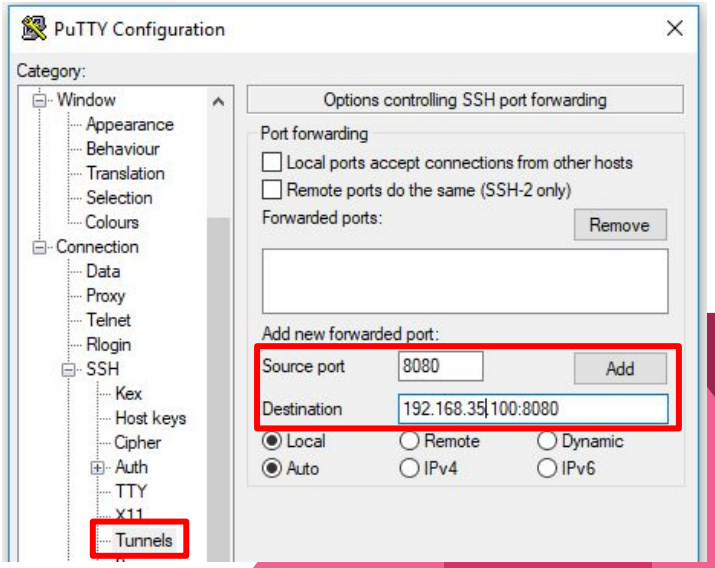
쿠버네티스 객체 - 포드

- 포드 관리
  - ➢ 포트 포워딩을 사용한 포드 접속

```
# kubectl port-forward kuard-manifest 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

    - port-forward : 로컬 머신 -> 쿠버네티스 마스터 -> 노드 포드 인스턴스 보안채널 생성
  - ➢ 사용자 PC -> 쿠버네티스 마스터 ssh 포트포워딩 접속
    - putty 포트 포워딩 설정 후 ssh 접속



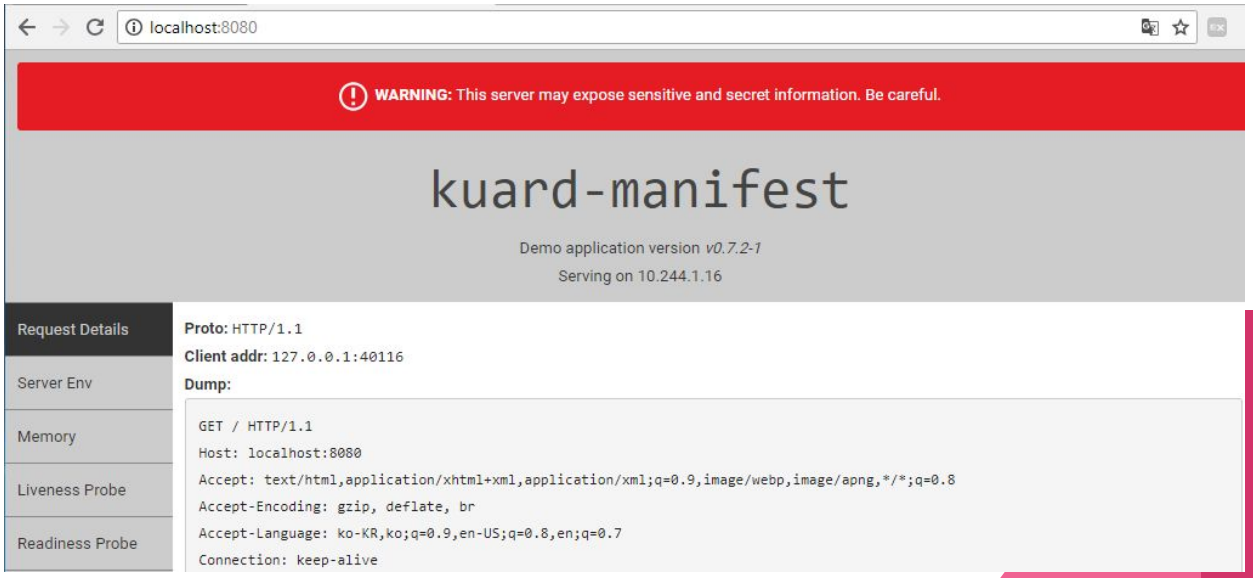


쿠버네티스 객체 - 포드

- 포드 관리
  - ➢ 사용자 PC의 포트포워딩 확인

```
C:\Users\usesr1>netstat -anop tcp
..
TCP    127.0.0.1:8080         0.0.0.0:0              LISTENING       8564
..
```

  - ○ http://localhost:8080 을 통해 포드에 접속 테스트



쿠버네티스 객체 - 포드

- 포드 관리
  - ➢ 실행중인 포드 인스턴스의 로그 확인

```
# kubectl logs -f kuard-manifest2018/06/2
9 22:05:50 Starting kuard version: v0.7.2-1
..
2018/06/29 22:05:50 Config:
{
  "address": ":8080",
  "debug": false,
  "debug-sitedata-dir": "./sitedata",
..
}
2018/06/29 22:05:50 Could not find certificates to serve TLS
2018/06/29 22:05:50 Serving on HTTP on :8080
2018/06/29 22:09:54 127.0.0.1:40116 GET /
2018/06/29 22:09:54 Loading template for index.html
```

  - -f : 로그내용을 스트림으로 출력
  - --previous : 컨테이너 이전 인스턴스의 로그를 출력
    컨테이너 시작 과정에 문제 발생으로 계속 재시작하는 경우 분석에 유용

쿠버네티스 객체 - 포드

- 포드 관리
  - ➢ 포드 컨테이너에서의 명령 실행

```
# kubectl exec kuard-manifest date
Fri Jun 29 22:33:20 UTC 2018
```

  - ➢ 포드 컨테이너 입출력 쉘 연결

```
# kubectl exec -it kuard-manifest ash
~ $ date
Fri Jun 29 22:35:37 UTC 2018
```

    - ○ exit 또는 Ctrl+D : 컨테이너 빠져나오면서, 컨테이너 정지
  - ➢ 컨테이너 -> 호스트로 파일 복사

```
# kubectl cp kuard-manifest:/etc/hostname ./kuard-hosname.txt
```

  - ➢ 호스트 -> 컨테이너로 파일 복사

```
# kubectl cp $HOME/config.txxt kuard-manifest:/config.txt
```

쿠버네티스 객체 - 포드

- 포드 관리
  - 포드 삭제

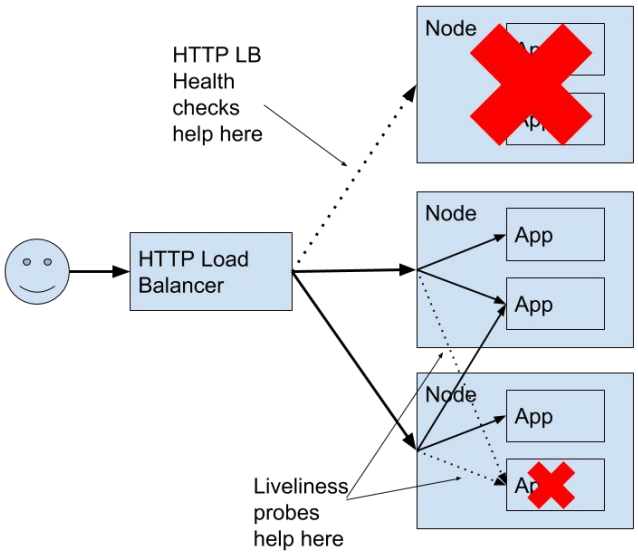
```
# kubectl delete pod kuard-manifest
pod "kuard-manifest" deleted

or

# kubectl delete -f kuard-pod.yaml
pod "kuard-manifest" deleted
```

쿠버네티스 객체 - 포드

- 포드 상태 검사 (Health Check)
  - 쿠버네티스에서 애플리케이션을 컨테이너로 실행하면 상태 검사 가능
  - 애플리케이션의 주요 프로세스 실행 여부 확인
  - 애플리케이션 활성 상태 검사 (liveness Probe) 기능 제공
    프로세스는 정상인데, 서비스가 되지않은 교착상태 (deadlock) 서비스 검증
  - 애플리케이션 서비스 검증은 애플리케이션에 맞는 검증 방법을 매니페스트에 정의해야 함



쿠버네티스 객체 - 포드

- 포드 상태 검사 (Health Check)
  - ➢ 상태 검사를 위한 매니페스트 작성

```
# vi kuard-pod-health.yaml

apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      livenessProbe:
        httpGet:
          path: /healthy
          port: 8080
        initialDelaySeconds: 5 ---> pod 가 생성되고 5초 후 부터 검사 호출
        timeoutSeconds: 1 --------> 요청 응답 대기시간은 1초, http 상태 리턴값은 200이상 400 미만시 정상
        periodSeconds: 10 --------> 10초에 한번씩 요청
        failureThreshold: 3 ------> 3회 이상 실패시 컨테이너 재시작
      ports:
      - containerPort: 8080
        name: http
        protocol: TCP
```

Http GET 요청을 /healthy 경로요 보냄

쿠버네티스 객체 - 포드

- 포드 상태 검사 (Health Check)
  - ➢ 상태 검사 테스트 포드 생성

```
# kubectl apply -f kuard-pod-health.yaml
pod "kuard" created
```

  - ➢ 접속테스트를 위한 포트포워딩 실행

```
# kubectl port-forward kuard 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

  - ➢ 애플리케이션 접속 후 상태검사 확인
    - ○ http://localhost:8080 접속 -> Liveness Probe 클릭

쿠버네티스 객체 - 포드

- 포드 자원 관리
  - 쿠버네티스는 애플리케이션이 사용할 최소자원 요청 설정이 가능

➢ 최소자원 설정 매니페스트

```
# vi kuard-pod-resreq.yaml

apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      resources:
        requests:      ----------> 최소자원 요청 설정
          cpu: "500m"   ---------> 최소 CPU 0.5 할당
          memory: "128Mi" ------> 최소 메모리 128M 할당
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

✓ 포드에 할당된 자원은 컨테이너별로 요청 되고, 컨테이너의 자원의 합은 포드가 요청하는 총 합이 된다.

쿠버네티스 객체 - 포드

- 포드 자원 관리
  - 쿠버네티스는 애플리케이션이 사용할 최대자원 제한 설정이 가능

➢ 최대자원 설정 매니페스트

```
# vi kuard-pod-reslim.yaml
apiVersion: v1
..
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      resources:
        requests:
          cpu: "500m"
          memory: "128Mi"
        requests:       ----------> 최대자원 사용 제한 설정
          cpu: "1000m"    --------> 최대 CPU 1.0 할당
          memory: "256Mi"   ------> 최대 메모리 256M 할당
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

✓ 애플리케이션에서 최대 사용량 이상의 자원요청이 발생하면 자원요청은 실패 발생 (예: malloc 명령 실패)

쿠버네티스 객체 - 포드

- 포드 볼륨 관리
  - 애플리케이션에서 영구적인 데이터저장을 위해 제공되는 저장소
  - 매니페스트에 접근가능한 볼륨을 정의하고, 컨테이너에 마운트 경로 정의

➢ 볼륨 사용을 위한 매니페스트 작성

```
# vi kuard-pod-vol.yaml
apiVersion: v1
..
spec:
  volumes:
    - name: "kuard-data"
      hostPath:
        path: "/var/lib/kuard"
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      volumeMounts:
        - mountPath: "/data"
          name: "kuard-data"
  ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

pod 에서 사용할 볼륨 정의
호스트의 /var/lib/kuard 를 kuard-data 로 지정

컨테이너의 마운트 경로 설정
kuard-data 로 지정될 볼륨을 컨테이너의 /data 에 마운트

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨 (Label) 과 애노테이션 (Annotation)
  - 쿠버네티스가 어떤 리소스를 사용할 것인지 알려주기 위한 필터링 제공
  - 파드, 리플리카 셋 또는 리플리카 컨틀롤러 서비스 에서 사용가능
  - 리벨과 애노테이션을 이용해 쿠버네티스 자원을 그룹화
- 라벨 (Label)
  - 포드와 리플리카셋 같은 쿠버네티스 객체에 첨부 가능한 키(key), 값(value)
  - 쿠버네티스 객체 식별에 매우 유용
  - 객체의 집합을 참조할 때 라벨 선택기(selector)를 사용
- 애노테이션 (Annotation)
  - 도구와 라이브러리에서 활용할 수 있게 식별 불가능한 정보를 유지하기 위해 설계
  - 쿠버네티스 객체에 추가적인 메타데이터를 저장하는 장소를 제공
  - 외부 시스템간 설정 정보를 전달하거나 도구 자체에 대한 정보를 제공하기 위해 사용
  - 쿠버네티스를 구동하는 다른 프로그램들이 API를 통해 데이터를 저장

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨
  - ➢ 라벨 적용 alpaca 애플리케이션 포드 생성

```
# kubectl run alpaca-prod \
--image=gcr.io/kuar-demo/kuard-amd64:1 \
--replicas=2 \
--labels="ver=1,app=alpaca,env=prod"
deployment.apps "alpaca-prod" created

# kubectl run alpaca-test \
--image=gcr.io/kuar-demo/kuard-amd64:2 \
--replicas=1 \
--labels="ver=2,app=alpaca,env=test"
deployment.apps "alpaca-test" created
```

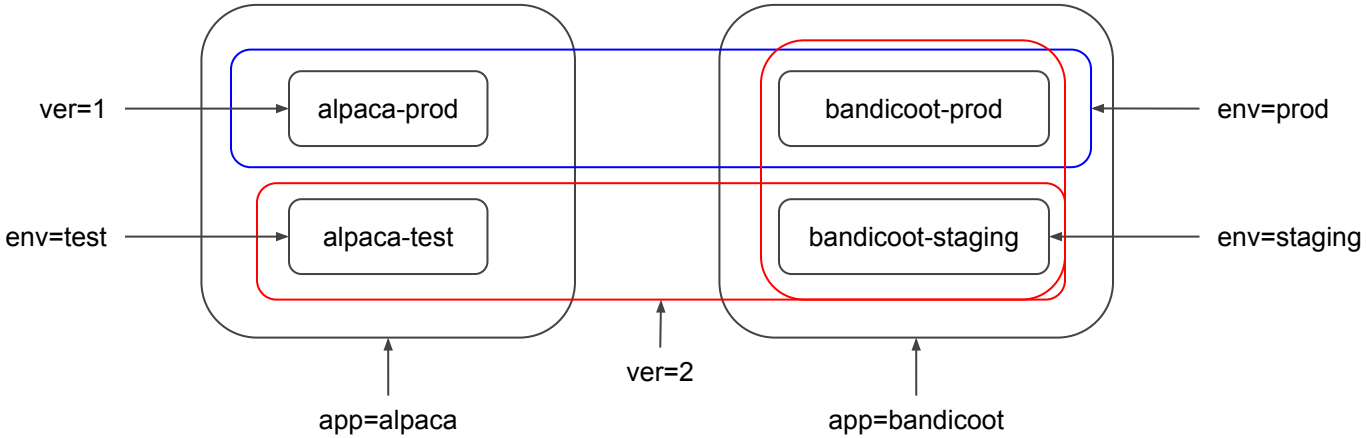
  - ➢ 라벨 적용 bandicoot 애플리케이션 포드 생성

```
# kubectl run bandicoot-prod \
--image=gcr.io/kuar-demo/kuard-amd64:2 \
--replicas=2 \
--labels="ver=2,app=bandicoot,env=prod"
deployment.apps "bandicoot-prod" created

# kubectl run bandicoot-staging \
--image=gcr.io/kuar-demo/kuard-amd64:2 \
--replicas=1 \
--labels="ver=2,app=bandicoot,env=staging"
deployment.apps "bandicoot-staging" created
```

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨
  - alpaca 애플리케이션 과 bandicoot 애플리케이션 라벨 적용 구조



➢ 애플리케이션 라벨 확인

```
# kubectl get deployment --show-labels
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE     LABELS
alpaca-prod        2         2         2            2           39m     app=alpaca,env=prod,ver=1
alpaca-test        1         1         1            1           34m     app=alpaca,env=test,ver=2
bandicoot-prod     2         2         2            2           28m     app=bandicoot,env=prod,ver=2
bandicoot-staging  1         1         1            1           27m     app=bandicoot,env=staging,ver=2
```

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨
  - ➢ 라벨 수정

```
# kubectl label deployments alpaca-test "canary=true"
deployment.extensions "alpaca-test" labeled
```

  - ➢ 라벨 적용 bandicoot 애플리케이션 포드 생성

```
~# kubectl get deployment -L canary
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE       CANARY
alpaca-prod         2         2         2            2           44m
alpaca-test         1         1         1            1           39m       true
bandicoot-prod      2         2         2            2           32m
bandicoot-staging   1         1         1            1           32m
```

    - -L [라벨명] : 지정한 라벨값을 열로 표시

  - ➢ 라벨 삭제

```
kubectl label deployments alpaca-test "canary-"
deployment.extensions "alpaca-test" labeled
```

    - 라벨- : 라벨 접미어에 대시(-) 를 추가해 라벨 삭제

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨 선택기
  - 라벨의 집합을 기반으로 쿠버네티스 객체를 필터링하는 데 사용
  - kubectl 명령과 다른 유형의 객체에서 사용

➢ 라벨선택기를 이용한 포드 조회

```
# kubectl get pods --selector="ver=2"
NAME                                READY     STATUS    RESTARTS   AGE
alpaca-test-6658d779cc-czkvb        1/1       Running   0          3h
bandicoot-prod-7bddc557cc-7np5r     1/1       Running   0          3h
bandicoot-prod-7bddc557cc-bbdwx     1/1       Running   0          3h
bandicoot-staging-7f4788b6df-r8ftx  1/1       Running   0          3h
```

➢ 여러 조건을 만족하는 포드 조회

```
# kubectl get pods --selector="app=bandicoot,ver=2"
NAME                                READY     STATUS    RESTARTS   AGE
bandicoot-prod-7bddc557cc-7np5r     1/1       Running   0          3h
bandicoot-prod-7bddc557cc-bbdwx     1/1       Running   0          3h
bandicoot-staging-7f4788b6df-r8ftx  1/1       Running   0          3h
```

  - 쉼표로 구분된 AND 연산의 조건 조회

쿠버네티스 객체 - 라벨과 애노테이션

- 라벨 선택기
  - ➢ 일련의 값 중 일치하는 조건의 포드 조회

```
# kubectl get pods --selector="app in (alpaca,bandicoot)"
NAME                                READY     STATUS    RESTARTS   AGE
alpaca-prod-65587bf567-b449f        1/1       Running   0          4h
alpaca-prod-65587bf567-hpgml        1/1       Running   0          4h
alpaca-test-6658d779cc-czkvb        1/1       Running   0          4h
bandicoot-prod-7bddc557cc-7np5r     1/1       Running   0          4h
bandicoot-prod-7bddc557cc-bbdwx     1/1       Running   0          4h
bandicoot-staging-7f4788b6df-r8ftx  1/1       Running   0          4h
```

  - ○ 라벤 선택기 연산자

연산자	설명
key = value	value 값과 일치하는 문자열을 가지는 키를 선택
key !- value	value 값과 일치하지 않는 문자열을 가지는 키를 선택
key in (값 1, 값 2 ..)	일련된 값과 일치하는 키를 선택
key notin (값 1, 값 2 ..)	일련된 값과 일치하지 않는 키를 선택
key	키와 일치하는 라벨을 갖는 리소스 선택
!key	키와 일치하지 않는 라벨을 갖는 리소스 선택

쿠버네티스 객체 - 라벨과 애노테이션

- 애노테이션
  - 객체의 출처, 객체의 사용 방법, 객체에 대한 추가 정보 제공에 사용
  - 애노테이션은 라벨과 일부 기능이 겹치며, 사용하는 시기와 취향에 따라 사용
  - 정보를 추가하고 선택기에서 사용하는 경우 해당 애노테이션을 라벨로 생성
  - 애노테이션은 쿠버네티스 객체의 공통 metadata 섹션에 정의

➢ 애노테이션 정의

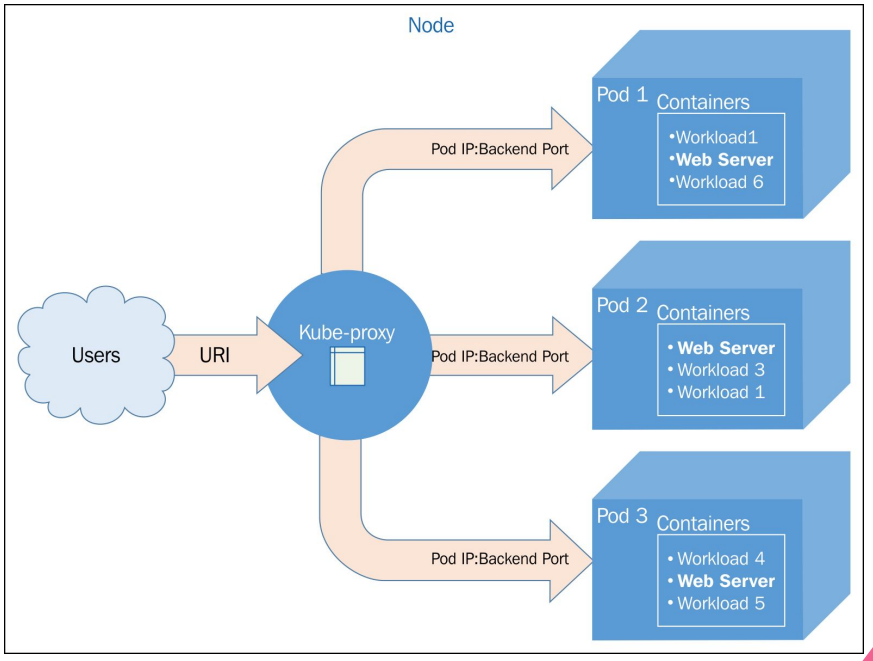
```
...
metadata:
  annotations:
    example.com/icon-url: "https://example.com/icon.png"
...
```

✓ 예제로 만든 디플로이먼트 삭제

```
# kubectl delete deployment --all
deployment.extensions "alpaca-prod" deleted
deployment.extensions "alpaca-test" deleted
deployment.extensions "bandicoot-prod" deleted
deployment.extensions "bandicoot-staging" deleted
```

쿠버네티스 객체 - 서비스

- 서비스
  - 클라이언트가 애플리케이션에 접근할 수 있는 연결 객체를 제공
  - 서비스 로드밸런싱 풀을 구성, 포드 멤버 그룹은 라벨과 선택기 정의로 결정됨
  - 클러스터 IP 와 포트를 생성하고 포드 엔드포인트 IP 와 포트로 연결됨
  - 모든 노드의 kube-proxy 를 통한 클라이어트 접속이 가능



쿠버네티스 객체 - 서비스

- 서비스 타입
  - ClusterIP : 기본 설정값으로, 서비스에 클러스터 IP를 할당함
    쿠버네티스 클러스터 내부에서만 접근이 가능
  - LoadBalancer : 클라우드 벤더의 로드밸런싱 기능을 사용
    클라우드 벤더의 로드밸런싱 외부 IP를 통해 서비스 접근이 가능
  - NodePort : 기본 설정인 서비스 클러스터 IP 할당 과 노드 IP를 통한 접근을 모두 허용
    노드의 임의의 포트 또는 지정된 포트를 통해 서비스 접근 가능
  - ExternalName : 외부 서비스를 쿠버네티스 내부에서 호출할때 사용
    외부 접근할 서비스명은 도메인 또는 IP로 맵핑되어 호출됨

쿠버네티스 객체 - 서비스

- 서비스
  - ➢ kubectl run 명령으로 alpaca-prod 포드 생성
    * kubectl run 명령은 디플로이먼트 객체를 생성하는 명령이다.

```
# kubectl run alpaca-prod \
--image=gcr.io/kuar-demo/kuard-amd64:1 \
--replicas=3 \
--port=8080 \
--labels="ver=1,app=alpaca,env=prod"
deployment.apps "alpaca-prod" created
```

  - ➢ kubectl expose 를 이용한 서비스 생성

```
# kubectl expose deployment alpaca-prod
service "alpaca-prod" exposed
```

    - ○ kubectl expose 명령은 디플로이먼트 정의로 부터 라벨 선택기와 관련 포트(8080)를 편리하게 가져 온다. 또한 서비스에서 클러스터 IP라는 새로운 유형의 가상IP가 할당
      클러스터IP : 로드밸런싱 수행을 위한 특별한 목적의 IP

  - ➢ 서비스 리스트 확인

```
# kubectl get services -o wide
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE       SELECTOR
alpaca-prod   ClusterIP   10.99.58.167   <none>        8080/TCP   1m        app=alpaca,env=prod,ver=1
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP    25d       <none>
```

쿠버네티스 객체 - 서비스

- 서비스
  - ClusterIP 와 POD 포트 연결 구조 확인

```
# kubectl get services
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
alpaca-prod   ClusterIP   10.106.43.73   <none>        8080/TCP   2m41s

# kubectl get endpoints
NAME          ENDPOINTS                                                  AGE
alpaca-prod   192.168.233.198:8080,192.168.9.69:8080,192.168.9.70:8080   2m31s
```

    - 클러스터 IP -> 각 POD IP 로 로드밸런싱 되는구조, 포트는 8080
      10.106.43.73:8080 -> 192.168.233.198:8080,192.168.9.69:8080,192.168.9.70:8080

  - 접속 테스트를 위한 호스트 포트 포워딩

```
# ALPACA_POD=$(kubectl get pods -l app=alpaca -o jsonpath='{.items[1].metadata.name}')
# kubectl port-forward $ALPACA_POD 48858:8080
Forwarding from 127.0.0.1:48858 -> 8080
Forwarding from [::1]:48858 -> 8080
```

    - alpaca-prod 포드중 첫번째 포드 이름을 ALPACA_POD 에 저장
    - 호스트 48858포트를 -> 첫번째 alpaca-prod 포드 8080 포트로 포워딩

쿠버네티스 객체 - 서비스

- 서비스 준비상태 확인
  - ➢ readinessProbe 설정을 통한 포드의 준비상태 확인

```
# kubectl edit deployment/alpaca-prod
..
     containers:
      - image: gcr.io/kuar-demo/kuard-amd64:1
        imagePullPolicy: IfNotPresent
        name: alpaca-prod
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          periodSeconds: 2      ----------> 서비스체크 시간간격
          initialDelaySeconds : 0
          failureThreshold: 3   ----------> 3회이상 체크실패시 서비스 제외
          successThreshold: 1   ----------> 1회이상 성공시 서비스 추가
..
```

    - ○ 2초에 한번씩 체크, 3번실패시 endpoint 삭제, 1번이상 성공시 endpoint 추가

  - ➢ readinessProbe 설정을 통한 포드의 준비상태 확인

```
kubectl get endpoints alpaca-prod --watch
NAME          ENDPOINTS                                                  AGE
alpaca-prod   192.168.233.201:8080,192.168.233.202:8080,192.168.9.72:8080   150m
alpaca-prod   192.168.233.201:8080,192.168.9.72:8080   150m
alpaca-prod   192.168.233.201:8080,192.168.233.202:8080,192.168.9.72:8080   151m
```

쿠버네티스 객체 - 서비스

- 서비스
  - ➢ alpaca-prod 매니페스트 생성

```
# vi alpaca-prod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: alpaca
    env: prod
  name: alpaca-prod
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: alpaca-prod
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

  - ➢ alpaca-prod 포드 생성

```
# kubectl apply -f alpaca-prod.yaml
pod "alpaca-prod" created
```

쿠버네티스 객체 - 서비스

- 서비스
  - ➢ alpaca-svc 서비스 매니페스트 생성

```
# vi alpaca-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: alpaca-svc
spec:
  selector:
    app: alpaca
  ports:
    - port: 80       ----------> 클러스터 IP 접속 포트
      protocol: TCP   --------> 서비스 접속 프로토콜
      targetPort: 8080  ------> 포드 애플리케이션 접속 포트
      nodePort: 30030  ------> 서비스 접속을 위한 노드 접속 포트
  type: NodePort
```

  - ➢ alpaca-svc 서비스 생성

```
# kubectl apply -f alpaca-prod.yaml
pod "alpaca-prod" created
```

  - ➢ alpaca-svc 서비스 확인

```
# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
alpaca-svc   NodePort    10.97.120.107   <none>        80:30030/TCP   6s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        27d
```

쿠버네티스 객체 - 서비스

- 서비스
  - ➢ 멀티포트 사용 서비스 매니페스트

```
# vi alpaca-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: alpaca-svc
spec:
  selector:
    app: alpaca
  ports:
    - name: http   ----------> 포트명
      port: 80     ----------> 클러스터 IP 접속 포트
      protocol: TCP  --------> 서비스 접속 프로토콜
      targetPort: 8080 ------> 포드 애플리케이션 접속 포트
      nodePort: 30030 ------> 서비스 접속을 위한 노드 접속 포트
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8082
      nodePort: 30040
type: NodePort
```

쿠버네티스 객체 - 서비스

- 서비스
  - ExternalName 타입 설정 매니페스트 (도메인명 사용)

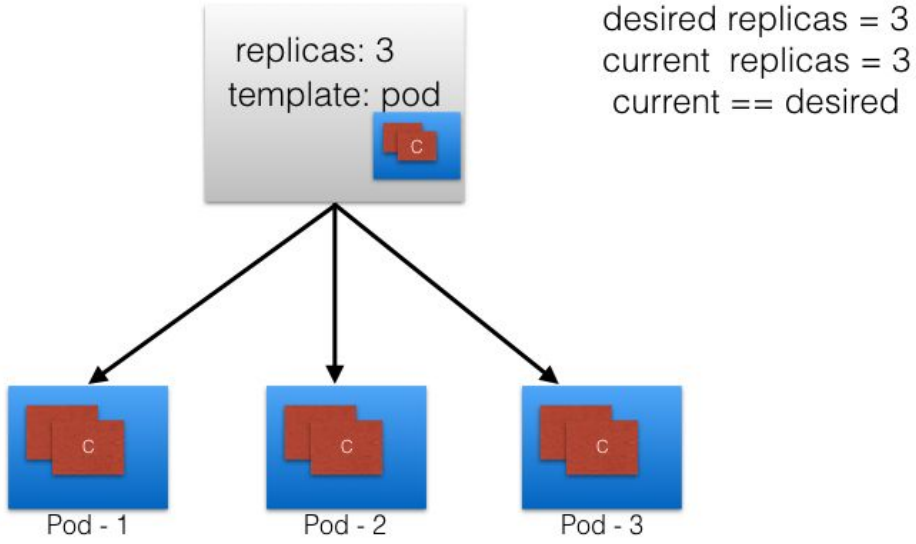
```
apiVersion: v1
kind: Service
apiVersion: v1
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

  - ExternalName 타입 설정 매니페스트 (IP 사용)

```
apiVersion: v1
kind: Service
metadata:
  name: hello-node-svc
spec:
  selector:
    app: hello-node
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
  externalIPs:
  - 80.11.12.11
```

쿠버네티스 객체 - 리플리카

- 리플리카 컨트롤러
  - 포드 복제 집합을 쉽게 생성하고, 상시 관리 모니터링 하는 포드 관리자 역할 수행
  - 포드 복제 집합을 논리적 객체로 정의 하고 문제 발생시 자동 생성 및 스케줄링
  - 복제 생성할 포드는 선택기와 템플릿, 복제할 포드 개수 설정값을 참조
  - 복제된 포드는 로드밸런서를 통해 분산된 트래픽 요청을 처리함



참조 : https://sites.google.com/site/edxkubernetes/kubernetes-building

쿠버네티스 객체 - 리플리카

- 리플리카 컨트롤러
  - ➢ kuard-rc 리플리카 컨트롤러 매니페스트

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: kuard-rc
spec:
  replicas: 2      ----------> 복제할 포드 개수
  selector :
    app : kuard    ----------> 선택기에 복제할 포드 라벨 지정
  template:
    metadata:                                          ┐
      labels:                                          │
        app: kuard                                     │
        version: "2"                                   │
    spec:                                              │
      containers:                                      ├ 복제 생성할 포드의 템플릿 정의
        - name: kuard                                  │
          image: "gcr.io/kuar-demo/kuard-amd64:2"      │
          ports:                                       │
          - containerPort: 80                          ┘
```

  - ➢ kuard-rc 리플리카 컨트롤러 생성

```
# kubectl apply -f kuard-rc.yaml
replicationcontroller "kuard-rc" created
```

쿠버네티스 객체 - 리플리카

- 리플리카 컨트롤러
  - ➢ kuard-rc 리플리카 생성 확인

```
# kubectl get rc -o wide
NAME       DESIRED   CURRENT   READY     AGE       CONTAINERS   IMAGES                              SELECTOR
kuard-rc   2         2         2         37s       kuard        gcr.io/kuar-demo/kuard-amd64:2      app=kuard
```

  - ➢ kuard-rc 리플리카 컨트롤러 생성

```
# kubectl get pods -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP            NODE
kuard-rc-dnpgj   1/1       Running   0          1m        10.244.2.25   kube-node2
kuard-rc-xrhsh   1/1       Running   0          1m        10.244.1.31   kube-node1
```

  - ➢ kuard-rc 리플리카 컨트롤러 포드 개수 확장

```
# vi kuard-rc.yaml
..
spec:
  replicas: 3      ----------> 복제할 포드 개수 변경
..
```

  - ➢ kuard-rc 리플리카 컨트롤러 변경 설정 적용

```
# kubectl apply -f kuard-rc.yaml
replicationcontroller "kuard-rc" configured
```

쿠버네티스 객체 - 리플리카

- 리플리카 컨트롤러
  - ➢ scale 명령을 이용한 kuard-rc 리플라카 컨트롤러 포드 개수 확장

```
# kubectl scale rc kuard-rc --replicas 4
replicationcontroller "kuard-rc" scaled
```

  - ➢ kuard-rc 리플리카 컨트롤러 확인

```
# kubectl get rc
NAME       DESIRED   CURRENT   READY     AGE
kuard-rc   4         4         4         4m
```

  - ➢ kuard-rc 리플리카 컨트롤러 포드 확인

```
# kubectl get pods -o wide
NAME                       READY     STATUS    RESTARTS   AGE       IP            NODE
kuard-rc-j9xwx             1/1       Running   0          5m        10.244.1.41   kube-node1
kuard-rc-rh2p5             1/1       Running   0          5m        10.244.1.42   kube-node1
kuard-rc-swzcd             1/1       Running   0          5m        10.244.2.36   kube-node2
kuard-rc-v7lj7             1/1       Running   0          5m        10.244.2.35   kube-node2
```

쿠버네티스 객체 - 리플리카

- 리플리카셋
  - 리플리카 컨트롤러의 향상된 버전
  - 복제할 포드의 선택기 정의를 더 자세히 설정 가능 (Set 기반 선택기 설정)

➢ kuard-rs 리플리카셋 매니페스트1

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: kuard-rs
spec:
  replicas: 3
  selector:
     matchLabels:
       app: kuard
  template:
    metadata:
      labels:
        app: kuard
        version: "2"
..
```

  - matchLabels 옵션을 사용

쿠버네티스 객체 - 리플리카

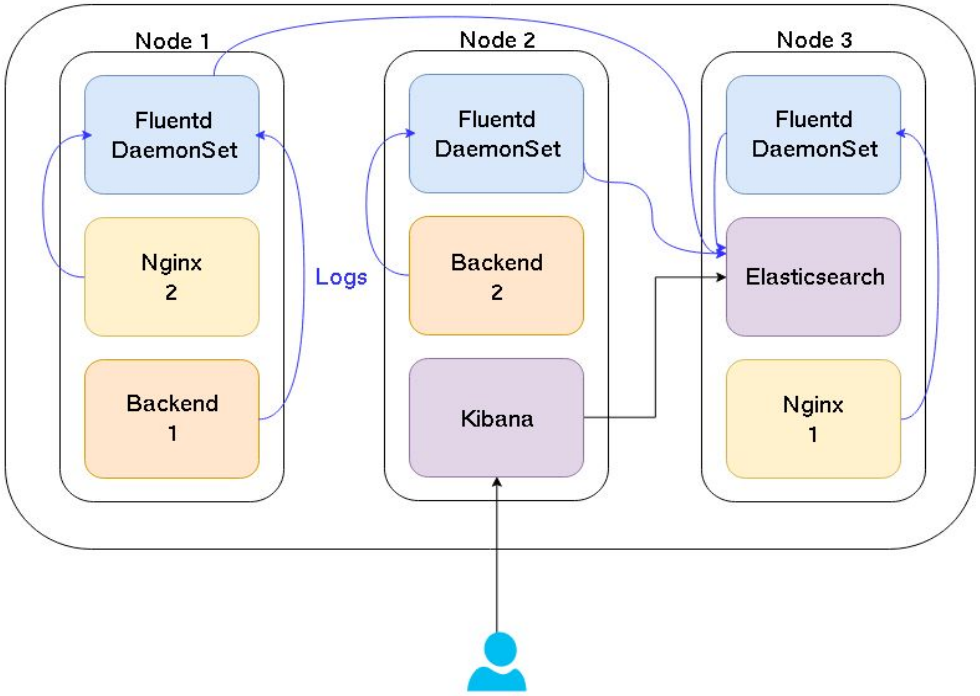
- 리플리카셋
  - ➢ kuard-rs 리플리카셋 매니페스트2

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: kuard-rs
spec:
   replicas: 3
   selector:
     matchExpressions:
      - {key: app, operator: In, values: [kuard, kuard-test, kuard-prod]}
      - {key: version, operator: NotIn, values: [1]}
  template:
    metadata:
      labels:
        app: kuard
        version: "2"
..
```

  - ○ matchExpressions 사용시 선택기 연산자를 사용해 자세한 설정 가능
  - ○ In, NotIn, Exist, DoesNotExist 연자사를 사용

쿠버네티스 객체 - 데몬셋

- 데몬셋
  - 클러스터의 모든 단일노드에 포드를 복제할때 사용
  - 로그 수집기나 모니터링 에이전트와 같은 포드를 모든 노드에 복제 사용시 용이함
  - 노드별로 하나의 포드가 동작해야 할 때 사용



Fluentd 와 Elasticsearch 를 이용한 로그 수집 및 모니터링 환경 구성

쿠버네티스 객체 - 데몬셋

- 데몬셋
  - ➢ fluentd-ds 데몬셋 매니페스트 생성

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-ds
  namespace: kube-system
  labels:
    app: fluentd
spec:
  template:
    metadata:
      labels:
        app: fluentd
    spec:
      containers:
      - name: fluentd
        image: fluent/fluentd:v0.14.10
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

```
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

쿠버네티스 객체 - 데몬셋

- 리플리카 데몬셋
  - ➢ fluentd-ds 데몬셋 생성

```
# kubectl apply -f fluentd-ds.yaml
daemonset.extensions "fluentd-ds" created
```

  - ➢ fluentd-ds 데몬셋 생성 확인

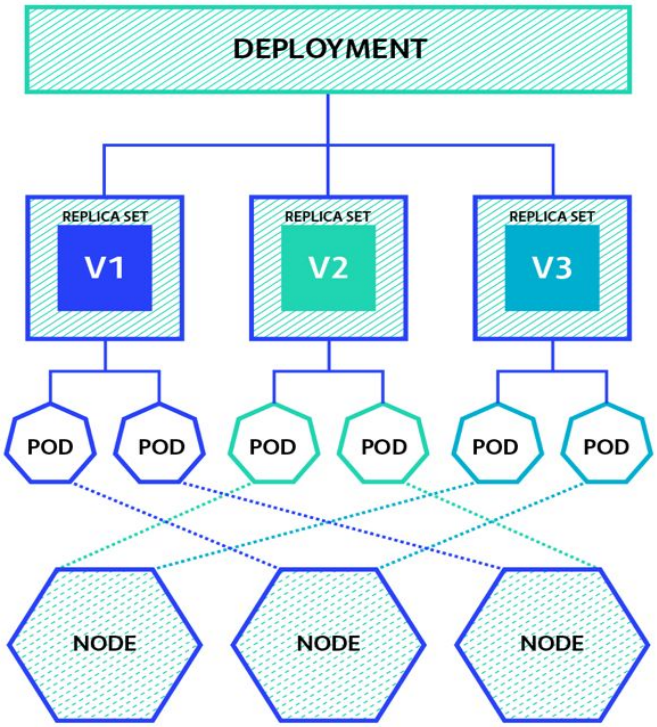
```
# kubectl get ds --all-namespaces
NAMESPACE     NAME              DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR                   AGE
kube-system   fluentd-ds        2         2         2         2            2           <none>                          25m
kube-system   kube-flannel-ds   3         3         3         3            3           beta.kubernetes.io/arch=amd64   32d
kube-system   kube-proxy        3         3         3         3            3           <none>                          32d
```

  - ➢ fluentd-ds 포드 복제본 생성 확인

```
# kubectl get pods --all-namespaces
NAMESPACE     NAME                          READY     STATUS    RESTARTS   AGE
..
kube-system   kube-flannel-ds-84m45         1/1       Running   14         32d
kube-system   kube-flannel-ds-fnbwk         1/1       Running   8          32d
kube-system   kube-flannel-ds-wm45d         1/1       Running   6          31d
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트
  - 애플리케이션 배포를 위한 컨트롤러, 쿠버네티스 1.2 버전부터 지원
  - 새로운 버전의 애플리케이션을 서비스 중단 및 오류 없이 배포 가능하도록 지원
  - 디플로이먼트는 리플리카셋을 관리, 라벨과 선택기를 통해 리플케이션 객체를 확인



쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트
  - node-js-deploy 디플로이먼트 매니페스트

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: node-js-deploy
labels:
  name: node-js-deploy
spec:
  replicas: 1
template:
  metadata:
  labels:
    name: node-js-deploy
  spec:
    containers:
    - name: node-js-deploy
    image: jonbaier/pod-scaling:0.1
    ports:
    - containerPort: 80
```

  - node-js-deploy 디플로이먼트 생성

```
# kubectl apply -f node-js-deploy.yaml
deployment.extensions "node-js-deploy" created
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트
  - ➢ node-js-deploy 디플로이먼트 생성확인

```
# kubectl get deploy
NAME             DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
node-js-deploy   1         1         1            1           9h
```

  - ➢ node-js-deploy 리플리카셋 생성확인

```
# kubectl get rs
NAME                        DESIRED   CURRENT   READY     AGE
node-js-deploy-764fd87fd7   1         1         1         9h
```

  - ➢ node-js-deploy 포드 생성확인

```
# kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
node-js-deploy-764fd87fd7-nh88v   1/1       Running   0          9h
```

  - ➢ scale 명령을 이용한 node-js-deploy 디플로이먼트 확장

```
# kubectl scale deployment node-js-deploy --replicas 3
deployment.extensions "node-js-deploy" scaled
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 업데이트
  - ➢ node-js-deploy 디플로이먼트 현재 이미지 확인

```
# kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
node-js-deploy-764fd87fd7-nh88v   1/1       Running   0          10h

# kubectl describe pod node-js-deploy-764fd87fd7-nh88v | grep Image
    Image:          jonbaier/pod-scaling:0.1
..
```

    - ○ 이미지 버전: jonbaier/pod-scaling:0.1
  - ➢ node-js-deploy 배포 이미지 업데이트 실행

```
# kubectl set image deployment node-js-deploy node-js-deploy=jonbaier/pod-scaling:0.2
deployment.apps "node-js-deploy" image updated
```

    - ○ 이미지 업데이트: jonbaier/pod-scaling:0.1 -> jonbaier/pod-scaling:0.2
  - ➢ rollout 명령을 이용한 업데이트 실행 결과 확인

```
# kubectl rollout status deployment node-js-deploy
deployment "node-js-deploy" successfully rolled out
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 업데이트
  - ➢ node-js-deploy 디플로이먼트 업데이트 매니페스트 수정

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: node-js-deploy
labels:
  name: node-js-deploy
spec:
  replicas: 3
template:
  metadata:
  labels:
    name: node-js-deploy
  spec:
    containers:
    - name: node-js-deploy
    image: jonbaier/pod-scaling:0.2
    ports:
    - containerPort: 80
```

  - ➢ node-js-deploy 디플로이먼트 업데이트 실행

```
# kubectl apply -f node-js-deploy-update.yaml
deployment.extensions "node-js-deploy" configured
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 롤백
  - ➢ node-js-deploy 디플로이먼트 롤백을 위한 리플리카셋 확인

```
# kubectl get rs -o wide
NAME                         DESIRED   CURRENT   READY     AGE       CONTAINERS       IMAGES
node-js-deploy-764fd87fd7    0         0         0         10m       node-js-deploy   jonbaier/pod-scaling:0.1 ..
node-js-deploy-f87d6fbb      3         3         3         7m        node-js-deploy   jonbaier/pod-scaling:0.2 ..
```

  - ➢ node-js-deploy 디플로이먼트 롤아웃 히스토리 확인

```
# kubectl rollout history deployment node-js-deploy
deployments "node-js-deploy"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

  - ➢ Revision 1 로 롤백 실행

```
# kubectl rollout undo deployment node-js-deploy --to-revision 1
deployment.apps "node-js-deploy"
```

쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 롤백
  - ➢ 롤백을 실행결과 확인 위한 리플리카셋 확인

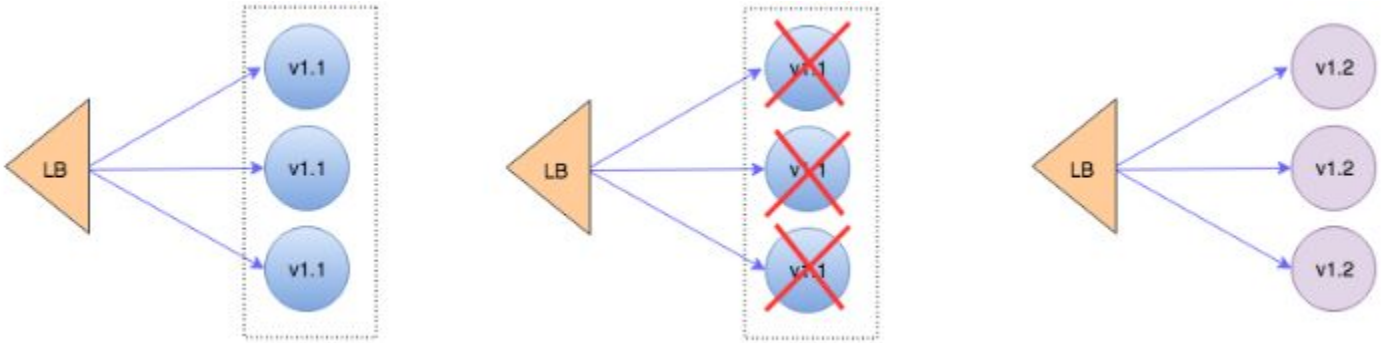
```
# kubectl get rs -o wide
NAME                          DESIRED   CURRENT   READY     AGE       CONTAINERS       IMAGES
node-js-deploy-764fd87fd7     3         3         3         16m       node-js-deploy   jonbaier/pod-scaling:0.1
node-js-deploy-f87d6fbb       0         0         0         14m       node-js-deploy   jonbaier/pod-scaling:0.2
```

  - ➢ node-js-deploy 디플로이먼트 롤아웃 히스토리 확인

```
# kubectl rollout history deployment node-js-deploy
deployments "node-js-deploy"
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

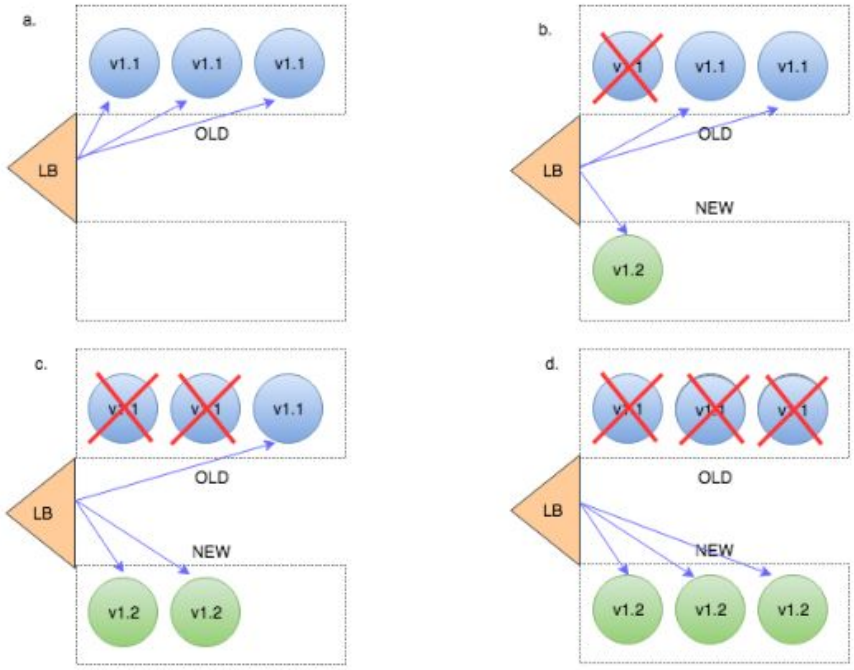
쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 업데이트 전략
  - 애플리케이션 버전 변경을 위한 전략 설정
    - 재생성 전략 (Recreate)
      디플로이먼트와 관련된 모든 포드를 중지하고, 새로운 버전의 포드를 생성
      빠르고 간편한 반면, 서비스 다운타임이 발생
      다운타임이 허용되는 테스트 배포에 적합



쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 업데이트 전략
  - 애플리케이션 버전 변경을 위한 전략 설정
    - 롤링업데이트 전략 (RollingUpdate)
      한번에 몇 개씩 포드를 업데이트 하는 방식으로, 다운타임 없이 업데이트 가능
      재성성 전략 보다 느리지만, 정교하고 안정적
      업데이트 되는 동안 버전차이로 인한 서비스 중단이 없도록 구성해야 함



쿠버네티스 객체 - 디플로이먼트

- 디플로이먼트 업데이트 전략

➢ 재생성 전략 (Recreate) 매니페스트

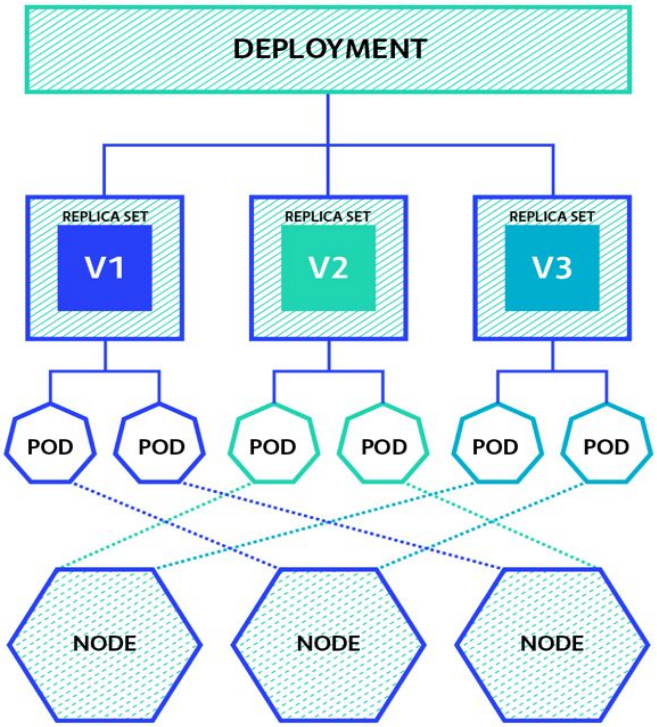
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: node-js-deploy
labels:
  name: node-js-deploy
spec:
  replicas: 3
  strategy:
    type: Recreate
template:
  metadata:
  labels:
    name: node-js-deploy
  spec:
    containers:
    - name: node-js-deploy
    image: jonbaier/pod-scaling:0.2
    ports:
    - containerPort: 80
```

➢ 롤링업데이트 전략 (RollingUpdate) 매니페스트

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: node-js-deploy
labels:
  name: node-js-deploy
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
template:
  metadata:
  labels:
    name: node-js-deploy
  spec:
    containers:
    - name: node-js-deploy
    image: jonbaier/pod-scaling:0.2
    ports:
    - containerPort: 80
```

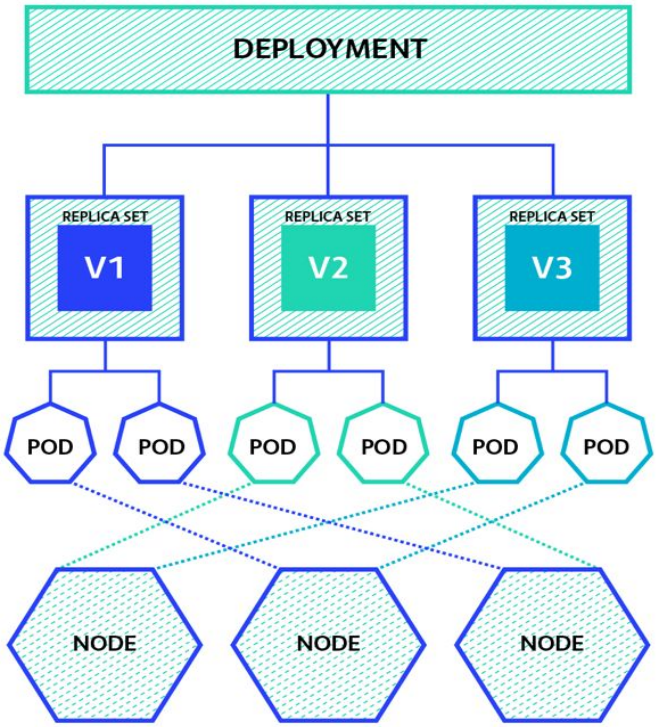
쿠버네티스 객체 - 잡

- 디플로이먼트
  - 애플리케이션 배포를 위한 컨트롤러, 쿠버네티스 1.2 버전부터 지원
  - 새로운 버전의 애플리케이션을 서비스 중단 및 오류 없이 배포 가능하도록 지원
  - 디플로이먼트는 리플리카셋을 관리, 라벨과 선택기를 통해 리플케이션 객체를 확인



쿠버네티스 객체 - 컨피그맵

- 디플로이먼트
  - 애플리케이션 배포를 위한 컨트롤러, 쿠버네티스 1.2 버전부터 지원
  - 새로운 버전의 애플리케이션을 서비스 중단 및 오류 없이 배포 가능하도록 지원
  - 디플로이먼트는 리플리카셋을 관리, 라벨과 선택기를 통해 리플케이션 객체를 확인



쿠버네티스 객체 - 퍼시스턴트 스토리지

- 퍼시스턴트 스토리지
  - 컨테이너의 쓰기가능 레이어 와 도커 볼륨은 컨테이너 수명을 뛰어넘지 못함
  - 쿠버네티스는 데이터 저장을 위한 영구(Persistent) 볼륨을 제공 가능
  - 쿠버네티스 포드로 제공된 볼륨은 동일한 포드내에서 데이터 공유가 가능
- 퍼시스턴트 스토리지 볼륨 타입
  - 임시디스크

    노드 머신 자체 스토리지 볼륨이나 RAM 디스크 옵션과 함께 사용

    포드가 생성될때 생성되고, 포드가 삭제 될때 함께 삭제되는 임시볼륨
  - nfs
  - ceph
  - gitrepo

    git 저장소를 새로운 빈 폴더로 복제 후 사용
  - 클라우드 볼륨

    퍼블릭 클라우드 업체가 제공하는 볼륨을 사용

쿠버네티스 객체 - 퍼시스턴트 스토리지

- 퍼시스턴트 볼륨
  - 쿠버네티스에서 영구 볼륨을 제공하기 위한 객체
  - 포드 생성시 퍼시스턴트클래임에 의해 볼륨을 정의, 포드와 연결 된다.
  - 포드가 삭제 되더라도, 관리자가 직접 삭제 하지않으면 삭제되지 않는다.
- 퍼시스턴트 스토리지 볼륨 타입
  - 임시디스크

    노드 머신 자체 스토리지 볼륨이나 RAM 디스크 옵션과 함께 사용

    포드가 생성될때 생성되고, 포드가 삭제 될때 함께 삭제되는 임시볼륨
  - nfs
  - ceph
  - gitrepo

    git 저장소를 새로운 빈 폴더로 복제 후 사용
  - 클라우드 볼륨

    퍼블릭 클라우드 업체가 제공하는 볼륨을 사용

수고하셨습니다.