Docker 조금 더 활용하기

2015. 10. 11

㈜ 오픈소스컨설팅

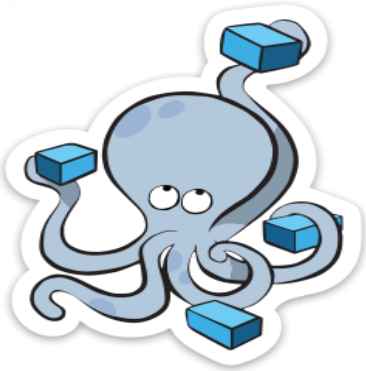
김호진

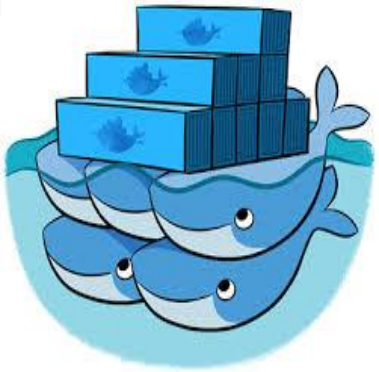
Contents

Contents

intro

- Docker의 개념
- 그리고 locally application을 만들고, 이를 cloud에 올리는 방법
- Compose, Machine and Swarm을 이용한 서비스 제공





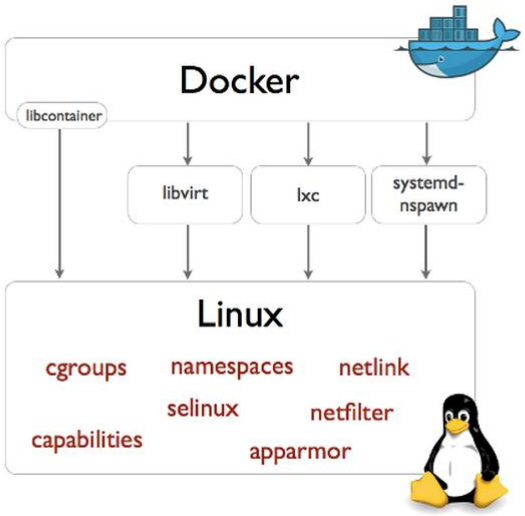
클라우드 속의 Docker

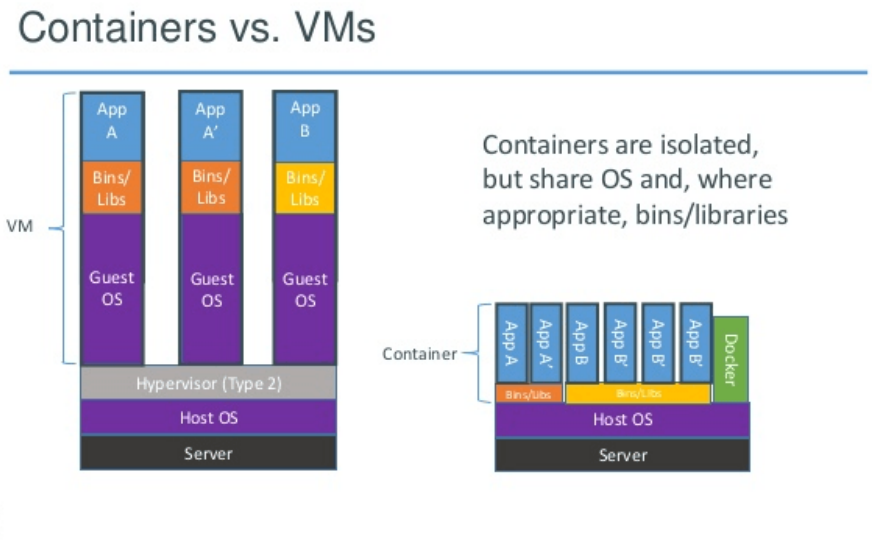
IBM cloud and open technologies

출처: Animesh Singh - http://www.slideshare.net/AnimeshSingh/cloud-foundry-docker-openstack-leading-open-source-triumvirate

Docker internal

- "Docker"는 미국 Docker 사 (구 dotCloud)가 개발 한 오픈 소스 컨테이너 관리 소프트웨어의 하나. Go 언어로 구현
- 경량 가상화 환경인 컨테이너를 제공하고 응용 프로그램 실행 환경을 그대로 Docker 이미지로 저장 가능
- 가상 서버가 아닌 애플리케이션에 최적화되어 있기 때문에 원칙적으로 하나의 컨테이너에 하나의 응용 프로그램이 실행 구성
- 원래 Docker는 LXC를 통해 Linux 컨테이너 기능을 이용. Docker Ver0.9보다 libcontainer라는 독자적인 라이브러리를 통하도록 개선. 옵션으로 LXC를 선택 가능





출처 : Steven J. Vaughan-Nichols - Docker libcontainer unifies Linux container powers

Docker internal – linux container

Linux Container

- **Linux 커널이 가진 기능**을 이용하여 구축 된 컨테이너. 줄여서, LXC로 표기

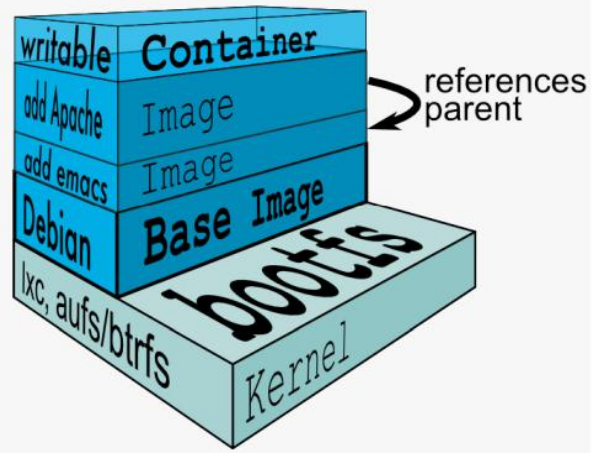
- Namespace
  - 마운트 호스트 도메인 이름, 프로세스 공간 소켓이나 메시지 큐 공간, 사용자 그룹, 네트워크 등의 이름 공간마다 식별 은폐 할 수 있는 기능
- Cgroup
  - "Control Group"의 약자. 프로세스를 그룹화하고 그룹 내에 존재하는 프로세스에 대해 그룹 단위의 자원 관리를 할 수있는 기능.
- Chroot
  - 루트 디렉토리를 특정 디렉토리에 변경할 수있는 기능.
- Netns
  - (Linux Network Namespace)은 호스트 서버에 가상으로 브리지 "br0"며 "veth"라는 직결 된 가상 NIC의 쌍으로 이루어진 가상 네트워크를 구축한다.
  - 컨테이너마다 별도의 veth가 생성되어 그 한쪽을 컨테이너 내부의 이더넷 "eth0"로 할당한다.

Docker internal - AUFS

- AUFS (Another Union FS)은 계층 구조의 파일 시스템
- 부모의 파일 시스템을 read-only로 그 위에 쓰기 가능한 계층을 추가한다. 이렇게 여러 레이어를 쌓아 레이어를 통해 하나의 파일 시스템과 같이 취급한다.
- 부모 컨테이너에서 차등 관리 할 수 있기 때문에 컨테이너 이미지의 용량을 경량화 할 수 있다.

Containers, images & AUFS

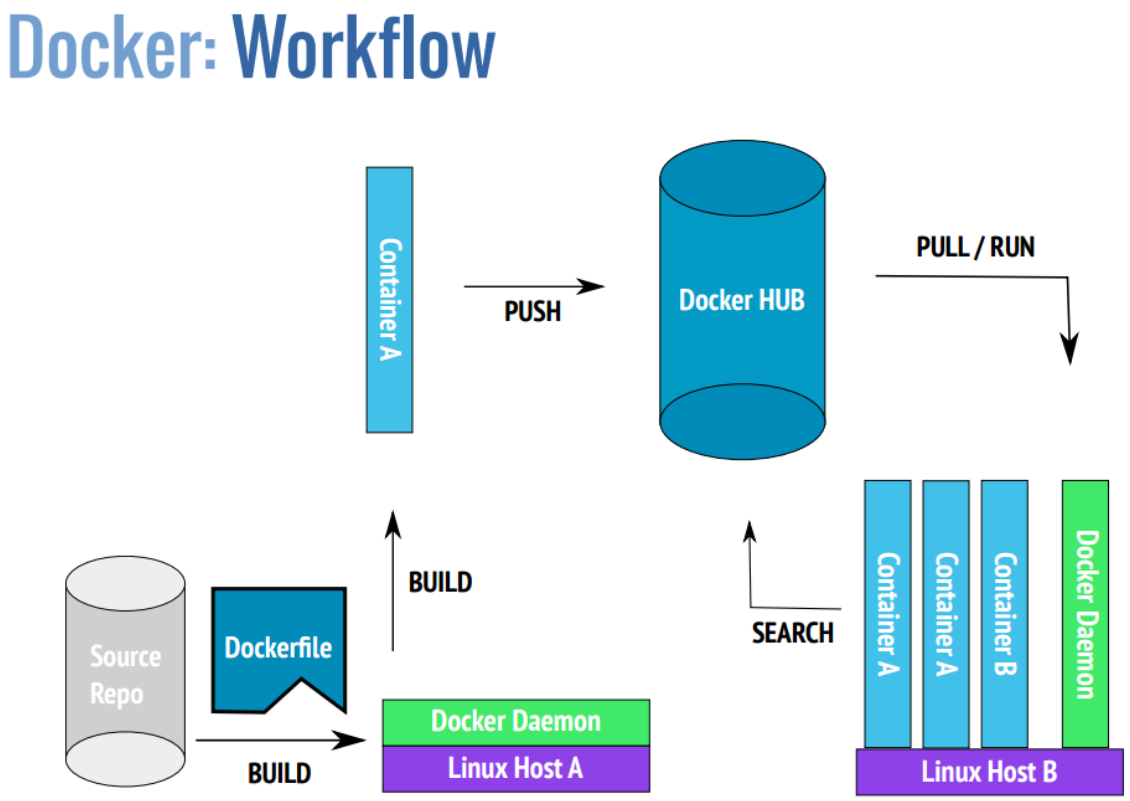
1. RO images depends on parent images
2. Add AUFS RW layer
3. All layers + meta is a container



출처 : Colin Surprenant - Docker

Docker internal - Docker image & workflow

- Docker 이미지는 Docker 컨테이너 생성의 기반- ubuntu이나 centos 등의 배포가 이미지화되어 있음.
- Docker 이미지를 기반으로 개별 응용 프로그램의 환경을 설치하여 만들 컨테이너 실행 환경.

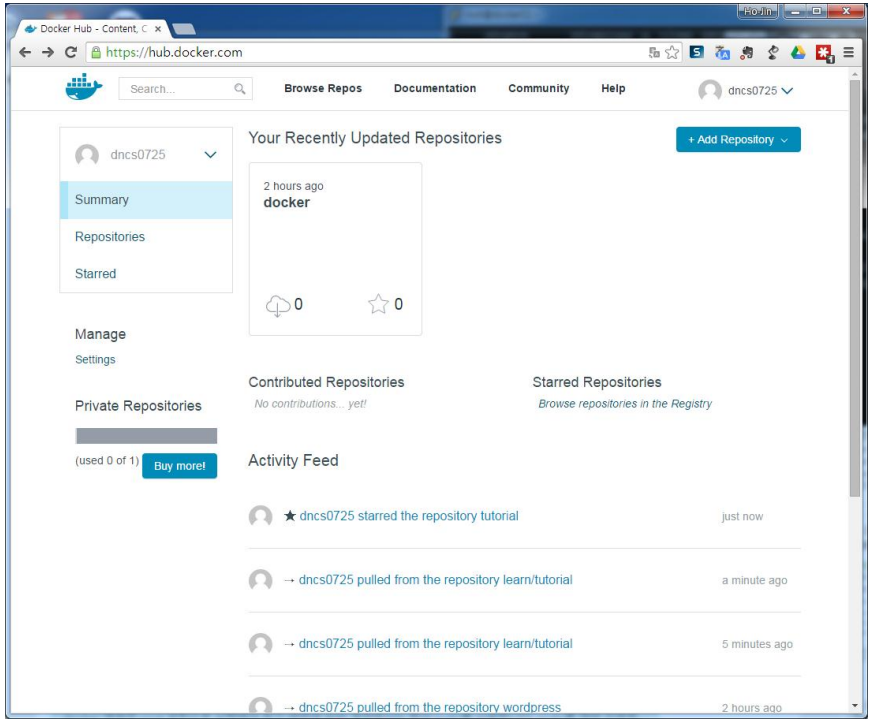


출처 : Andrea Laspada - http://www.slideshare.net/Pokerone/docker-45628208

Docker - Getting Started with Docker Hub.

Docker Hub

- 먼저 docker site에 가입 (history 관리)
- Hub.docker.com
- Docker 사는 Docker Hub라는 클라우드 저장소를 제공
- 다양한 단체와 개인이 만든 Docker 이미지를 이용하거나 직접 만든 이미지를 공유.




```
[root@Tiger ~]# docker login
Username: oscinfra
Password:
Email: khoj@osci.kr
Login Succeeded
```

Docker - Installing Docker on CentOS 7

2015 년 10월 현재

- Hostname setting
- Docker install / Docker start 등록 / 최신버전 update

```
#docker installation/setting
yum -y install update > /dev/null ; yum -y install docker; systemctl enable docker;
systemctl start docker; systemctl status docker | grep Active

# check the version
[root@centos7 ~]# docker version
Client:
 Version:      1.8.2
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   0a8c2e3
 Built:        Wed Oct  7 17:25:19 UTC 2015
 OS/Arch:      linux/amd64

Server:
 Version:      1.8.2
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   0a8c2e3
 Built:        Wed Oct  7 17:25:19 UTC 2015
 OS/Arch:      linux/amd64
```

Docker - Docker 이미지 검색

최신 이미지 검색

- centos 최신 Docker 이미지를 download

```
[root@centos7 ~]# sudo docker pull centos:latest
latest: Pulling from library/centos

47d44cb6f252: Already exists
f6f39725d938: Already exists
f9a8cbc8dd13: Already exists
f37e6a610a37: Already exists
Digest: sha256:2d9573acf37315cb8fe2a1420769c3b83f59d8f286fd8898a580578c0d5e66c6
Status: Image is up to date for centos:latest

[root@centos7 ~]# docker history centos:latest
IMAGE               CREATED             CREATED BY                                      SIZE
0f73ae75014f   4 weeks ago    /bin/sh -c #(nop) CMD ["/bin/bash"]             0 B
f37e6a610a37   4 weeks ago    /bin/sh -c #(nop) LABEL License=GPLv2           0 B
f9a8cbc8dd13   4 weeks ago    /bin/sh -c #(nop) LABEL Vendor=CentOS           0 B
f6f39725d938   4 weeks ago    /bin/sh -c #(nop) ADD file:2c002b8a427ce98fc1   172.3 MB
47d44cb6f252   4 weeks ago    /bin/sh -c #(nop) MAINTAINER The CentOS Proje   0 B
[root@centos7 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
ubuntu              latest              cdd474520b8c        2 days ago      187.9 MB
centos              latest              0f73ae75014f        4 weeks ago     172.3 MB
```

Docker - Docker 이미지 검색

최신 이미지 검색

- ubuntu 최신 Docker 이미지를 download

```
[root@centos7 ~]# sudo docker pull ubuntu:latest
latest: Pulling from library/ubuntu

48731f0a6276: Already exists
9302827ed0a5: Already exists
f03f3645bde1: Already exists
Digest: sha256:40cd91cbcaac863fa2fc0b8a8eaa440565ef5ce498d60177988f6f55ec691d9d
Status: Image is up to date for ubuntu:latest
[root@centos7 ~]# docker history cdd474520b8c
IMAGE               CREATED             CREATED BY
SIZE                COMMENT
cdd474520b8c   2 days ago    /bin/sh -c #(nop) CMD ["/bin/bash"]              0 B
f03f3645bde1   2 days ago    /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/   1.895 kB
9302827ed0a5   2 days ago    /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic   194.5 kB
48731f0a6276   2 days ago    /bin/sh -c #(nop) ADD file:ed2337b3477da68f9b   187.7 MB
[root@centos7 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
ubuntu              latest              cdd474520b8c        2 days ago      187.9 MB
centos              latest              0f73ae75014f        4 weeks ago     172.3 MB
```

Docker - Docker 구축 기초

Docker Command

```
[root@Docker-Host ~]# docker
```

```
Commands:
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders
    create    Create a new contai
    diff      Inspect changes on
    events    Get real time events from the server
    exec      Run a command in a running container
    export    Stream the contents of a container as a tar archive
    history   Show the history of an image
    images    List images
    import    Create a new filesy
    info      Display system-wide
    inspect   Return low-level in
    kill      Kill a running cont
    load      Load an image from a tar archive
    login     Register or log in
    logout    Log out from a Dock
    logs      Fetch the logs of a
    port      Lookup the public-facing port that is NAT-ed to PRIVATE_PORT
    pause     Pause all processes within a container
    ps        List containers
    pull      Pull an image or a repository from a Docker registry server
    push      Push an image or a
    rename    Rename an existing
    restart   Restart a running c
    rm        Remove one or more containers
    rmi       Remove one or more images
    run       Run a command in a
    save      Save an image to a
    search    Search for an image
    start     Start a stopped con
    stats     Display a stream of
    stop      Stop a running cont
    tag       Tag an image into a
    top       Lookup the running
    unpause   Unpause a paused co
    version   Show the Docker ver
    wait      Block until a conta

Run 'docker COMMAND --help' for more information on a command.
```

Dockerfile 을 통하여 이미지를 생성

Docker 이미지를 컨트롤

실행중인 컨테이너 강제종료

컨테이너 리스트를 확인

rm : 컨테이너 삭제
rmi : 이미지 삭제
run : 컨테이너에서 실행할 명령어

컨테이너 시작

컨테이너 정지

```
Usage: docker ps [OPTIONS]

List containers

  -a, --all=false
  --before=
  -f, --filter=[]
  --help=false
  -l, --latest=false
  -n=-1
  --no-trunc=false
  -q, --quiet=false
  -s, --size=false
  --since=
```

Docker - Docker 컨테이너 생성 + 시작

- Docker은 가상 서버가 아닌 애플리케이션에 최적화되어 있기 때문에 원칙적으로 하나의 컨테이너에 하나의 응용 프로그램이 실행 구성
- centos 최신 버전 이미지에서 centos01라는 이름으로 컨테이너를 작성하고 응용 프로그램에 bash를 지정하여 실행
- 즉, Docker은 docker run에서 만든 컨테이너에서이 명령에서 지정하는 하나의 애플리케이션을 실행
- 지정한 응용 프로그램이 exit 함과 동시에 그 Docker 컨테이너도 정지
- 즉, Docker 컨테이너의 bash를 exit하면 컨테이너 자체도 종료

```
[root@centos7 ~]# docker run -it --name ubuntu01 ubuntu /bin/bash

root@dbe8896b758a:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="14.04.3 LTS, Trusty Tahr"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 14.04.3 LTS"

[root@centos7 ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
dbe8896b758a        ubuntu              "/bin/bash"         2 minutes ago       Up 2
minutes                             ubuntu01
```

Docker - Docker 컨테이너 생성 + 시작

- centos 최신 버전 이미지에서 centos01라는 이름으로 컨테이너를 작성하고 응용 프로그램에 bash를 지정하여 실행
- Docker 컨테이너의 bash를 exit하면 컨테이너 자체도 종료

```
[root@centos7 ~]# docker run -i -t --name centos01 centos:latest /bin/bash
[root@a51a1f6629aa /]# cat /etc/redhat-release
CentOS Linux release 7.1.1503 (Core)
[root@a51a1f6629aa /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
15: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:05 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.5/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:5/64 scope link
       valid_lft forever preferred_lft forever
[root@centos7 ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
a51a1f6629aa        centos:latest       "/bin/bash"         About a minute ago   Up
About a minute                          centos01
```

Docker - Docker 컨테이너 분리 +연결 + 재시작

- 일시 분리
- 다시 연결
- 기존 컨테이너 centos01를 다시 시작

```
[root@centos7 ~]# docker run -i -t --name centos01 centos:latest /bin/bash
[root@a51a1f6629aa /]# [Ctrl + p] → [Ctrl + q]
[root@centos7 ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
a51a1f6629aa        centos:latest       "/bin/bash"              16 minutes ago
Up 16 minutes                                 centos01
[root@centos7 ~]# docker attach a51a1f6629aa

[root@a51a1f6629aa /]#
```

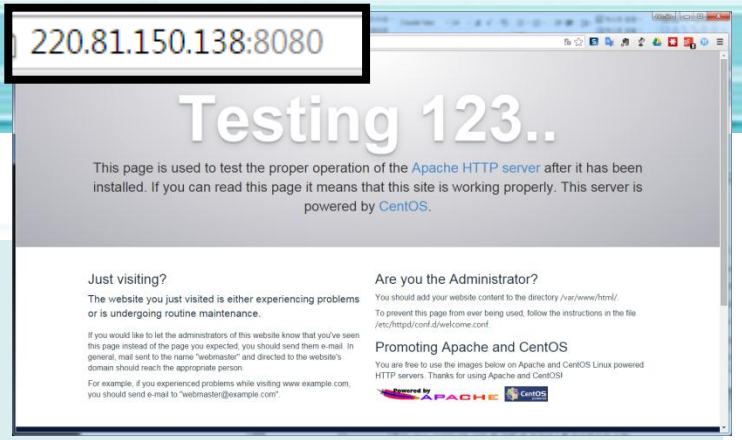
```
[root@centos7 ~]# docker attach a51a1f6629aa
[root@a51a1f6629aa /]# exit
exit
[root@centos7 ~]# docker ps -a | grep centos
a51a1f6629aa        centos:latest       "/bin/bash"              20 minutes ago
Exited (0) 9 seconds ago                      centos01
[root@centos7 ~]# docker start -i centos01
[root@a51a1f6629aa /]#
```

Docker - Docker 컨테이너 관리

- 실행중인 centos01 컨테이너를 중지
- Docker 컨테이너 삭제

```
[root@a51a1f6629aa /]# [root@centos7 ~]#
[root@centos7 ~]# docker ps | grep centos
a51a1f6629aa        centos:latest       "/bin/bash"              25 minutes ago
Up 4 minutes                               centos01
[root@centos7 ~]# docker stop centos01
centos01
[root@centos7 ~]# docker images | grep centos
centos              latest              0f73ae75014f        4 weeks ago         172.3
MB
[root@centos7 ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                          PORTS                     NAMES
a51a1f6629aa        centos:latest       "/bin/bash"              26 minutes ago
Exited (137) 19 seconds ago                                   centos01

[root@centos7 ~]# docker rm centos01
centos01
```

Docker - Docker 컨테이너 관리

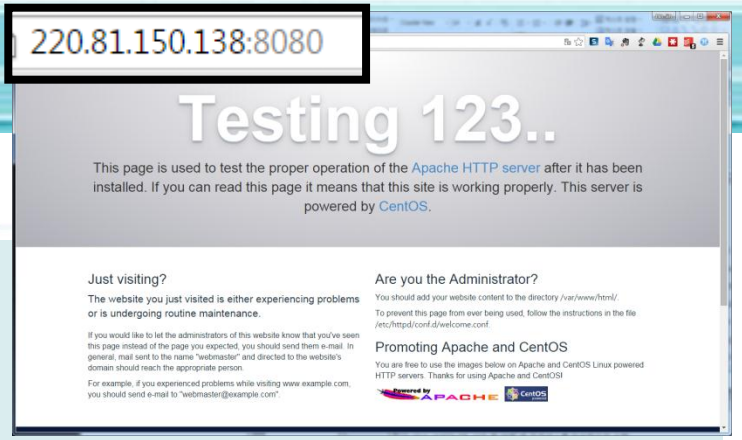


Docker 이미지 만들기

- centos03 컨테이너를 생성하고 bash에서 시작
- centos03 컨테이너에 httpd를 설치
- Ctrl + p + q에서 Docker 컨테이너를 분리
- Oscinfra/httpd라는 이미지 생성
- Oscinfra/httpd라는 이미지를 가지고 Web01이라는 컨테이너를 실행

```
[root@centos7 ~]# docker run -it --name centos03 centos:latest /bin/bash
[root@04e2e6423b12 /]# yum install -y httpd
….
Installed:
  httpd.x86_64 0:2.4.6-31.el7.centos.1
[root@04e2e6423b12 /]# [root@centos7 ~]# docker stop centos03
[root@centos7 ~]# docker commit centos03 oscinfra/httpd
72c1fec3a9bd9b0e7ed8db4e7d69c90cee6339b158847963f501eb180b2cba40
[root@centos7 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
oscinfra/httpd      latest              72c1fec3a9bd        13 seconds ago      278.8
MB
centos              latest              0f73ae75014f        4 weeks ago         172.3
MB
[root@centos7 ~]# docker run -d -p 8080:80 --name web01 oscinfra/httpd /usr/sbin/httpd
-D FOREGROUND
be20ff0d21b0886cb1303a6b0e2eedc805297314dfd644015d83094a82fdc943
```

Docker - Docker 컨테이너 관리

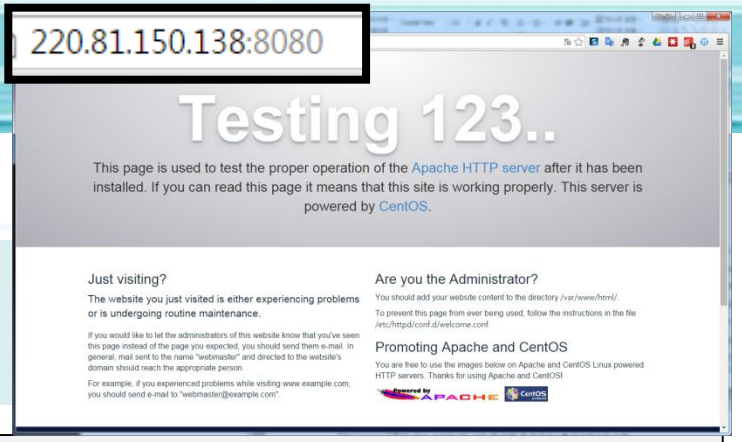


Docker 이미지 실행

- centos03 컨테이너를 생성하고 bash에서 시작
- centos03 컨테이너에 httpd를 설치
- Ctrl + p + q에서 Docker 컨테이너를 분리
- Oscinfra/httpd라는 이미지 생성
- Oscinfra/httpd라는 이미지를 가지고 Web01이라는 컨테이너를 실행

```
[root@centos7 ~]# docker run -it --name centos03 centos:latest /bin/bash
[root@04e2e6423b12 /]# yum install -y httpd
….
Installed:
  httpd.x86_64 0:2.4.6-31.el7.centos.1
[root@04e2e6423b12 /]# [root@centos7 ~]# docker stop centos03
[root@centos7 ~]# docker commit centos03 oscinfra/httpd
72c1fec3a9bd9b0e7ed8db4e7d69c90cee6339b158847963f501eb180b2cba40
[root@centos7 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
VIRTUAL SIZE
oscinfra/httpd      latest              72c1fec3a9bd        13 seconds ago      278.8
MB
centos              latest              0f73ae75014f        4 weeks ago         172.3
MB
[root@centos7 ~]# docker run -d -p 8080:80 --name web01 oscinfra/httpd /usr/sbin/httpd
-D FOREGROUND
be20ff0d21b0886cb1303a6b0e2eedc805297314dfd644015d83094a82fdc943
```

Docker - Docker 컨테이너 관리



Web01 컨테이너 연결

- 백그라운드에서 실행중인 web01 컨테이너에 연결
- 백그라운드에서 실행중인 web01 컨테이너를 중지

```
[root@centos7 ~]# docker exec -it web01 /bin/bash
[root@be20ff0d21b0 /]# ps -ef | grep httpd
root          1      0  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache        5      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache        6      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache        7      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache        8      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache        9      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache       10      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache       11      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
apache       12      1  0 19:31 ?        00:00:00 /usr/sbin/httpd -D FOREGROUND
root         29     13  0 19:38 ?        00:00:00 grep --color=auto httpd
[root@be20ff0d21b0 /]# [root@centos7 ~]#
[root@centos7 ~]# docker stop web01
web01
```

Docker - Docker Hub에서 이미지 공유

백그라운드에서 실행중인 web01 컨테이너에 연결

- Docker Hub에 로그인
- Docker Hub에 방금 만든 “oscinfra / httpd"이미지를 PUSH

```
[root@centos7 ~]# docker login
Username: oscinfra
Password:
Email: khoj@osci.kr
WARNING: login credentials saved in /root/.docker/config.json
Login Succeeded
[root@centos7 ~]# docker push oscinfra/httpd
The push refers to a repository [docker.io/oscinfra/httpd] (len: 1)

72c1fec3a9bd: Buffering to Disk
```

Docker - Dockerfile

```
[root@centos7 httpd]# cat Dockerfile


FROM centos:latest
MAINTAINER oscinfra<khoj@osci.kr>
RUN yum install -y httpd
EXPOSE 80
CMD ["/usr/sbin/httpd" "-D" "FOREGROUND"]
```

```
[root@centos7 httpd]# cat Dockerfile


① centos(latest)를 기반으로하는
② 관리자는 hoge oscinfra<khoj@osci.kr>
③ httpd를 설치
④ 80 번 포트를 오픈
⑤ httpd 서버를 FOREGROUND로 시작
```

```
FROM            기반이 되는 Docker 이미지 지정
MAINTAINER      작성자 정보
RUN             명령의 실행
ADD             파일 / 디렉토리 추가
CMD             컨테이너의 실행 명령 1
ENTRYPOINT      컨테이너의 실행 명령 2
WORKDIR         작업 디렉토리 지정
ENV             환경 변수 지정
USER            실행 사용자 지정
EXPOSE 포트 내보내기
VOLUME 볼륨 마운트
```

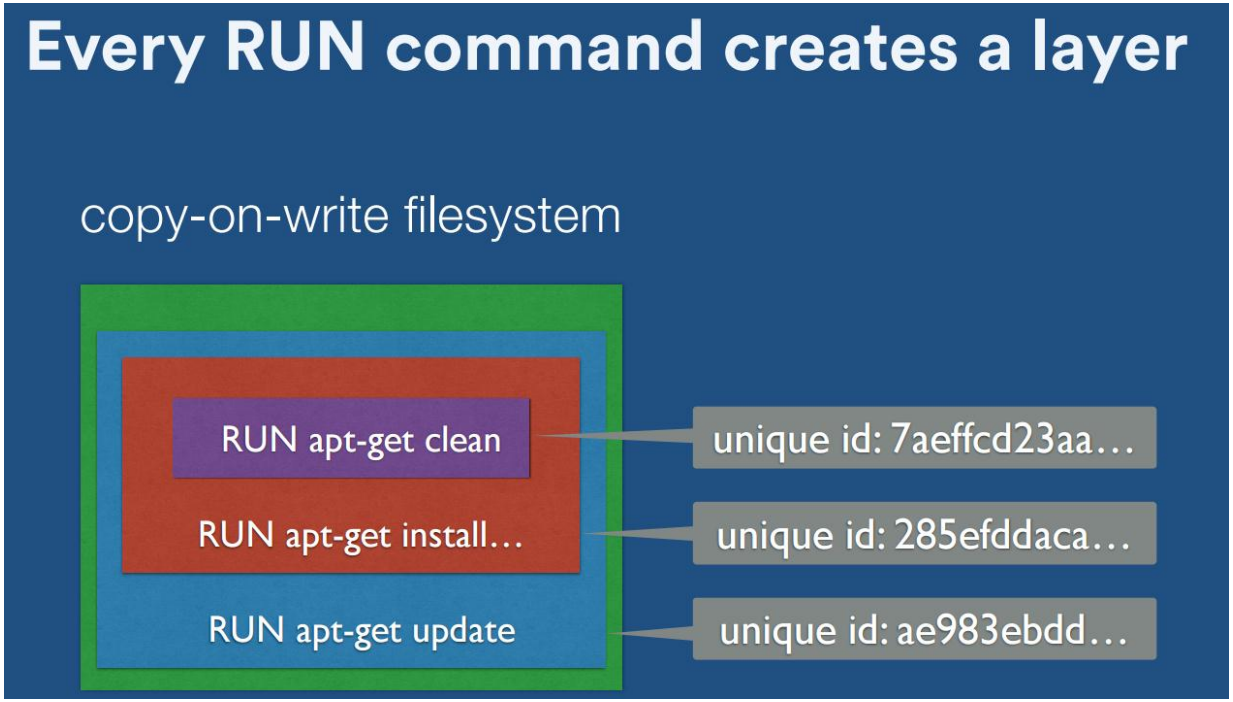
Docker - Dockerfile

- oscinfra/httpd:1.0이라는Docker 이미지를 생성

```
[root@centos7 httpd]# docker build -t oscinfra/httpd:1.0 ./
Sending build context to Docker daemon 2.048 kB
Step 0 : FROM centos:latest
 ---> 0f73ae75014f
Step 1 : MAINTAINER oscinfra<khoj@osci.kr>
 ---> Using cache
 ---> 92af619625fc
Step 2 : RUN yum install -y httpd
 ---> Running in 1387cc55e281
 ---> 3887d832801c
Removing intermediate container 1387cc55e281
Step 3 : EXPOSE 80
 ---> Running in e4ec8d71efb2
 ---> 2d19e2d29feb
Removing intermediate container e4ec8d71efb2
Step 4 : CMD ["httpd", "-D" "FOREGROUND"]
 ---> Running in 20d5eaab4f76
 ---> 041a74b4ddb1
Removing intermediate container 20d5eaab4f76
Successfully built 041a74b4ddb1
```

Docker - Docker 이미지 빌드

```
FROM stackbrew/ubuntu:13.10

RUN apt-get update
RUN apt-get install -y software-…-common python …
RUN add-apt-repository -y ppa:chris-lea/node.js
RUN apt-get update
RUN apt-get install -y nodejs
```



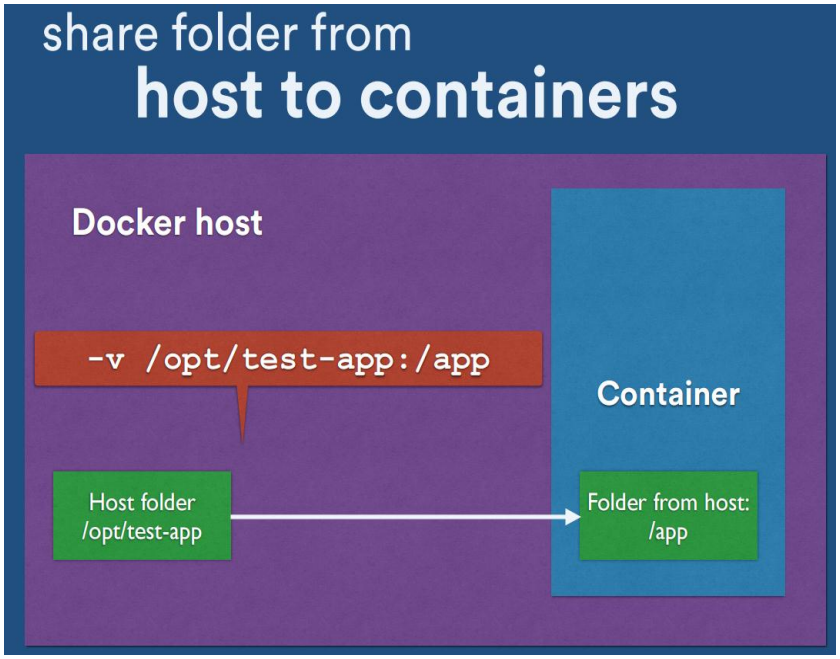
출처 : http://www.slideshare.net/durdn/be-a-better-developer-with-docker

Docker - Data Volume

호스트 볼륨 공유

- 여러 컨테이너간에 영구적인 데이터와 공유 데이터를 처리하는 특별한 볼륨(디렉토리).
- Data Volume 대한 변경은 직접 반영되어 이미지의 변경에 포함되지 않는다.
- Data Volume은 참조 할 컨테이너가 없어져도 살아난다.

```
docker run -it -v /opt/test-app:/app ubuntu /bin/bash
```




```
[root@centos7 test-app]# pwd
/opt/test-app
[root@centos7 test-app]# touch from_host_to_container
[root@centos7 test-app]# ls
from_container_to_host   from_host_to_container
```

```
root@fa6912b4909b:/app# pwd
/app
root@fa6912b4909b:/app# touch from_container_to_host
root@fa6912b4909b:/app# ls
from_container_to_host   from_host_to_container
```

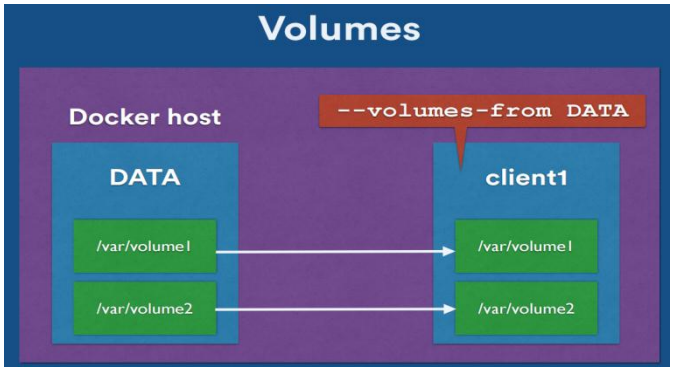
출처 : http://www.slideshare.net/durdn/be-a-better-developer-with-docker

Docker - Data Volume Container

컨테이너 사이의 볼륨 공유

```
FROM        busybox
VOLUME      ["/var/volume1", "/var/volume2"]
CMD         ["/bin/true"]
```

```
FROM ubuntu
…
…
VOLUME ["/var/volume"]
CMD ["/bin/bash"]
```




```
docker run -v /var/volume1 \
           -v /var/volume2 \
           --name DATA busybox true
```

```
docker run -t -i -rm --volumes-from DATA \
 --name client1 ubuntu bash
```

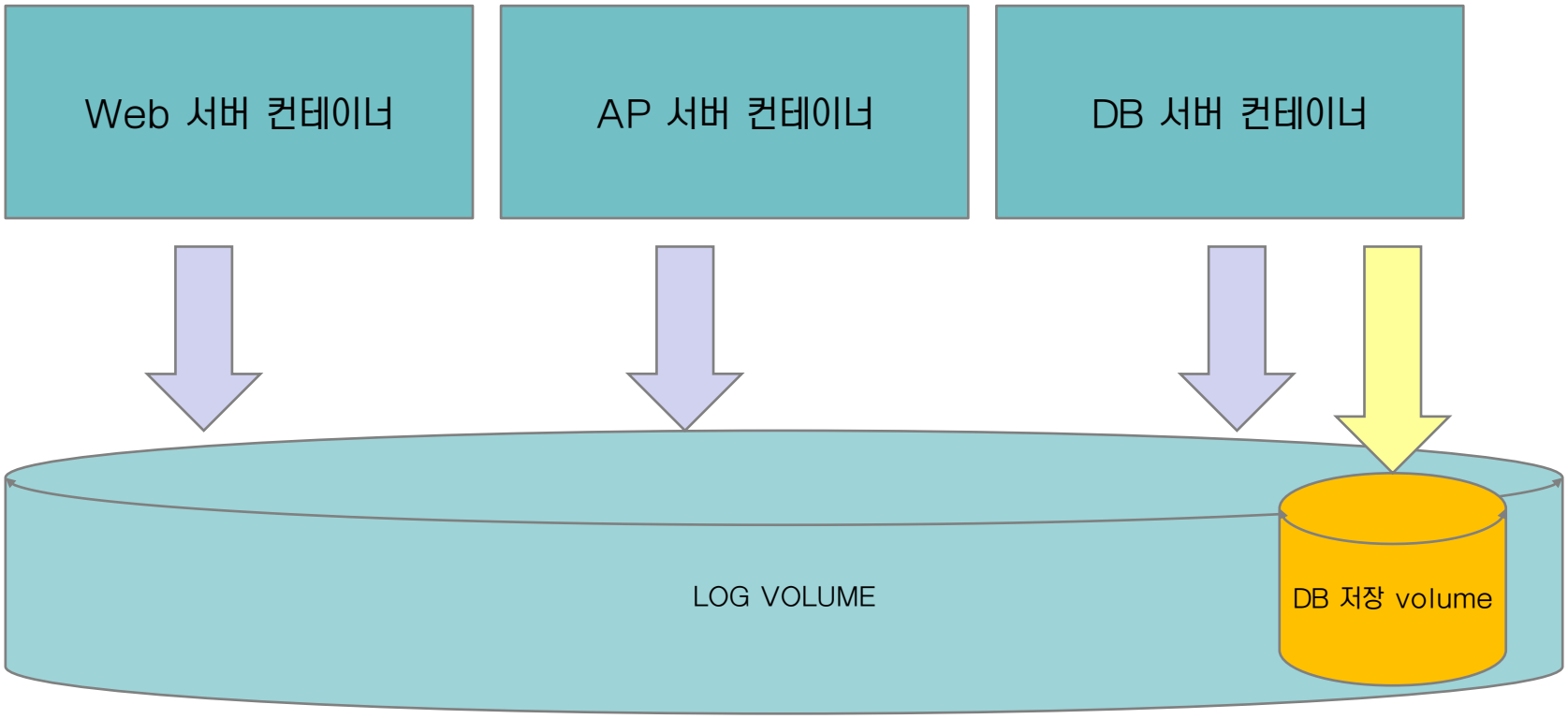
```
[root@centos7 ~]# docker run -it -v /var/volume --name vol1 ubuntu /bin/bash
root@3d58c7f5820e:/# cd /var/volume/
root@3d58c7f5820e:/var/volume# touch from_host_to_client
root@3d58c7f5820e:/var/volume# ls
from_client_to_host  from_host_to_client
```

```
[root@centos7 ~]# docker run -it --volumes-from vol1 ubuntu /bin/bash
root@01dd31951f2d:/# cd /var/volume/
root@01dd31951f2d:/var/volume# touch from_client_to_host
from_client_to_host  from_host_to_client
```

출처 : http://www.slideshare.net/durdn/be-a-better-developer-with-docker

Docker - Data Volume Container

- 사용자 데이터 등 깨지지 않는 정보의 저장 장소로 Data Volume Container를 이용한다.



Docker - 네트워크 포트 매핑

- 컨테이너를 시작할 때 컨테이너 내부에서 사용하는 포트를 임의의 호스트 측 포트 번호 (49152 이상 권장)에 할당
- 호스트 측의 포트 번호는 49000 ~ 49900의 범위에서 자동으로 임의의 번호를 할당하거나 사용자가 임의의 번호를 지정 가능
- 컨테이너에서 열린 포트를 호스트 측 임의의 포트 번호에 매핑

```
[root@centos7 ~]# docker run -d -P oscinfra/sshd2:latest /usr/sbin/sshd -D
a210be9d2fae3b4a9967b504785307e0afc321df8fbc8447a9578ba7d092214c
[root@centos7 ~]# docker run -d -p 22 oscinfra/sshd2:latest /usr/sbin/sshd -D
815cfb3316ed767b83926e1bf0060c6e8387216ec7559de25dc32e82a21f11ae
```

```
[root@centos7 ~]# docker ps -a
815cfb3316ed        oscinfra/sshd2:latest    "/usr/sbin/sshd -D"    2 minutes ago       Up 2 minutes
0.0.0.0:32768->22/tcp    stoic_thompson
54e6440174ad        ubuntu:latest            "/usr/sbin/sshd -D"    6 minutes ago       Created
condescending_davinci
a210be9d2fae        oscinfra/sshd2:latest    "/usr/sbin/sshd -D"    7 minutes ago       Up 7 minutes
determined_wilson
```

- 컨테이너 포트 22이 호스트의 어떤 포트에 매핑되었는지 아는 방법

```
[root@centos7 ~]# docker port 815cfb3316ed 22
0.0.0.0:32768
[root@centos7 ~]# docker port a210be9d2fae 22
Error: No public port '22/tcp' published for a210be9d2fae
```

Docker - 네트워크 포트 매핑

- 컨테이너 포트 22을 유저가 지정하는 호스트 측면 포트 번호 2222에 매핑

```
[root@centos7 ~]# docker run -d -p 2222:22 oscinfra/sshd2:latest /usr/sbin/sshd -D
03ca82b92ac41b991262bc929938c2eb741f7a7e475173c7fd37841f04b6508a
[root@centos7 ~]# docker ps -a
CONTAINER ID        IMAGE                    COMMAND               CREATED             STATUS
PORTS                     NAMES
03ca82b92ac4        oscinfra/sshd2:latest    "/usr/sbin/sshd -D"   6 seconds ago       Up 5 seconds
0.0.0.0:2222->22/tcp      high_jones
815cfb3316ed        oscinfra/sshd2:latest    "/usr/sbin/sshd -D"   12 minutes ago      Up 12 minutes
0.0.0.0:32768->22/tcp     stoic_thompson

[root@centos7 ~]# docker port 03ca82b92ac4 22
0.0.0.0:2222
```

Docker - 컨테이너 링크

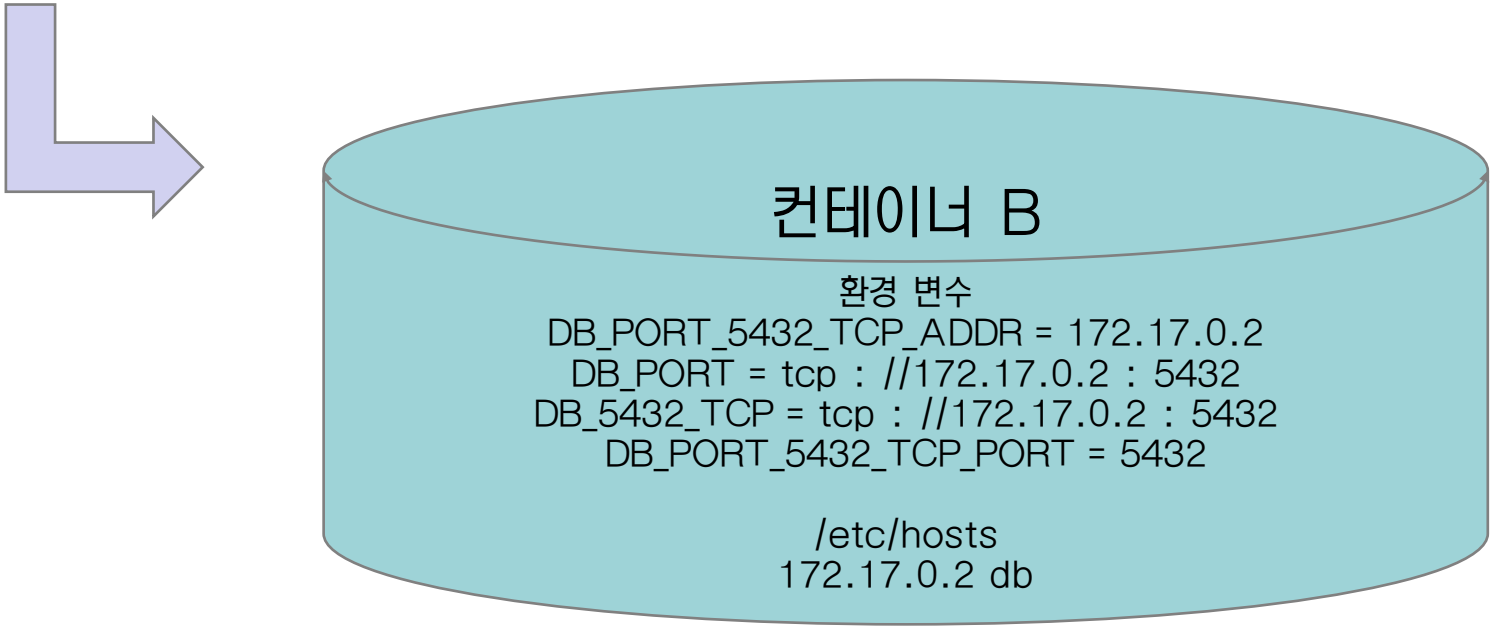
- 여러 컨테이너간에 전용 네트워크를 구축

원본 컨테이너 A:

```
docker run –d --name pg ubuntu:latest postmaster -D /usr/local/pgsql/data
```

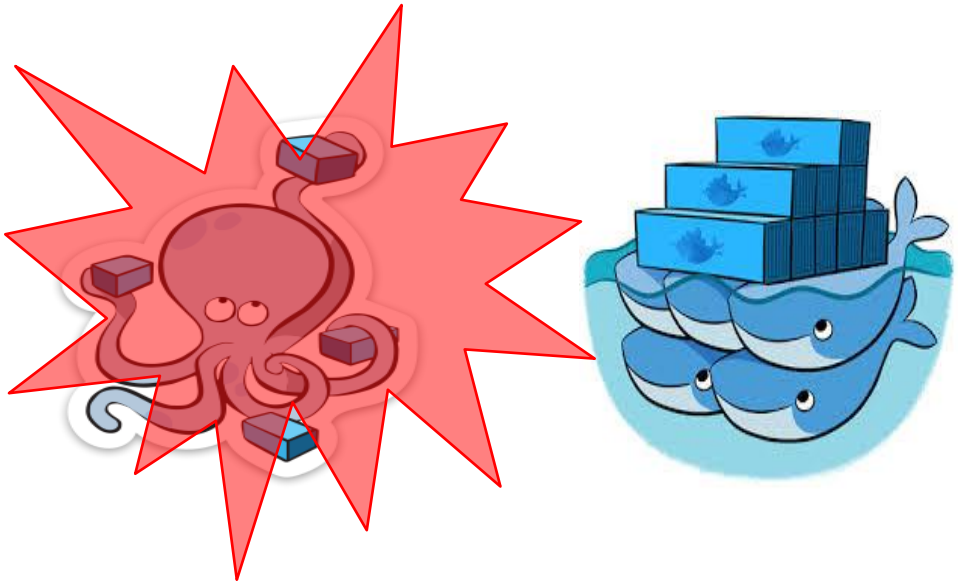
컨테이너 A에 연결하는 컨테이너 B:

```
docker run -d --link pg:db ubuntu:latest /someService
```



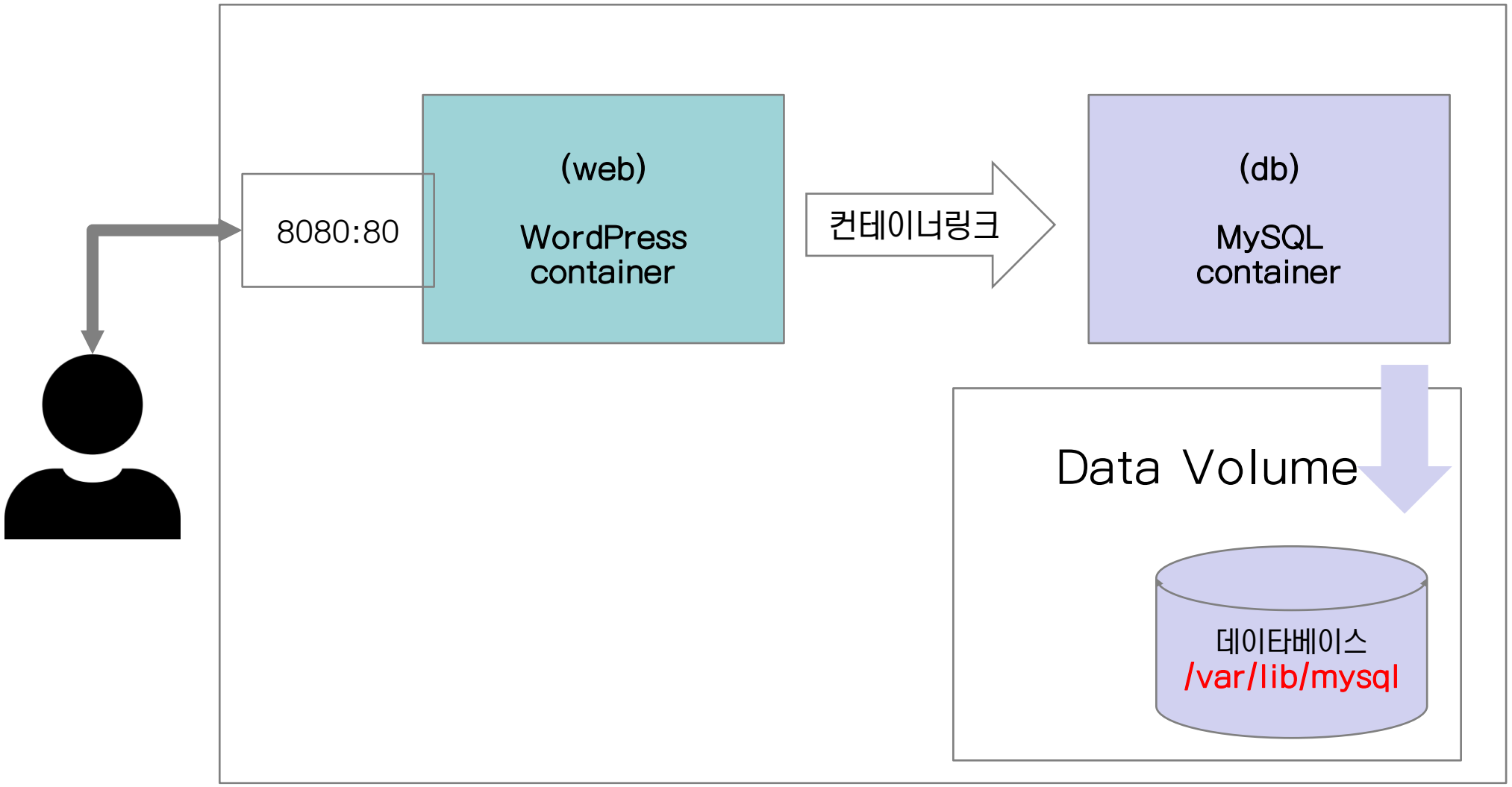
Docker Compose

소개



- 도커 복잡한 응용 프로그램을 정의하고 실행하기위한 도구입니다
- 멀티 컨테이너를 하나의 파일로 운용
- 하나의 command로 모든 docker file들 실행

Docker compose



Docker compose - web - db - storage를 컨테이너로 구축

- Web 서버의 Dockerfile

```
FROM ubuntu:latest
MAINTAINER oscinfra<oscinfra@gmail.com>

VOLUME /var/lib/mysql
CMD /bin/bash
```

- DataVolume container (storage)

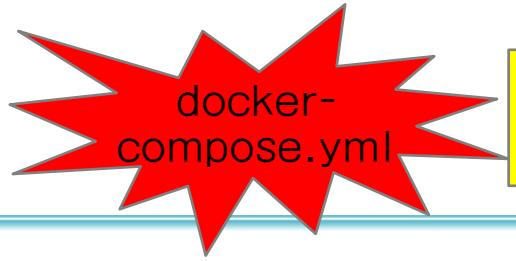
```
docker build -t storage ./
docker run -it --name storage storage
```

- mysql container 작성

```
docker run --name db  --volumes-from storage  -e MYSQL_ROOT_PASSWORD=password -d mysql:5.7
```

- wordpress container 작성

```
docker run --name web2  --link db:mysql -p 8080:80 -d wordpress:latest
```



시스템 구성 자동화
여러 컨테이너 갖춰진 시스템 구성 설정 파일 (YAML)으로 정의 할 수 한꺼번에 구축하고 관리 할 수있는 도구

Docker compose

Compose install

```
[root@docker188 ~]# curl -L
https://github.com/docker/compose/releases/download/VERSION_NUM/docker-compose-‘uname -s‘-
‘uname -m‘ > /usr/local/bin/docker-compose

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   403    0   403    0     0    343      0 --:--:--  0:00:01 --:--:--   343
100 5140k  100 5140k    0     0   1031k      0  0:00:04  0:00:04 --:--:-- 1871k

 [root@docker188 ~]# chmod +x /usr/local/bin/docker-compose
```

Docker Compose를 사용

- Data Volume 컨테이너 이미지 (storage)를 구축

```
docker-compose build storage
```

- WordPress 시스템을 구성하는 컨테이너를 한꺼번에 만들고 시작

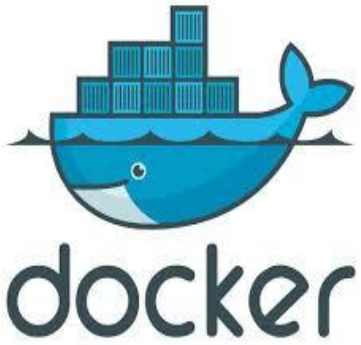
```
docker-compose up -d
```

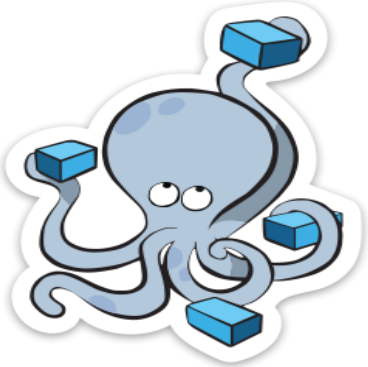
- 기타 기본적인 이용 방법

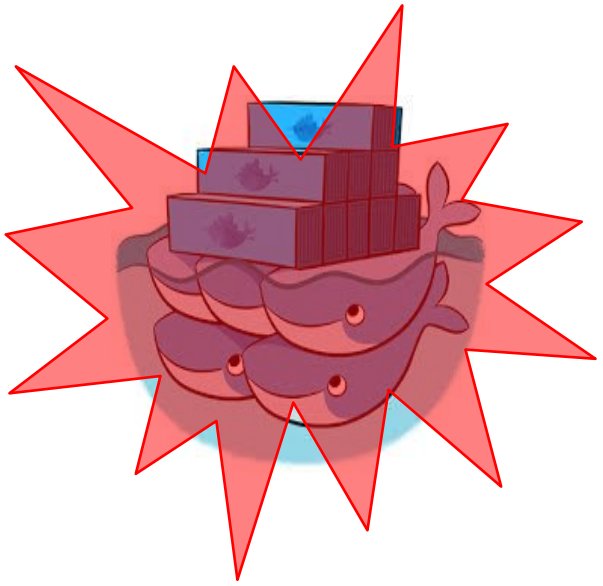
```
docker-compose stop                    시스템을 구성하는 컨테이너를 단번에 시작 (다시 시작)
docker-compose start  (restart)

docker-compose logs                    시스템을 구성하는 컨테이너의 로그를 보기
docker-compose rm                      시스템을 구성하는 컨테이너를 단번에 제거
```

Docker Swarm

소개





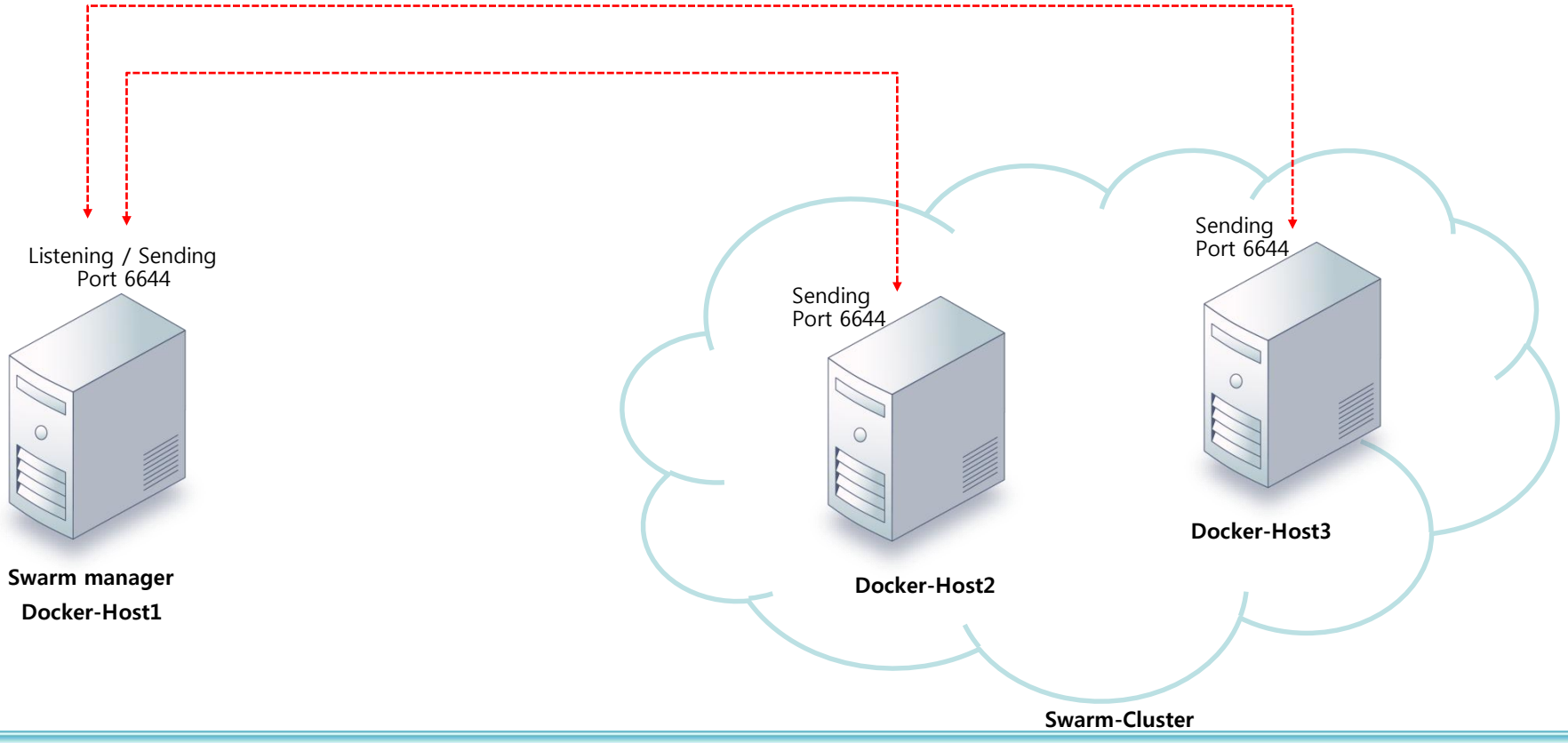


- Docker Native Clustering System
- 클러스터 / 컨테이너 / 이미지를 컨트롤
- 클러스터 그룹에서 비교적 쉬운 WorkLoad 제어
- Docker API 지원
- Swarm은 매니저 Docker Node는 클러스터의 노드
- 특별히 필요로 하는 인프라 조건은 없음
- 손쉽게 확장 된!
- 시스템의 가용성
- Docker를 여러 호스트에 분산

Docker Swarm

Docker Swarm Architecture

- Swarm 중앙 집중식 관리
- 각 노드의 커맨드는 클러스터를 커져 Swarm을 거쳐 최종 노드에서 수행
- 클러스터의 전략과 필터를 통하여 컨테이너 수행 제어
- Docker Swarm은 여러 호스트 Docker를 함께 클러스터링하여 마치 하나의 Docker 엔진처럼 취급



Docker Swarm

- Data Volume 컨테이너 이미지 (storage)를 구축

```
docker run --rm swarm create
…
e1fc62b24f818a2af00fb4ddb8899965
```

* swarm create하면 token이라는 문자열이 표시되므로 이를 사용하여 각 node를 구성합니다
  호스트 1에서 Swarm manager 컨테이너를 구축

- 호스트 1에서 Swarm manager 컨테이너를 구축

```
docker run -p 2377:2375 -d swarm manage token://e1fc62b24f818a2af00fb4ddb8899965
```

- 각 호스트 각각 Swarm node 컨테이너를 구축

```
docker run -d swarm join --addr = 노드 IP 주소 : 2375 token : // e1fc62b24f818a2af00fb4ddb8899965
```

- Docker 명령의 대상에 manager의 IP 주소와 포트를 지정하면 Docker Swarm 클러스터에 액세스 할 수있다! !

```
docker  -H tcp://192.168.1.11:2377  info
Containers: 0
Nodes: 3
 agent-2: 192.168.1.13:2375
    └Containers: 0
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 514.5 MiB
```

```
agent-1: 192.168.1.12:2375
    └Containers: 0
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 514.5 MiB
 agent-0: 192.168.1.11:2375
    └Containers: 0
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 514.5 MiB
```

Docker Swarm

- 여러 Docker 명령도 사용가능

```
$  docker -H tcp://192.168.1.11:2377 ps
$  docker -H tcp://192.168.1.11:2377 images
$  docker -H tcp://192.168.1.11:2377 run
```

- 환경 변수를 설정하면 "- H tcp : // IP 주소 : 포트"가 불필요

```
$  export DOCKER_HOST=tcp://192.168.1.11:2377
$  docker  ps -a
$  docker  run ...
```

Docker Swarm에 컨테이너 생성

- Docker Swarm 클러스터에 nginx 컨테이너 만들기

```
docker run -d -p 80:80 --name front1 nginx
$ docker ps

CONTAINER ID    IMAGE           COMMAND                  CREATED          STATUS
PORTS                                          NAMES
a4b0755a685   nginx:latest    "nginx -g 'daemon of   32 seconds ago  Up 27 seconds  192.168.1.12:80-
>80/tcp      front1
```

- Docker Swarm 클러스터에 추가 nginx 컨테이너 만들기

```
docker run -d -p 80:80 --name front2 nginx
$ docker ps

CONTAINER ID    IMAGE           COMMAND                  CREATED          STATUS
PORTS                                          NAMES
a4b0755a685   nginx:latest    "nginx -g 'daemon of   32 seconds ago  Up 27 seconds  192.168.1.12:80-
>80/tcp     front1
fa586fafd753    nginx:latest    "nginx -g 'daemon of   3 minutes ago    Up 3 minutes
192.168.1.11:80->80/tcp     front2
```

Docker Swarm에 컨테이너 선호도

- 컨테이너 선호도

```
$ docker run -d -p 3306:3306 --name mydb  mysql
$ docker run -d -p 80:80 -e affinity:container==mydb --name web1 nginx
$ docker run -d -p 80:80 -e affinity:container!=mydb  --name web2 nginx
$ docker ps

CONTAINER ID     IMAGE           COMMAND     CREATED             STATUS              PORTS
NAMES
87c4376856a     mysql:latest    "mysql"           1 minutes ago      Up 57 seconds  192.168.1.12:3306-
>3306/tcp     mydb
a4b0755a685    nginx:latest     "nginx "          32 seconds ago    Up 27 seconds  192.168.1.12:80-
>80/tcp              web1
87c4376856a    nginx:latest     "nginx"          14 seconds ago    Up 10 seconds  192.168.1.13:80-
>80/tcp              web2
```

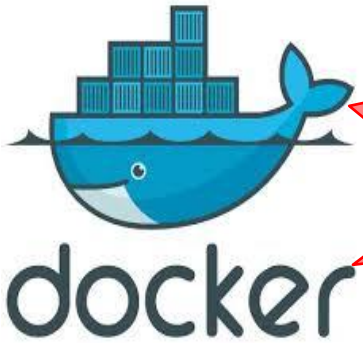
Docker Swarm

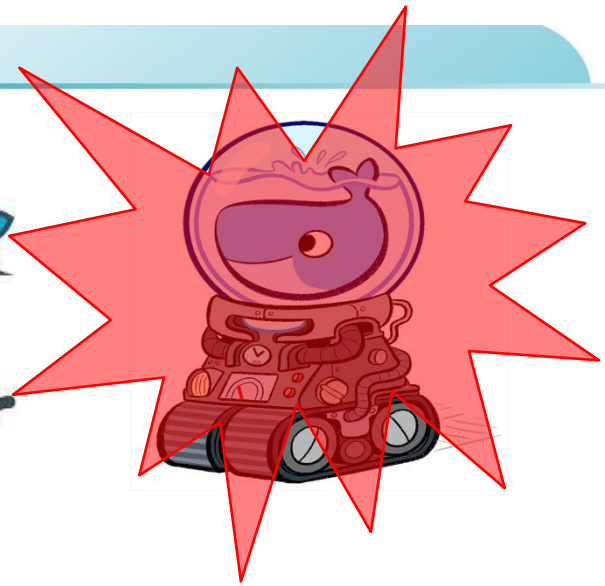
- Portfilter

```
docker run -d -p 80:80 --name front1 nginx
$ docker run -d -p 80:80 --name front2 nginx
$ docker run -d -p 80:80 --name front3 nginx
$ docker ps

CONTAINER ID    IMAGE           COMMAND    CREATED              STATUS              PORTS
NAMES
a4b0755a685   nginx:latest    "nginx -g 'daemon of   32 seconds ago  Up 27 seconds  192.168.1.13:80-
>80/tcp    front1
fa586fafd753    nginx:latest    "nginx -g 'daemon of   3 minutes ago    Up 3 minutes
192.168.1.12:80->80/tcp    front2
963841b1388   nginx:latest    "nginx -g 'daemon of   3 minutes ago    Up 3 minutes
192.168.1.11:80->80/tcp    front1


$ docker run -d -p 80:80 --name front4 nginx
2015/7/15 00:33:20 Error response from daemon: no resources available to schedule container
```

Docker machine

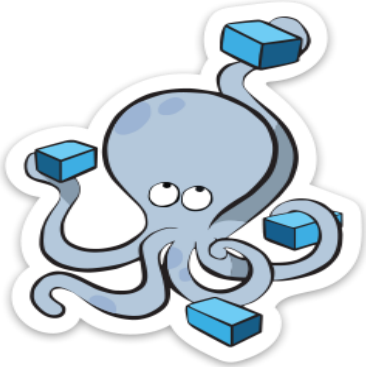
소개

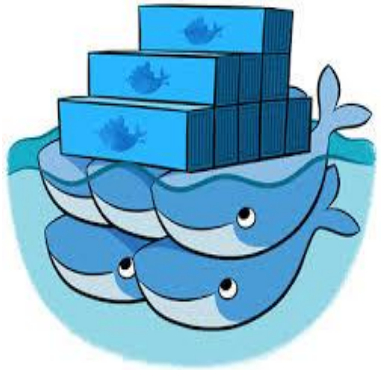




- Docker가 움직이는 서버가 많이 늘었다!
- Docker Swarm도 증가 할 것 같다! !

- 호스트 서버의 설정 이라든지 Docker 설치 라든지
- Swarm 구축 이라든지 수작업으로하는 것은 불가
- Docker Machine을 사용하자





- Docker Machine은
- 로컬 또는 클라우드에 Docker 엔진이 움직이는 호스트(vm)를 구축하고 관리 할 수있는 도구!
- Docker Swarm 클러스터도 구축 할 수있다.

Docker machine

- Install

```
curl -L https://github.com/docker/machine/releases/download/v0.3.0/docker-machine_darwin-amd64 > /usr/local/bin/docker-machine

$ chmod +x /usr/local/bin/docker-machine
$ docker-machine -v
machine version 0.3.0
```

- 로컬에 Docker 호스트를 만들
- "host1"라는 Docker 호스트 (Virtualbox에서 vm)을 구축

```
docker-machine create --driver virtualbox  host1
```

- 구축 한 Docker 호스트보기

```
docker-machine ls

NAME    ACTIVE    DRIVER          STATE        URL                                SWARM
host1                 virtualbox    Running    tcp://192.168.99.100:2376
```

* 실제로 VirtualBox에 boot2docker의 vm 이미지를 다운로드하고 시작합니다.
  따라서 로컬 PC에 Virtualbox 4.3.28가 설치되어있는 것이 전제가됩니다.

Docker machine

- Host 활성화

```
eval "$(docker-machine env host1)"
docker-machine env  host1

export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.100.99:2376"
export DOCKER_CERT_PATH="/Users/<username>/.docker/machine/machines/host1"
export DOCKER_MACHINE_NAME="host1"
```

- Docker 호스트 목록을 보기

```
docker-machine ls

NAME    ACTIVE    DRIVER         STATE        URL                                SWARM
host1     *             virtualbox     Running    tcp://192.168.99.100:2376
```

- 활성 Docker 호스트에 컨테이너를 생성

```
$ docker run -d -p 80:80 --name web nginx
$ docker ps
$ docker stop ...
$ docker rm ...
```

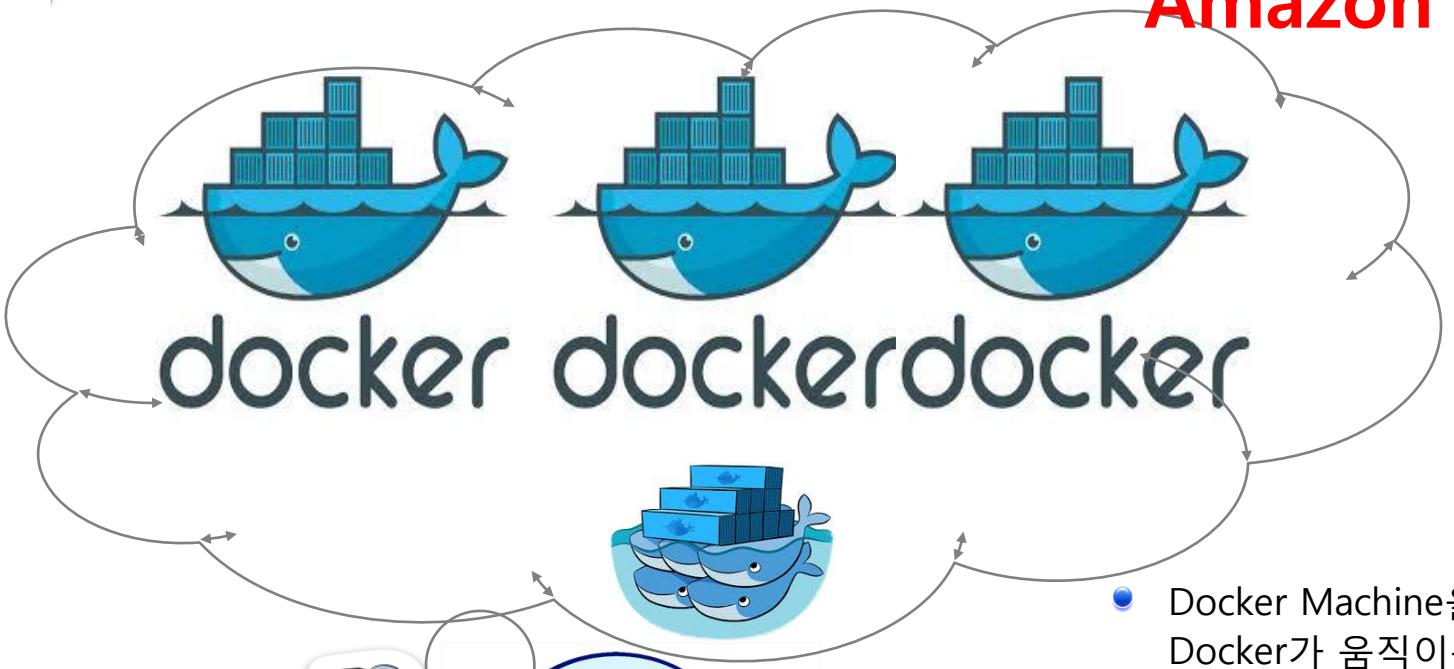
Docker machine

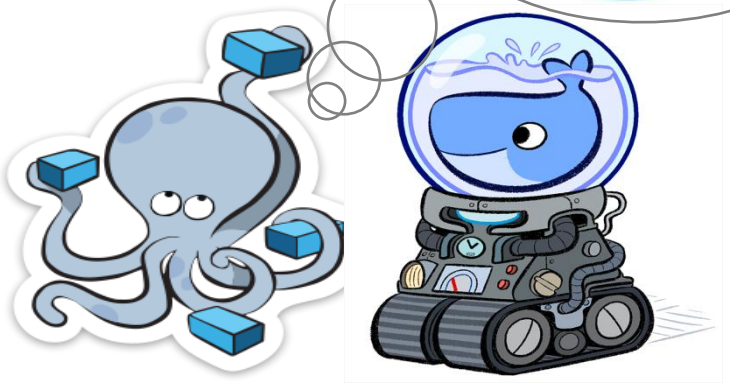
- 활성화 된 Docker 호스트에 대해서는 일반적으로 Docker 명령을 사용할 수 있음

```
$ docker-machine stop [호스트 이름]
$ docker-machine start [호스트 이름]
$ docker-machine rm [호스트 이름]
```

소개

Amazon Web Service





- Docker Machine을 사용하여 Amazon EC2에 Docker가 움직이는 호스트를 여러 대 구축
- 동시에 이러한 호스트를 클러스터링하고 Docker Swarm 환경을 구축
- Docker Swarm 환경에 Docker Compose에서 컨테이너를 생성하고 연계

Docker Machine + Docker Swarm

- Docker Machine에서 Docker Swarm 클러스터링을 create

```
docker-machine create --driver amazonec2  ₩
  --amazonec2-access-key  [your access-key]    ₩
  --amazonec2-secret-key   [your secet-key]  ₩
  --amazonec2-ami ami-f4b06cf4   ₩
  --amazonec2-region ap-northeast-1    ₩
  --amazonec2-vpc-id  [your vcp-id]   ₩
  swarm-create
```

- Docker Machine에서 Swarm manager되는 호스트 1 'master'를 구축

```
eval $(docker-machine env swarm-create)
$ docker run swarm create
…
  7d1ca38131c37fe03af760221de31a01

$ docker-machine create --driver amazonec2    ₩
  --amazonec2-access-key  [your access-key]      ₩
  --amazonec2-secret-key   [your secet-key]    ₩
  --amazonec2-ami ami-f4b06cf4     ₩
  --amazonec2-region ap-northeast-1       ₩
  --amazonec2-vpc-id  [your vcp-id]  ₩
  --swarm  ₩
  --swarm-master  ₩
  --swarm-discovery token://7d1ca38131c37fe03af760221de31a01  ₩
  master
```

Docker Machine + Docker Swarm

- Docker Machine에서 Swarm node되는 호스트 2 "node1"을 구축

```
$ docker-machine create --driver amazonec2    ₩
  --amazonec2-access-key  [your access-key]     ₩
  --amazonec2-secret-key   [your secet-key]    ₩
  --amazonec2-ami ami-f4b06cf4     ₩
  --amazonec2-region ap-northeast-1       ₩
  --amazonec2-vpc-id  [your vcp-id]  ₩
  --swarm  ₩
  --swarm-discovery token://7d1ca38131c37fe03af760221de31a01  ₩
  node1
```

- Docker Machine에서 Swarm node되는 호스트 3 "node2"을 구축

```
$ docker-machine create --driver amazonec2    ₩
  --amazonec2-access-key  [your access-key]     ₩
  --amazonec2-secret-key   [your secet-key]    ₩
  --amazonec2-ami ami-f4b06cf4     ₩
  --amazonec2-region ap-northeast-1       ₩
  --amazonec2-vpc-id  [your vcp-id]  ₩
  --swarm  ₩
  --swarm-discovery token://7d1ca38131c37fe03af760221de31a01  ₩
  node2
```

Docker Machine + Docker Swarm

- Swarm manager 호스트 1 "master"연결 확인

```
$ eval $(docker-machine env  --swarm master)
$ docker info
Containers: 4
Images: 3
Role: primary
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 3
  dm-master: 52.69.171.103:2376
    └Containers: 2
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 1.018 GiB
    └Labels: executiondriver=native-0.2, kernelversion=3.13.0-53-generic, operatingsystem=Ubuntu 14.04.2 LTS,
provider=amazonec2, storagedriver=aufs
  dm-node1: 52.69.146.1:2376
    └Containers: 1
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 1.018 GiB
    └Labels: executiondriver=native-0.2, kernelversion=3.13.0-53-generic, operatingsystem=Ubuntu 14.04.2 LTS,
provider=amazonec2, storagedriver=aufs
  dm-node2: 52.69.190.50:2376
    └Containers: 1
    └Reserved CPUs: 0 / 1
    └Reserved Memory: 0 B / 1.018 GiB
    └Labels: executiondriver=native-0.2, kernelversion=3.13.0-53-generic, operatingsystem=Ubuntu 14.04.2 LTS,
provider=amazonec2, storagedriver=aufs
CPUs: 3
Total Memory: 3.053 GiB
```

Docker Machine + Docker Swarm

- Docker Machine에서 AWS에 호스트를 구축하는 명령 예

```
$ docker-machine create --driver amazonec2  ₩
  --amazonec2-access-key  [your access-key]    ₩
  --amazonec2-secret-key   [your secet-key]  ₩
  --amazonec2-ami        [ami-id of the instance to use]   ₩
  --amazonec2-region   [region to use]    ₩
  --amazonec2-vpc-id   [your vcp-id]   ₩
  [host name]
```

- Asia Pacific (Tokyo) Region의 경우 일부 매개 변수는 다음과 같습니다

```
 --amazonec2-ami  => ami-f4b06cf4
 --amazonec2-region => ap-northeast-1
```

- Docker Machine에서 Swarm manager를 구축하는 명령 예

```
docker-machine create --driver [Driver]  ₩
  --swarm  ₩
  --swarm-master  ₩
  --swarm-discovery token://7d1ca38131c37fe03af760221de31a01  ₩
  [host name]
```

Docker Machine + Docker Swarm

- Docker Machine에서 Swarm node를 구축하는 명령 예

```
$ docker-machine create --driver [Driver]  ₩
  --swarm  ₩
  --swarm-discovery token://7d1ca38131c37fe03af760221de31a01  ₩
  [host name]
```

Docker compose + Docker Swarm

- nginx 컨테이너 정의

```
docker-compose.yml
nginx:
    image: nginx:latest
    ports:
      - "80:80"
```

- nginx 컨테이너를 생성

```
$ docker-compose up -d
Creating nginx_nginx_1...

$ docker ps

CONTAINER ID    IMAGE           COMMAND          CREATED          STATUS              PORTS
NAMES
2a6ab7f5b98a    nginx:latest    "nginx -g 'daeon   7seconds ago    Up 7 seconds    52.69.146.1:80->80/tcp
node1/nginx_nginx_1
```

Docker compose + Docker Swarm

- Scale-out

```
$ docker-compose scale nginx=2
Service nginx specifies a port on the host. If multiple containers for this service are created on a
single host, the port will clash.
Creating nginx_nginx_2...
Starting nginx_nginx_2...
$ docker ps

CONTAINER ID    IMAGE           COMMAND    CREATED          STATUS                          PORTS
NAMES
312649a3b648  nginx:latest   "nginx -g 'da   5 seconds ago  Up 4 seconds            52.69.190.50:80-
>80/tcp node2/nginx_nginx_2
2a6ab7f5b98a   nginx:latest    "nginx -g 'da   a minute ago   Up About a minute  52.69.146.1:80-
>80/tcp   node1/nginx_nginx_1
```

- Scale-out again !!

```
$ docker-compose scale nginx=3
Service nginx specifies a port on the host. If multiple containers for this service are created on a single host, the port will clash.
Creating nginx_nginx_3...
Starting nginx_nginx_3...
docker ps로 확인
$ docker ps

CONTAINER ID    IMAGE          COMMAND    CREATED          STATUS                    PORTS                                NAMES
82caa4b2b626  nginx:latest   "nginx -g 'da   4 seconds ago  Up 2 seconds        52.69.171.103:80->80/tcp master/nginx_nginx_3
312649a3b648  nginx:latest   "nginx -g 'da   5 seconds ago  Up 4 seconds        52.69.190.50:80->80/tcp   node2/nginx_nginx_2
2a6ab7f5b98a   nginx:latest    "nginx -g 'da   a minute ago   Up About a minute  52.69.146.1:80->80/tcp   node1/nginx_nginx_1
```

Docker compose + Docker Swarm

- Scale-out again !! ➔ error

```
$ docker-compose scale nginx=4
Service nginx specifies a port on the host. If multiple containers for this service are created on a
single host, the port will clash.
Creating nginx_nginx_4...
unable to find a node with port 80 available
docker ps로 확인
$ docker ps

CONTAINER ID    IMAGE           COMMAND    CREATED          STATUS                    PORTS
NAMES
82caa4b2b626  nginx:latest   "nginx -g 'da   4 seconds ago  Up 2 seconds          52.69.171.103:80-
>80/tcp master/nginx_nginx_3
312649a3b648  nginx:latest   "nginx -g 'da   5 seconds ago  Up 4 seconds          52.69.190.50:80-
>80/tcp   node2/nginx_nginx_2
2a6ab7f5b98a   nginx:latest    "nginx -g 'da   a minute ago   Up About a minute  52.69.146.1:80-
>80/tcp   node1/nginx_nginx_1
```

Docker compose + Docker Swarm

- Scale-in

```
docker-compose scale nginx=1
Service nginx specifies a port on the host. If multiple containers for this service are created on a
single host, the port will clash.
Stopping nginx_nginx_3...
Stopping nginx_nginx_2...
Removing nginx_nginx_3...
Removing nginx_nginx_2...

$ docker ps

CONTAINER ID    IMAGE           COMMAND    CREATED          STATUS                      PORTS
NAMES
2a6ab7f5b98a    nginx:latest    "nginx -g 'da    a minute ago    Up About a minute    52.69.146.1:80-
>80/tcp    node1/nginx_nginx_1
```

Volume 에 link 시도

- wordpress 시스템 docker-compose.yml

```
storage:
      build: ./storage
db:
      image: mysql:5.7
      volumes_from:
            - storage
      ports:
            - "3306:3306"
      environment:
            MYSQL_ROOT_PASSWORD: password
web:
      image: wordpress:latest
      links:
            - db:mysql
      ports:
            - "80:80"
```

- Data Volume 컨테이너 이미지 (storage)를 구축

```
$ docker-compose build storage
```

- WordPress 시스템을 구성하는 컨테이너를 한꺼번에 만들고 시작

```
docker-compose up -d
```

Volume 에 link 시도

- status

```
$ docker ps

CONTAINER ID     IMAGE                    COMMAND        CREATED             STATUS
PORTS                                     NAMES
9b214205ddb0   wordpress:latest    "/entrypoint.sh    6 seconds ago    Up 5 seconds   52.69.146.1:80-
>80/tcp              node1/wp1_web_1
bdffe99d4b6e     mysql:5.7                "/entrypoint.sh    6 seconds ago    Up 5 seconds
52.69.146.1:3306->3306/tcp   node1/wp1_db_1
```

```
storage:
    build: ./storage
db:
    image: mysql:5.7
    volumes_from:
        - storage
    ports:
        - "3306:3306"
    environment:
        MYSQL_ROOT_PASSWORD: password
web:
    image: wordpress:latest
    ports:
        - "80:80"
    environment:
        - "affinity:container!=wp1_db*"
    environment:
        WORDPRESS_DB_HOST: 192.168.99.102:3306
        WORDPRESS_DB_PASSWORD: password
```

- docker swarm 주의사항

Volume을 이용하여 컨테이너가 사용하는 디스크를 마운트하거나Link를 이용하여 컨테이너 간의 연계를 시키려고하면 이 컨테이너는 동일한 노드에 생성되어 버린다! !

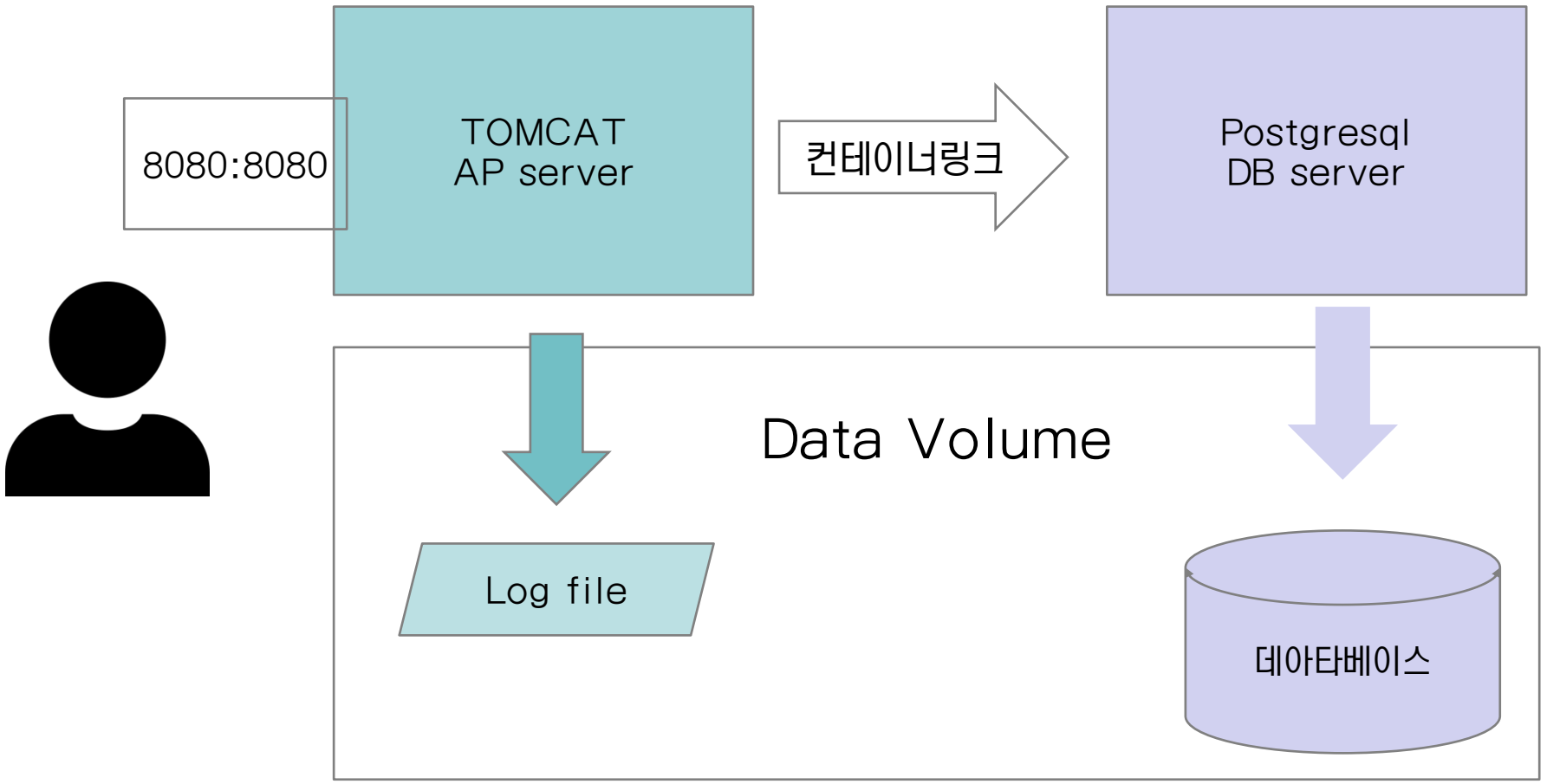
db 및 web을 다른 노드에 배분

Volume 에 link 시도

- WordPress 시스템을 구성하는 컨테이너를 한꺼번에 만들고 시작

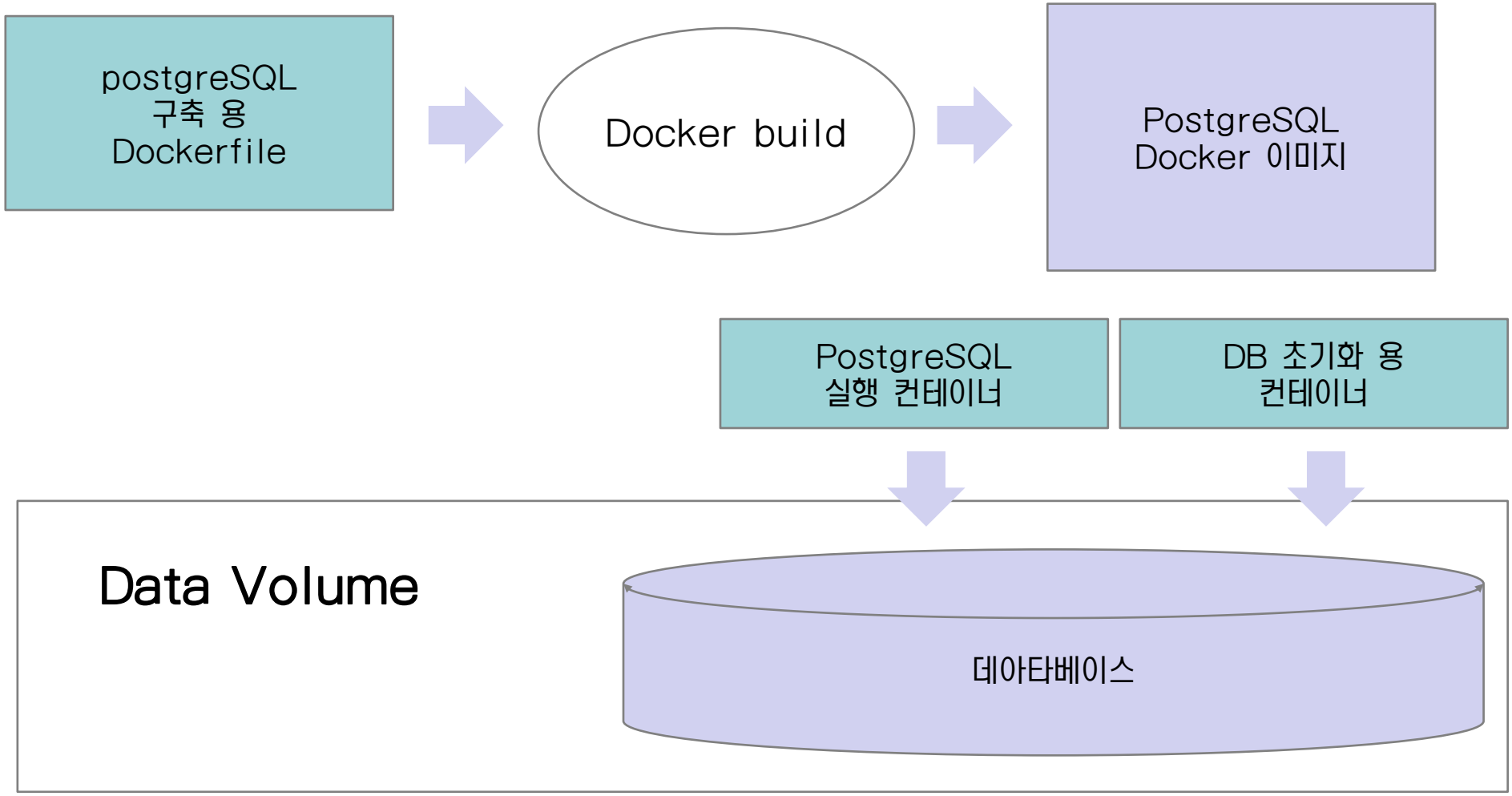
```
$ docker-compose up -d
$ docker ps

CONTAINER ID    IMAGE                    COMMAND        CREATED             STATUS
PORTS                                    NAMES
016e3a049ab3  mysql:5.7                "/entrypoint.sh   6 seconds ago    Up 5 seconds
52.69.146.1:3306->3306/tcp  node1/wp1_db_1
8e739ff21dbd    wordpress:latest   "/entrypoint.sh   6 seconds ago   Up 5 seconds   52.69.190.50:80-
>80/tcp          node2/wp1_web_1
```

AP +DB 프로그램 실행 환경 구축



AP +DB 프로그램 실행 환경 구축

- DB 서버 구축



AP +DB 프로그램 실행 환경 구축

- AP 서버의 Dockerfile

```
FROM centos:latest
MAINTAINER oscinfra
# Tomcat 설치
RUN yum -y install tomcat
RUN echo 'export CATALINA_BASE="/usr/share/tomcat"' >> /root/.bashrc
# Tomcat 시작 스크립트를 복사
ADD tomcat_start.sh /usr/bin/
RUN chmod a+x /usr/bin/tomcat_start.sh
# 프로그램을 배포
#ADD bookmgr.war /usr/share/tomcat/webapps/
# PORT open
EXPOSE 8080
# 컨테이너 실행 명령
ENTRYPOINT ["/usr/bin/tomcat_start.sh"]
```

```
[root@centos7 was]# cat /usr/bin/tomcat_start.sh
#! /bin/bash
export CATALINA_BASE="/usr/share/tomcat"
/usr/sbin/tomcat start
trap '/usr/sbin/tomcat stop; sleep 2; exit 0' TERM
while :
do
sleep 1
done
```

AP +DB 프로그램 실행 환경 구축

- AP image 생성

```
mkdir -p /docker_mnt/pgsql/data
```

- Postgresql 초기 설정 컨테이너 시작

```
docker run -it --rm -v /docker_mnt/pgsql/data:/var/lib/pgsql/data oscinfra/postgres:1.0 /bin/bash
```

- Postgresql의 초기 설치

```
컨테이너의 bash에서 작업
# chown -R postgres:postgres /var/lib/pgsql/data
# su - postgres
$ initdb –encoding=UTF8 --no-locale
$ pg_ctl -D /var/lib/pgsql/data start
$ psql ・・ 필요에 따라 DB 오브젝트를 구축 해당 컨테이너를 종료
```

* PostgreSQL 초기 설정은 Data Volume에 저장되는 데이터베이스 파일을 만들 것이다. 따라서 본 작업은 처음 한 번만 하면된다.

- DB 서버 컨테이너 생성하고 PostgreSQL을 시작

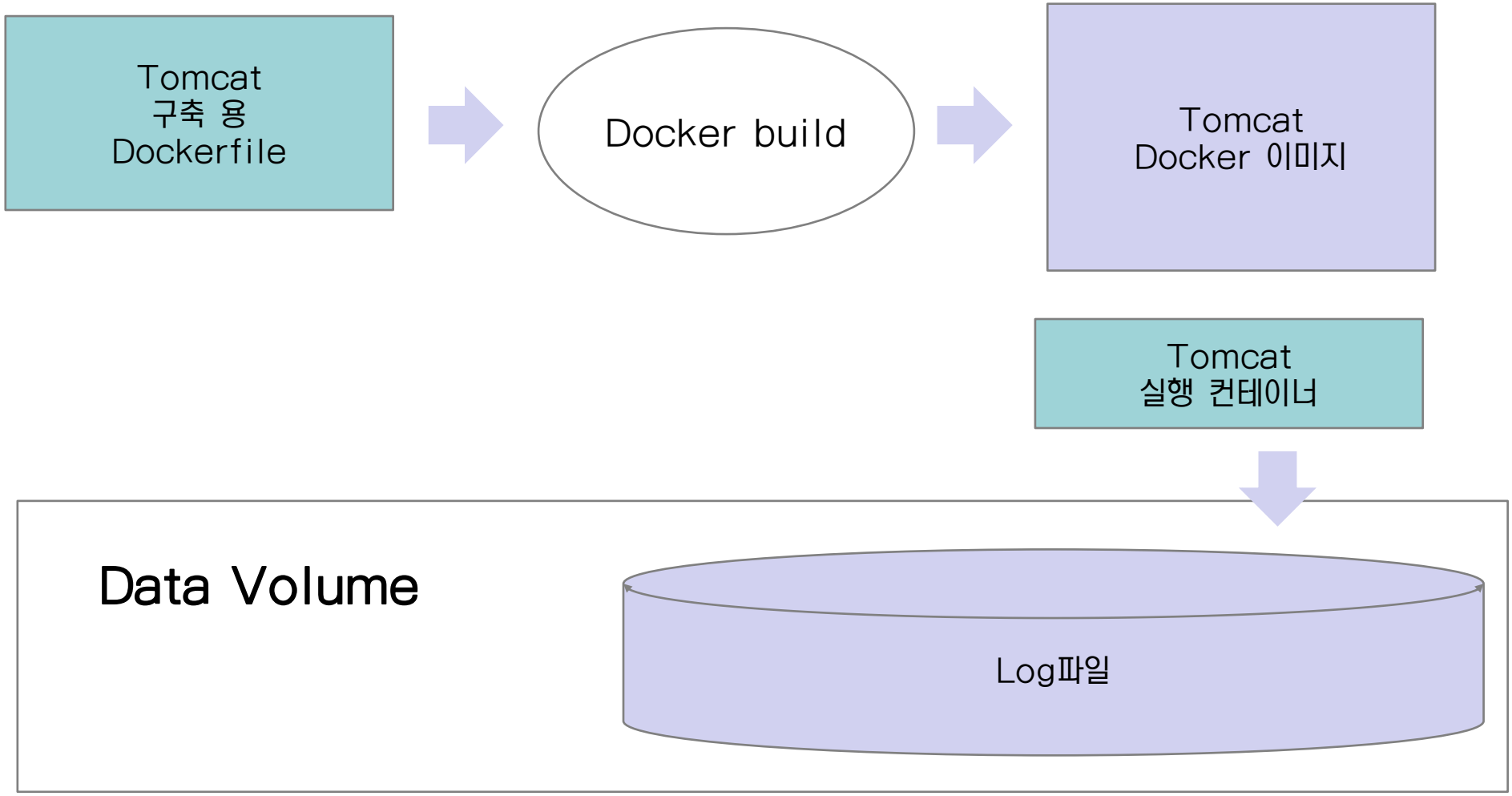
```
sudo docker run -d -v /docker_mnt/pgsql/data:/var/lib/pgsql/data -p 5432:5432 --name db01
oscinfra/postgres:1.0
```

AP +DB 프로그램 실행 환경 구축

```
[root@c79217eded71 /]# chown -R postgres:postgres /var/lib/pgsql/data
[root@c79217eded71 /]# su - postgres
-bash-4.2$ initdb –encoding=UTF8 --no-locale
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

creating directory –encoding=UTF8 ... Ok
Success. You can now start the database server using:

    postgres -D –encoding=UTF8
or
    pg_ctl -D –encoding=UTF8 -l logfile start

-bash-4.2$ pg_ctl -D /var/lib/pgsql/data start
server starting
-bash-4.2$ postgres cannot access the server configuration file "/var/lib/pgsql/data/postgresql.conf":
No such file or directory

[root@centos7 data]# docker run -d -v /docker_mnt/pgsql/data:/var/lib/pgsql/data -p 5432:5432 --name
db01 oscinfra/postgres:1.0
c4b0487a6e1cb070c8c8b3e1b6c6590419cf1fa61dc43a50051dc36757b8686c
```

AP +DB 프로그램 실행 환경 구축

- AP 서버 구축



AP +DB 프로그램 실행 환경 구축

- AP서버의 Dockerfile

```
FROM centos : latest
MAINTAINER hoge
# Tomcat 설치
RUN yum -y install tomcat
RUN echo 'export CATALINA_BASE = "/usr/share/tomcat"'>> /root/.bashrc
# Tomcat 시작 스크립트를 복사
ADD tomcat_start.sh /usr/bin/
RUN chmod a + x /usr/bin/tomcat_start.sh
# 도서 관리 응용 프로그램을 배포
ADD bookmgr.war /usr/share/tomcat/webapps/
# PORT open
EXPOSE 8080
# 컨테이너 실행 명령
ENTRYPOINT [ "/usr/bin/tomcat_start.sh"]
```

- tomcat_start.sh

```
#! /bin/bash
export CATALINA_BASE="/usr/share/tomcat"
/usr/sbin/tomcat start
trap '/usr/sbin/tomcat stop; sleep 2; exit 0' TERM
while :
do
sleep 1
done
```

AP +DB 프로그램 실행 환경 구축

- Data Volume을 저장할 디렉터리

```
docker build -t oscinfra/tomcat:1.0
```

- Data Volume을 저장할 디렉터리를 호스트에 작성

```
mkdir -p /docker_mnt/tomcat/logs
chmod a+w /docker_mnt/tomcat/logs
```

- AP 서버 컨테이너를 생성, Tomcat을 시작

```
docker run -d -v /docker_mnt/tomcat/logs:/usr/share/tomcat/logs -p 8080:8080 --link db01:db --name
ap01 oscinfra/tomcat:1.0
```

* Dockerfile의 ENTRYPOINT에 정의 된 시작 명령은 docker run에서 다른 명령으로 변경할 수 없다.

감사합니다