

# 도커 컨테이너 만들기 🚢

**docker** 18.09.0

**docker-compose** 1.22.0

Kim Chungsub  
Creative Lab Director

 subicura

 subicura

 **purpleworks**

# Agenda

- 도커 설치하기 (Vagrant / Docker for Mac / Docker for Windows)
- 컨테이너 실행하기 `run`
- 컨테이너 목록 확인하기 `ps`
- 컨테이너 중지하기 `stop`
- 컨테이너 제거하기 `rm`
- 컨테이너 로그보기 `logs`
- 이미지 목록 확인하기 `images`
- 이미지 다운로드하기 `pull`
- 이미지 삭제하기 `rmi`
- 네트워크 만들기 `network`
- 볼륨 마운트 `-v`
- Docker Compose

---

## 도커 설치하기

# Linux

```
curl -s https://get.docker.com/ | sudo sh
```

명령어를 입력하고 패스워드를 넣으면 리눅스 배포판(ubuntu/centos/..)에 따라 자동으로 최신버전의 도커를 설치합니다.

# Vagrant

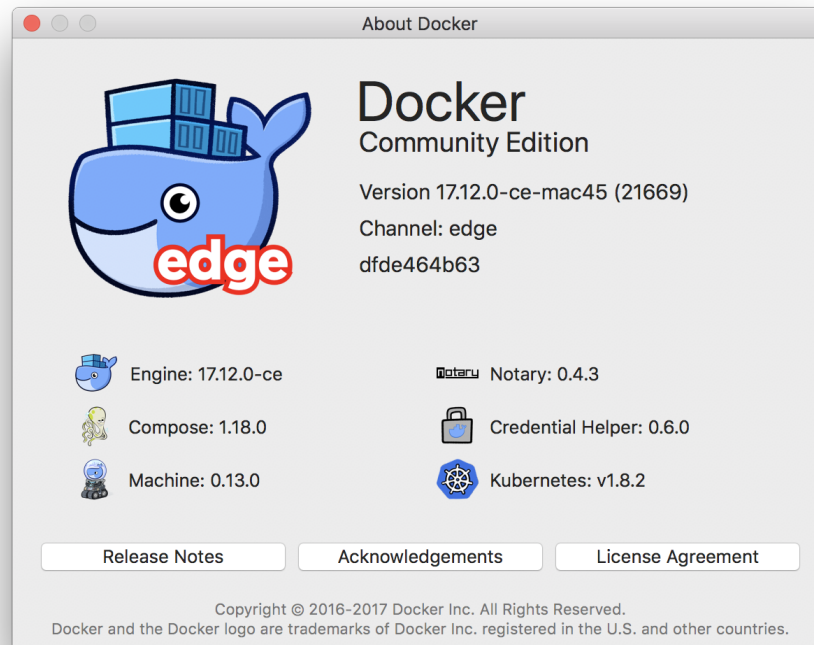
[virtualbox](#)와 [vagrant](#)를 설치합니다.

[Vagrantfile](#)을 다운로드 하고 다음 명령어를 입력합니다.

```
vagrant up  
vagrant ssh
```

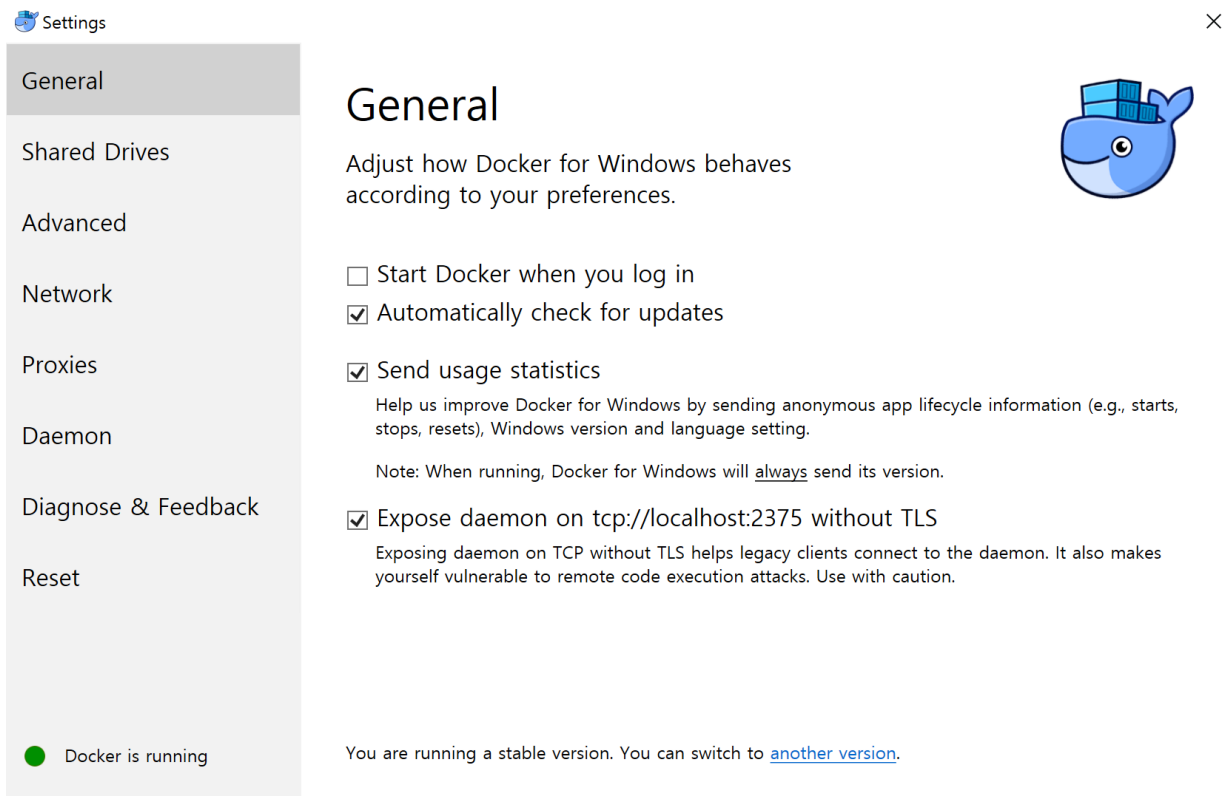
# Mac or Windows

[Docker for Mac](#) / [Docker for Windows](#)를 설치합니다.



# Docker for Windows

도커 데몬을 외부에서 접근할 수 있도록 **Expose daemon...** 옵션을 체크합니다.



The screenshot shows the 'Settings' application for Docker for Windows, specifically the 'General' tab. On the left is a sidebar with navigation options: General (selected), Shared Drives, Advanced, Network, Proxies, Daemon, Diagnose & Feedback, and Reset. At the bottom of the sidebar, a green dot indicates 'Docker is running'. The main content area is titled 'General' and includes a description: 'Adjust how Docker for Windows behaves according to your preferences.' To the right of the text is the Docker whale logo. Below the description are several settings: 'Start Docker when you log in' (unchecked), 'Automatically check for updates' (checked), 'Send usage statistics' (checked), and 'Expose daemon on tcp://localhost:2375 without TLS' (checked). Each checked option has a brief explanatory note. At the bottom, a status message says 'You are running a stable version. You can switch to [another version](#).'

Settings

General

Shared Drives

Advanced

Network

Proxies

Daemon

Diagnose & Feedback

Reset

Docker is running

### General

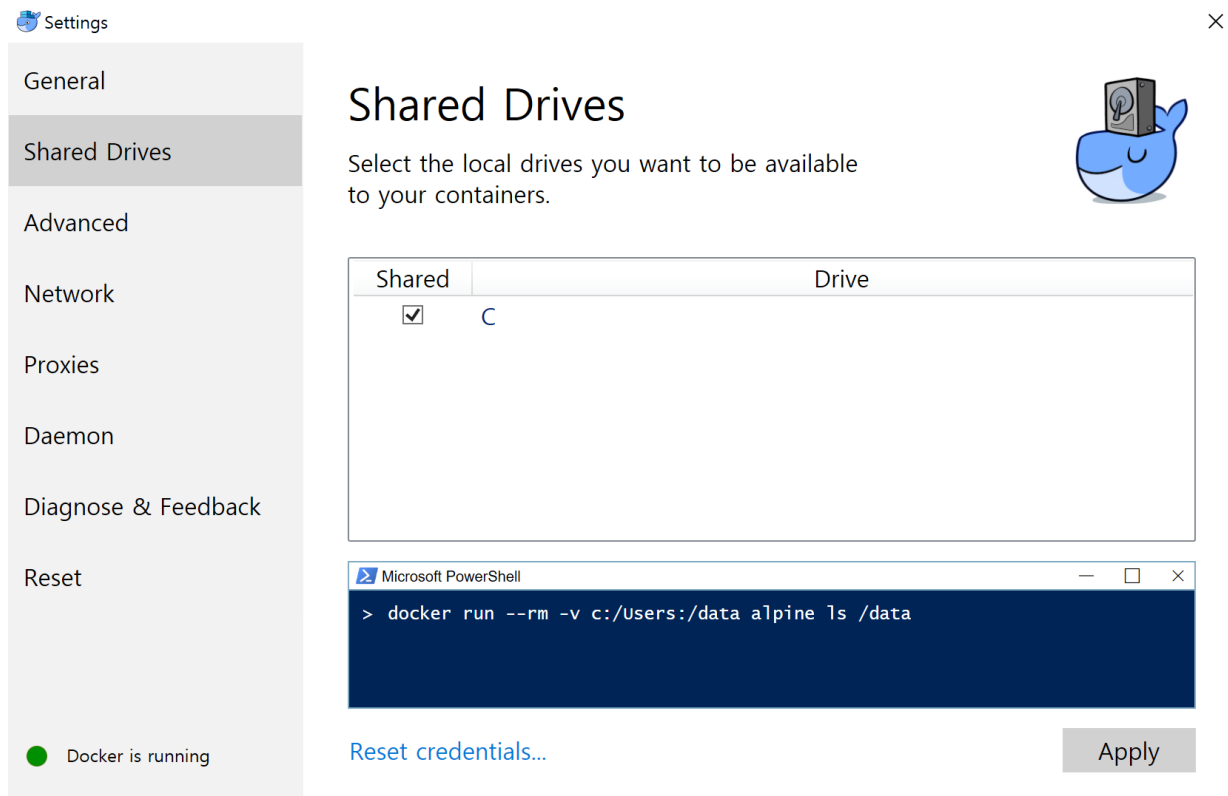
Adjust how Docker for Windows behaves according to your preferences.

- ☐ Start Docker when you log in
- ☒ Automatically check for updates
- ☒ Send usage statistics  
Help us improve Docker for Windows by sending anonymous app lifecycle information (e.g., starts, stops, resets), Windows version and language setting.  
Note: When running, Docker for Windows will always send its version.
- ☒ Expose daemon on tcp://localhost:2375 without TLS  
Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.

You are running a stable version. You can switch to [another version](#).

# Docker for Windows

볼륨을 마운트 할 수 있도록 공유 드라이브를 선택합니다.



The screenshot shows the Docker Desktop Settings window. On the left is a sidebar with navigation options: General, Shared Drives (selected), Advanced, Network, Proxies, Daemon, Diagnose & Feedback, and Reset. At the bottom of the sidebar, a green dot indicates 'Docker is running'. The main area is titled 'Shared Drives' with a subtitle 'Select the local drives you want to be available to your containers.' and a Docker whale icon. Below this is a table with two columns: 'Shared' and 'Drive'. The table contains one row with a checked checkbox in the 'Shared' column and 'C' in the 'Drive' column. At the bottom of the settings window, there is a 'Reset credentials...' link and an 'Apply' button. Overlaid on the bottom of the settings window is a Microsoft PowerShell terminal window with the command `> docker run --rm -v c:/Users:/data alpine ls /data`.

Settings

General

Shared Drives

Advanced

Network

Proxies

Daemon

Diagnose & Feedback

Reset

Docker is running

### Shared Drives

Select the local drives you want to be available to your containers.

Shared	Drive
<input checked="" type="checkbox"/>	C

Microsoft PowerShell

```
> docker run --rm -v c:/Users:/data alpine ls /data
```

[Reset credentials...](#) Apply

# Mac or Windows

도커는 linux만\* 지원하기 때문에 MacOS와 Windows에 설치되는 Docker는 실제로 가상머신에 설치됩니다. MacOS는 **xhyve** 를 사용하고 Windows는 **Hyper-V** 를 사용합니다.

터미널에서 입력하는 docker 명령어가 실제로 가상머신에서 실행된다는 점을 이해해야 합니다.

\* Docker for Windows는 Windows container를 지원하지만 여기서 다루지 않습니다.



# 설치확인

```
$ docker version
```

```
Client:
```

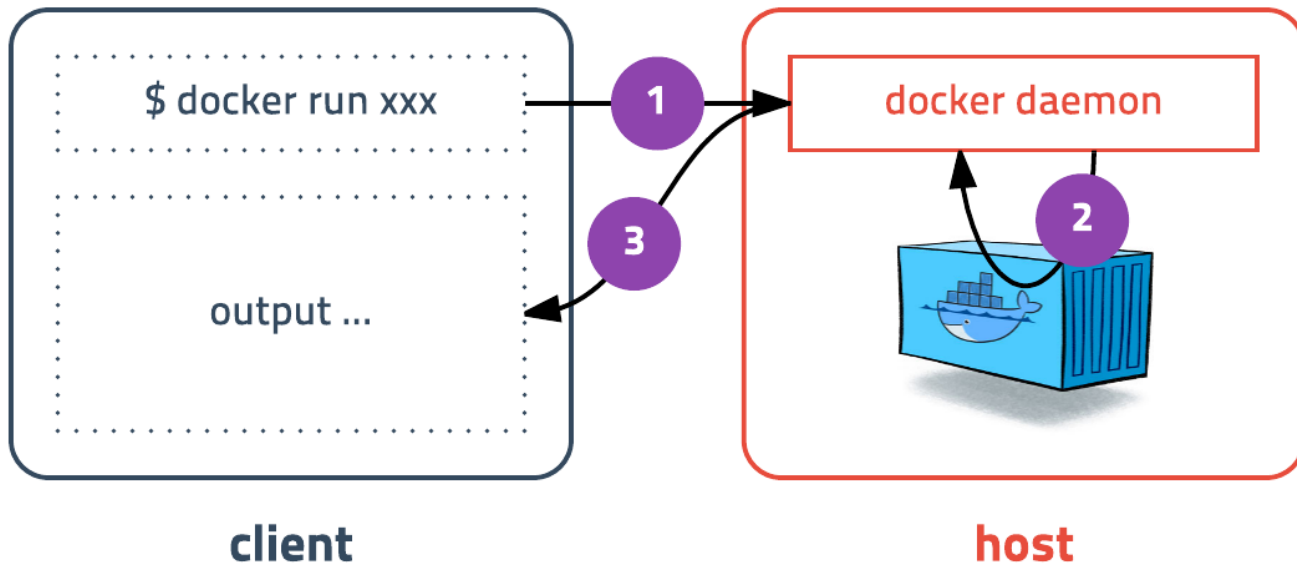
```
Version:      18.09.0
API version:   1.39
Go version:    go1.10.4
Git commit:    4d60db4
Built:         Wed Nov  7 00:49:01 2018
OS/Arch:       linux/amd64
Experimental:  false
```

```
Server: Docker Engine - Community
```

```
Engine:
```

```
Version:      18.09.0
API version:   1.39 (minimum version 1.12)
Go version:    go1.10.4
Git commit:    4d60db4
Built:         Wed Nov  7 00:16:44 2018
OS/Arch:       linux/amd64
Experimental:  false
```

# client(command) - server(daemon)



# run command

```
docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

### 옵션

### 설명

-d	detached mode 흔히 말하는 백그라운드 모드
-p	호스트와 컨테이너의 포트를 연결 (포워딩)
-v	호스트와 컨테이너의 디렉토리를 연결 (마운트)
-e	컨테이너 내에서 사용할 환경변수 설정
--name	컨테이너 이름 설정
--rm	프로세스 종료시 컨테이너 자동 제거
-it	-i와 -t를 동시에 사용한 것으로 터미널 입력을 위한 옵션
--network	네트워크 연결

# ubuntu 18.04

```
docker run ubuntu:18.04
```

**run**명령어를 사용하면 사용할 이미지가 저장되어 있는지 확인하고 없다면 다운로드(**pull**)를 한 후 컨테이너를 생성(**create**)하고 시작(**start**) 합니다.

컨테이너는 정상적으로 실행됐지만 뭘 하라고 명령어를 전달하지 않았기 때문에 컨테이너는 생성되자마자 종료됩니다. 컨테이너는 프로세스이기 때문에 실행중인 프로세스가 없으면 컨테이너는 종료됩니다.

# /bin/sh

```
docker run --rm -it ubuntu:18.04 /bin/sh
```

컨테이너 내부에 들어가기 위해 **sh** 을 실행하고 키보드 입력을 위해 **-it** (== -i -t) 옵션을 줍니다. 추가적으로 프로세스가 종료되면 컨테이너가 자동으로 삭제되도록 **--rm** 옵션도 추가하였습니다.

**--rm** 옵션이 없다면 컨테이너 종료되더라도 삭제되지 않고 남아있습니다.

# CentOS

```
docker run --rm -it centos:7 /bin/sh
```

도커는 다양한 리눅스 배포판을 실행할 수 있습니다. 공통점은 모두 동일한 커널<sup>kernel</sup>을 사용한다는 점입니다.

Ubuntu 또는 CentOS에 포함된 다양한 기본기능이 필요 없는 경우, Alpine이라는 초소형(약 5MB) 이미지를 사용할 수도 있습니다.

# Web Application

간단한 웹 애플리케이션을 컨테이너로 생성해봅니다.

```
docker run -d -p 4567:4567 subicura/docker-workshop-app:1
```

**detached mode**(백그라운드 모드)로 실행하기 위해 **-d** 옵션을 추가하고 **-p** 옵션을 추가하여 컨테이너의 포트를 호스트의 포트와 연결하였습니다.

브라우저를 열고 **localhost:4567**에 접속하면 컨테이너 아이디를 확인 할 수 있습니다.

**curl** 명령어로 확인해볼까요?

```
# mac & linux
curl localhost:4567
# windows
docker run --rm byrnedo/alpine-curl docker.for.win.localhost:4567
```

# Web Application v2

```
docker run -d \  
  -p 4568:4567 \  
  -e ENDPOINT=https://workshop-docker-kr.herokuapp.com/ \  
  -e PARAM_NAME=haha \  
  subicura/docker-workshop-app:2
```

호스트 포트를 **4568** 로 바꾸고 **2번 태그 이미지** 를 사용합니다. **-e** 옵션으로 환경변수를 설정해 주었습니다. **PARAM\_NAME** 에 본인의 이름이나 별명을 입력해보세요.

이번 이미지는 접속할 경우 접속기록을 <https://workshop-docker-kr.herokuapp.com/>에 남깁니다.



# Web Application v3

```
docker run -d \  
  -p 4569:4567 \  
  -e ENDPOINT=https://workshop-docker-kr.herokuapp.com/ \  
  -e PARAM_NAME=haha \  
  -e PARAM_MESSAGE=호호호 \  
  subicura/docker-workshop-app:3
```

**PARAM\_MESSAGE** 환경변수를 추가하고 이미지는 **3번** 태그를 사용합니다. 도커가 어떤지 메시지를 남겨주세요.

# Redis

redis는 메모리기반의 다양한 기능을 가진 스토리지입니다.

```
docker run --name=redis -d -p 1234:6379 redis
```

**telnet** 프로그램으로 테스트해봅니다.

```
# linux
telnet localhost 1234
# mac
docker run --rm -it mikesplain/telnet docker.for.mac.localhost 1234
# windows
docker run --rm -it mikesplain/telnet docker.for.win.localhost 1234

set hello world
+OK
get hello
$5
world
quit
```

# MySQL

가장 유명한 데이터베이스 중 하나입니다. [docker hub mysql](#)을 검색해서 어떤 옵션(환경변수)이 있는지 확인해 봅니다.

```
docker run -d -p 3306:3306 \  
  -e MYSQL_ALLOW_EMPTY_PASSWORD=true \  
  --name mysql \  
  mysql:5.7
```

mysql에 접속하여 database를 만듭니다.

```
docker exec -it mysql mysql  
create database wp CHARACTER SET utf8;  
grant all privileges on wp.* to wp@'%' identified by 'wp';  
flush privileges;  
quit
```

# exec

잠깐! `exec` 명령어를 사용했습니다. `exec` 명령어는 `run` 명령어와 달리 실행중인 도커 컨테이너에 접속할 때 사용하며 일반적으로 컨테이너 안에 ssh server등을 설치하지 않고 `exec`명령어로 접속합니다.

# Wordpress

워드프레스를 실행합니다.

```
# linux   WORDPRESS_DB_HOST=172.17.8.101
# mac     WORDPRESS_DB_HOST=docker.for.mac.localhost
# windows WORDPRESS_DB_HOST=docker.for.win.localhost
docker run -d -p 8080:80 \
  -e WORDPRESS_DB_HOST=172.17.8.101 \
  -e WORDPRESS_DB_NAME=wp \
  -e WORDPRESS_DB_USER=wp \
  -e WORDPRESS_DB_PASSWORD=wp \
  wordpress
```

컨테이너가 제대로 실행되었는지 웹 브라우저로 확인해봅니다.

```
# linux
172.17.8.101:8080
# mac / windows
localhost:8080
```

# Tensorflow

마지막으로 이렇게 활용할 수 있다라는 예제로 tensorflow를 실행해보도록 하겠습니다. tensorflow는 손쉽게 머신러닝을 할 수 있는 툴입니다.

```
docker run -it -p 8888:8888 tensorflow/tensorflow
```

---

## 컨테이너 목록 확인하기

# ps

실행중인 컨테이너 목록을 확인하는 명령어는 다음과 같습니다.

```
docker ps
```

중지된 컨테이너도 확인하려면 **-a** 옵션을 붙입니다.

```
docker ps -a
```

# stop

실행중인 컨테이너를 중지하는 명령어는 다음과 같습니다.

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

실행중인 컨테이너를 하나 또는 여러개 (띄어쓰기로 구분) 중지할 수 있습니다.



## 컨테이너 제거하기

# rm

종료된 컨테이너를 완전히 제거하는 명령어는 다음과 같습니다.

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

종료 명령어도 옵션은 특별한게 없습니다. 종료된 컨테이너를 하나 또는 여러개 삭제할 수 있습니다.

`mysql` 과 `wordpress` 를 제외하고 모두 삭제해주세요.

# logs

컨테이너가 정상적으로 동작하는지 확인하는 좋은 방법은 로그를 확인하는 것 입니다. 로그를 확인하는 방법은 다음과 같습니다.

```
docker logs [OPTIONS] CONTAINER
```

기본 옵션과 `-f`, `--tail` 옵션을 살펴봅니다.

---

## 이미지 목록 확인하기

# images

도커가 다운로드한 이미지 목록을 보는 명령어는 다음과 같습니다.

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

간단하게 도커 이미지 목록을 확인해보겠습니다.

```
docker images
```

## 이미지 다운로드하기

# pull

이미지를 다운로드하는 명령어는 다음과 같습니다.

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

ubuntu:14.04를 다운받아보겠습니다.

```
docker pull ubuntu:14.04
```

run명령어를 입력하면 이미지가 없을 때 자동으로 다운받으니 pull명령어를 언제 쓰는지 궁금할 수 있는데 pull은 최신버전으로 다시 다운 받습니다. 같은 태그지만 이미지가 업데이트 된 경우는 pull 명령어를 통해 새로 다운받을 수 있습니다.

---

## 이미지 삭제하기

# rmi

이미지를 삭제하는 방법은 다음과 같습니다.

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

images명령어를 통해 얻은 이미지 목록에서 이미지 ID를 입력하면 삭제가 됩니다. 단, 컨테이너가 실행중인 이미지는 삭제되지 않습니다. 컨테이너는 이미지들의 레이어를 기반으로 실행중이므로 당연히 삭제할 수 없습니다.

# network create

도커 컨테이너끼리 통신을 할 수 있는 가상 네트워크를 만듭니다.

```
docker network create [OPTIONS] NETWORK
```

**app-network** 라는 이름으로 wordpress와 mysql이 통신할 네트워크를 만듭니다.

```
docker network create app-network
```

# network connect

기존에 생성된 컨테이너에 네트워크를 추가합니다.

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

만들어 놓은 mysql에 네트워크를 추가합니다.

```
docker network connect app-network mysql
```

# run with network

워드프레스를 app-network에 속하게 생성하고 mysql을 IP가 아닌 **mysql** 로 바로 접근합니다.

```
docker run -d -p 8080:80 \  
  --network=app-network \  
  -e WORDPRESS_DB_HOST=mysql \  
  -e WORDPRESS_DB_NAME=wp \  
  -e WORDPRESS_DB_USER=wp \  
  -e WORDPRESS_DB_PASSWORD=wp \  
  wordpress
```

같은 네트워크에 속해 있으면 상대 컨테이너의 이름을 DNS로 조회하여 바로 접근 할 수 있습니다. 하나의 컨테이너는 여러개의 network에 속할 수 있으며 **Docker Swarm** 같은 클러스터에서 편리하게 사용할 수 있습니다.



# volume mount (-v)

mysql을 삭제후에 다시 실행합니다.

```
docker stop mysql
docker rm mysql
docker run -d -p 3306:3306 \
  -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
  --network=app-network \
  --name mysql \
  mysql:5.7
```

워드프레스를 접속하면 데이터베이스 오류가 발생합니다. 이전 데이터가 전부 초기화 되었습니다!



```
localhost:8080
```

# docker command (1/2)

<https://docs.docker.com/engine/reference/commandline/docker/>

- docker attach
- docker container
- docker cp
- docker diff
- docker export
- docker history
- docker image
- docker images
- docker import
- docker info
- docker inspect
- docker kill

## docker command (2/2)

- docker logs
- docker ps
- docker restart
- docker rm
- docker rmi
- docker run
- docker save
- docker search
- docker start
- docker stats
- docker stop
- docker top
- docker version

# 설치 확인

Docker for Mac / Windows는 기본으로 같이 설치됩니다.

```
$ docker-compose version  
  
docker-compose version 1.22.0, build f46880fe  
docker-py version: 3.4.1  
CPython version: 3.6.6  
OpenSSL version: OpenSSL 1.1.0f 25 May 2017
```

Linux는 다음 명령어로 설치합니다.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.22.0/docker-compose-$(uname -s)-  
sudo chmod +x /usr/local/bin/docker-compose
```

# docker-compose.yml

볼륨 마운트 옵션을 넣어서 wordpress와 database를 만듭니다. wordpress라는 디렉토리를 생성하고 docker-compose.yml 파일을 만듭니다.

```
version: '2'

services:
  db:
    image: mysql:5.7
    volumes:
      - ./mysql:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

뒤에 이어서..

# docker-compose.yml

```
wordpress:
  image: wordpress:latest
  volumes:
    - ./wp:/var/www/html
  ports:
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_PASSWORD: wordpress
```

# up

docker compose를 이용하여 mysql와 wordpress를 실행합니다.

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

# down

docker compose를 이용하여 mysql와 wordpress를 종료합니다.

```
docker-compose down
```



# Docker Compose

실제 운영환경에선 명령어를 입력하는 대신 대부분 Docker Compose를 사용합니다. Docker Compose는 docker의 거의 모든 기능을 사용할 수 있습니다.

Thanks 🙏