

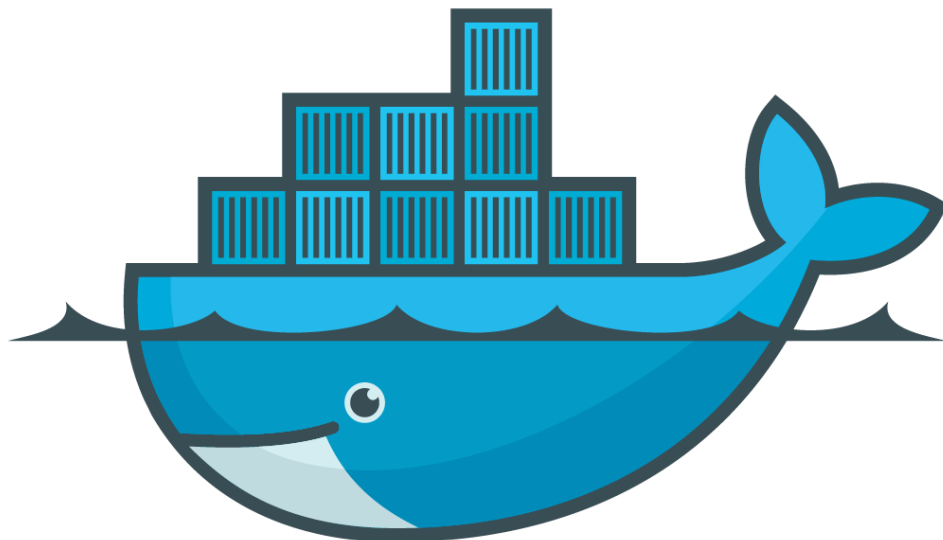
초보를 위한 도커 안내서 - 설치하고 컨테이너 실행하기

SERIES 2/3

subicura on 19 Jan 2017

좋아요 167개

공유하기



docker

docker logo

이 글은 초보를 위한 도커 안내서 - 설치부터 배포까지 2번째 글입니다. 이번엔 도커 설치부터 컨테이너를 실행하고 컨테이너를 둘러보는 방법에 대해 설명합니다. 도커에 대해 1도 모르는 분들을 위해 아주 가볍게 자주 쓰는 명령어만 다루었기 때문에 모든 명령어가 궁금하신 분은 [여기](#)를 참고해주세요.

- [초보를 위한 도커 안내서 - 도커란 무엇인가?](#) SERIES 1/3
- [초보를 위한 도커 안내서 - 설치하고 컨테이너 실행하기 ✓](#) SERIES 2/3
- [초보를 위한 도커 안내서 - 이미지 만들고 배포하기](#) SERIES 3/3



도커 설치하기

도커는 리눅스 컨테이너 기술이므로 macOS나 windows에 설치할 경우 가상머신에 설치가 됩니다. 리눅스 컨테이너 말고 윈도우즈 컨테이너라는 것도 존재하지만 여기서는 다루지 않습니다.

Linux

리눅스에 도커를 설치하는 방법은 자동 설치 스크립트를 이용하는 것이 가장 쉽습니다. 다음 명령어를 입력하면 root 권한을 요구하고 잠시 기다리면 설치가 완료됩니다. 음.. 참 쉽죠?

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



Docker Install (ubuntu)

sudo 없이 사용하기

docker는 기본적으로 root권한이 필요합니다. root가 아닌 사용자가 sudo없이 사용하려면 해당 사용자를 docker 그룹에 추가합니다.

- 1 `sudo usermod -aG docker $USER` # 현재 접속중인 사용자에게 권한주기
- 2 `sudo usermod -aG docker your-user` # your-user 사용자에게 권한주기

사용자가 로그인 중이라면 다시 로그인 후 권한이 적용됩니다.

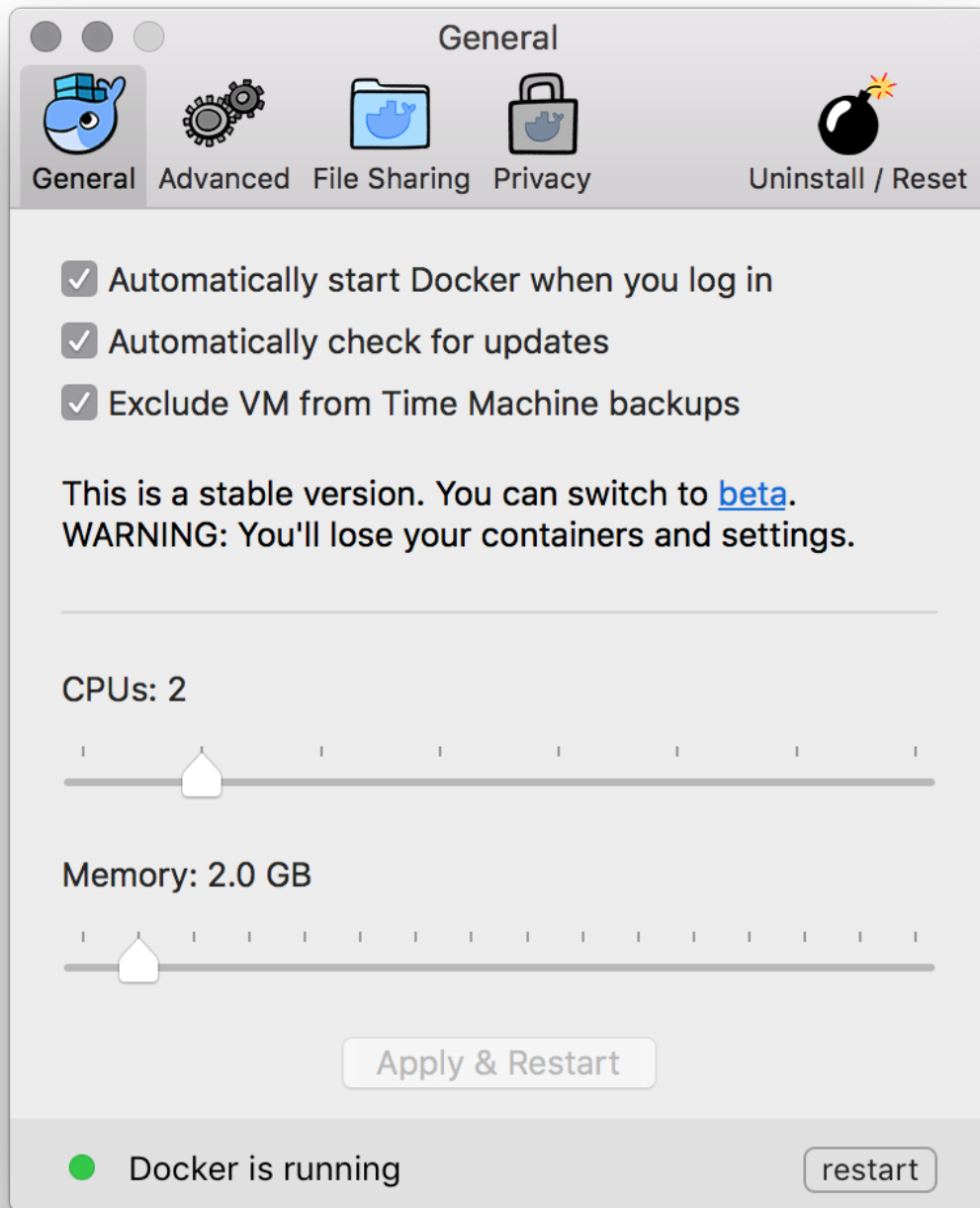
주의사항

- 도커를 실행하기 위한 kernel 버전은 3.10.x 이상입니다. ubuntu 14.04 이상을 사용하면 큰 문제가 없고 kernel의 버전이 낮을 경우 제대로 동작을 안하거나 문제가 생길 수 있습니다. 가급적 최신버전으로 업데이트 해주세요.
- ubuntu나 centos가 아닌 경우는 다른 방법이 필요합니다. 다른 리눅스를 쓰시는 분은 대부분 고오오급 개발자 분이시니 따로 설명하지 않아도 될 것 같아 [링크](#)로 대신하겠습니다. 절대 귀찮아서가 아님

Docker for Mac / Docker for Windows

도커를 맥이나 윈도우즈에 설치하려면 [Docker for mac](#) 또는 [Docker for windows](#)를 설치하면 됩니다. 파일을 다운받고 설치하고 재부팅하면 대부분 문제없이 완료됩니다. 소소한 옵션

ㄷ이 있는데 특별히 건드릴 부분은 없으나 한번 살펴보고 적절하게 설정하시면 됩니다. (windows는 공유 드라이브를 선택해주세요)



Docker for Mac

마치 네이티브스럽게 설치된 것 같지만 도커는 리눅스 컨테이너이므로 실제로는 가상머신에 설치가 되었습니다. 사용자는 가상머신을 사용한다는 느낌이 전혀 안드는데 그런부분을 굉장히 신경써서 설계하였습니다. 예를 들면, 포트를 연결하기 위해 도커 컨테이너의 특정 포트를 가상머신에 연결하고 다시 mac이나 windows의 포트와 연결해야 합니다. 디렉토리를 연결

하지만 디렉토리를 가상머신과 공유하고 그 디렉토리를 다시 컨테이너와 연결해야 합니다. 이 단계 추가적으로 거쳐야하는 부분을 자연스럽게 처리해줍니다.

docker for mac은 [xhyve](#)라는 macOS에서 제공하는 가상환경을 이용하고 docker for windows는 [Hyper-V](#)기능을 이용합니다. 따라서 OS가 최신버전이 아니면 동작하지 않을 수 있습니다.

가상머신에 설치하기

이런저런 이유로 Docker for ... 를 사용하지 못하는 경우 [Docker machine](#)을 이용할 수 있는데 처음 도커를 공부하는 경우에는 Virtual Box나 VMware같은 가상머신에 리눅스를 설치하고 리눅스에 접속하여 도커를 사용하는 방법을 권장합니다. 처음부터 Docker machine을 사용하면 환경이 약간 혼란스러울 수 있습니다.

설치확인하기

설치가 완료되었다면 정상적으로 설치되었는지 도커 명령어를 입력해 확인해 봅니다.

```
1 docker version
```

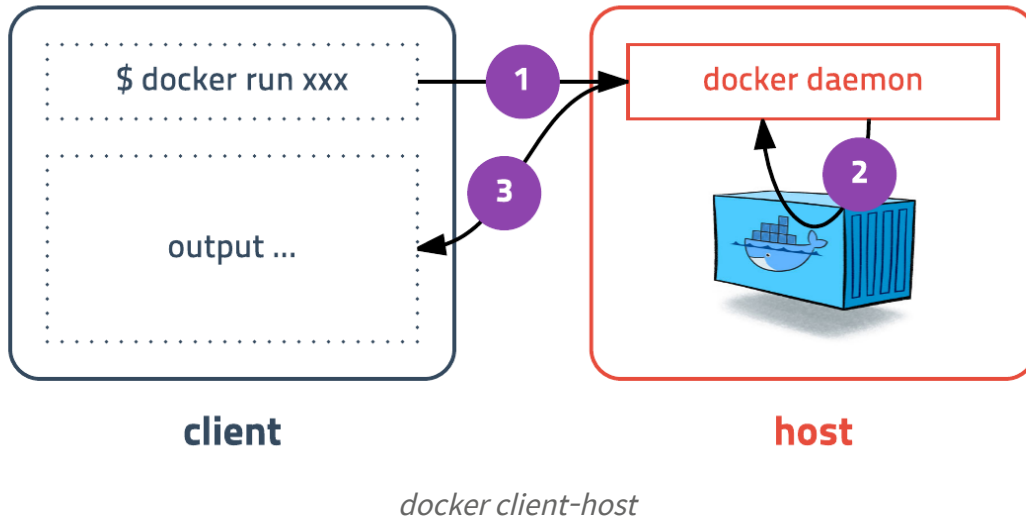
output:

```
1 Client:
2   Version:      1.12.6
3   API version:  1.24
4   Go version:   go1.6.4
5   Git commit:   78d1802
6   Built:        Wed Jan 11 00:23:16 2017
7   OS/Arch:      darwin/amd64
8
9 Server:
10  Version:      1.12.6
11  API version:  1.24
12  Go version:   go1.6.4
13  Git commit:   78d1802
14  Built:        Wed Jan 11 00:23:16 2017
15  OS/Arch:      linux/amd64
```

Client와 Server 정보가 정상적으로 출력되었다면 설치가 완료된 것입니다.

Server 정보가 안나오고 Error response from daemon: Bad response from Docker engine 이라는 메시지가 출력되는 경우는 보통 docker daemon이 정상적으로 실행되지 않았거나 sudo 를 입력하지 않은 경우입니다.

❗ 시, 특이한 부분을 찾으셨나요? 버전정보가 클라이언트와 서버로 나뉘어져 있습니다. 도커 하나의 실행파일이지만 실제로 클라이언트와 서버역할을 각각 할 수 있습니다. 도커 커맨드를 입력하면 도커 클라이언트가 도커 서버로 명령을 전송하고 결과를 받아 터미널에 출력해 줍니다.



기본값이 도커 서버의 소켓을 바라보고 있기 때문에 사용자는 의식하지 않고 마치 바로 명령을 내리는 것 같은 느낌을 받습니다. 이러한 설계가 mac이나 windows의 터미널에서 명령어를 입력했을때 가상 서버에 설치된 도커가 동작하는 이유입니다.

컨테이너 실행하기

이제! 드디어! 컨테이너를 실행해 보려고 합니다. 손이 근질근질하고 컨테이너의 강력크함을 보여드리고 싶기 때문에 여러개의 프로그램을 마구잡이로 손쉽게 띄워보겠습니다.

도커를 실행하는 명령어는 다음과 같습니다.

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

다음은 자주 사용하는 옵션들입니다.

옵션	설명
-d	detached mode 흔히 말하는 백그라운드 모드
-p	호스트와 컨테이너의 포트를 연결 (포워딩)
-v	호스트와 컨테이너의 디렉토리를 연결 (마운트)

옵션	설명
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
-link	컨테이너 연결 [컨테이너명:별칭]

엄청나게 직관적인 옵션으로 몇번 실행해보면 자연스럽게 익숙해집니다.

ubuntu 16.04 container

시작은 가볍게 ubuntu 16.04 컨테이너를 생성하고 컨테이너 내부에 들어가 봅니다.

```
1 docker run ubuntu:16.04
```

```

→ ~ docker run ubuntu:16.04
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu

b3e1c725a85f: Downloading  12.7 MB/50.22 MB
4daad8bdde31: Download complete
63fe8c0068a8: Download complete
4a70713c436f: Download complete
bd842a2105a8: Download complete
  
```

run ubuntu 16.04 container

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

위 예제는 ubuntu:16.04 이미지를 다운받은 적이 없기 때문에 이미지를 다운로드 한 후 컨테이너가 실행되었습니다. 컨테이너는 정상적으로 실행됐지만 뭘 하라고 명령어를 전달하지 않

되기 때문에 컨테이너는 생성되자마자 종료됩니다. 컨테이너는 프로세스이기 때문에 실행중 프로세스가 없으면 컨테이너는 종료됩니다.

이번에는 `/bin/bash` 명령어를 입력해서 `ubuntu:16.04` 컨테이너를 실행해 보겠습니다.

```
1  docker run --rm -it ubuntu:16.04 /bin/bash
2
3  # in container
4  $ cat /etc/issue
5  Ubuntu 16.04.1 LTS \n \l
6
7  $ ls
8  bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
9  boot  etc  lib   media  opt  root  sbin  sys  usr
```



run ubuntu 16.04 container with /bin/bash

컨테이너 내부에 들어가기 위해 `bash` 셸을 실행하고 키보드 입력을 위해 `-it` 옵션을 줍니다. 추가적으로 프로세스가 종료되면 컨테이너가 자동으로 삭제되도록 `--rm` 옵션도 추가하였습니다.

이번에는 바로 전에 이미지를 다운 받았기 때문에 이미지를 다운로드 하는 화면 없이 바로 실행되었습니다. `cat /etc/issue` 와 `ls` 를 실행해보면 `ubuntu` 리눅스인걸 알 수 있습니다. 뭔가 가벼운 가상머신 같나요?

`exit` 로 `bash` 셸을 종료하면 컨테이너도 같이 종료됩니다.


커를 이용하여 가장 기본적인 컨테이너를 순식간에 만들어 보았습니다.

redis container

2번째 컨테이너는 `redis` 입니다. redis는 메모리기반의 다양한 기능을 가진 스토리지입니다. 6379 포트로 통신하며 `telnet` 명령어로 테스트해 볼 수 있습니다. `redis` 컨테이너는 `detached mode`(백그라운드 모드)로 실행하기 위해 `-d` 옵션을 추가하고 `-p` 옵션을 추가하여 컨테이너의 포트를 호스트의 포트로 연결해보겠습니다.

`-d` 옵션이 없다면 프로세스가 `foreground`로 실행되어 아무키도 입력할 수 없게 됩니다. 컨테이너를 종료하려면 `ctrl + c` 를 입력해 주세요.

```
1    docker run -d -p 1234:6379 redis
2
3    # redis test
4    $ telnet localhost 1234
5    set mykey hello
6    +OK
7    get mykey
8    $5
9    hello
```



```
→ ~ docker run -d -p 1234:6379 redis
5dfff91d2cd859b48806231627aadf2066b9b10a8687c765b71ae6cae834ed751
→ ~ telnet localhost 1234
Trying ::1...
Connected to localhost.
Escape character is '^]'.
set mykey hello
+OK
get mykey
```

▶ 00:00

run redis container

1 옵션을 주었기 때문에 컨테이너를 실행하자마자 컨테이너의 ID(5dff91d2...)를 보여주 바로 쉘로 돌아왔습니다. 컨테이너는 종료된 것이 아니라 백그라운드 모드로 동작하고 있고 컨테이너 ID를 이용하여 컨테이너를 제어할 수 있습니다. -p 옵션을 이용하여 호스트의 1234포트 를 컨테이너의 6379포트 로 연결하였고 localhost의 1234포트로 접속하면 하면 redis를 사용할 수 있습니다.

테스트 결과 redis에 접속하여 새로운 키를 저장하고 불러오는데 성공했습니다. 실행이 간단한 건 물론이고 호스트의 포트만 다르게 하면 하나의 서버에 여러개의 redis 서버를 띄우는 것도 매우 간단합니다.

MySQL 5.7 container

3번째 실행할 컨테이너는 MySQL 서버 입니다. 가장 흔하게 사용하는 데이터베이스인데 이번에는 -e 옵션을 이용하여 환경변수를 설정하고 --name 옵션을 이용하여 컨테이너에 읽기 어려운 ID 대신 쉬운 이름을 부여할 예정입니다.

--name 옵션을 생략하면 도커가 자동으로 이름을 지어 줍니다. 이름은 유명한 과학자나 해커의 이름과 수식어를 조합하여 랜덤으로 **생성**합니다. (ex - boring_wozniak) 우리나라 과학자 **장영실**도 등록되어 있습니다.

[MySQL Docker hub](#) 페이지에 접속하면 간단한 사용법과 환경변수에 대한 설명이 있습니다. 여러가지 설정값이 있는데 패스워드 없이 root계정을 만들기 위해 MYSQL_ALLOW_EMPTY_PASSWORD 환경변수를 설정합니다. 그리고 컨테이너의 이름은 mysql 로 할당하고 백그라운드 모드로 띄우기 위해 -d 옵션을 줍니다. 포트는 3306포트 를 호스트에서 그대로 사용하겠습니다.

```

1  docker run -d -p 3306:3306 \
2      -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
3      --name mysql \
4      mysql:5.7
5
6  # MySQL test
7  $ mysql -h127.0.0.1 -uroot
8
9  mysql> show databases;
10  +-----+
11  | Database          |
12  +-----+
13  | information_schema |
14  | mysql              |
15  | performance_schema |
16  | sys                |
17  +-----+
18  4 rows in set (0.00 sec)

```

19

20 `mysql> quit`

```
→ ~ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql
mysql
```

00:00

run mysql 5.7 container

와우, 순식간에 MySQL서버가 실행되었습니다. 이번 테스트는 호스트 OS에 MySQL 클라이언트가 설치되어 있어야 합니다. 추후에 실행중인 MySQL 도커 컨테이너에 접속하여 클라이언트를 실행해 보도록 하겠습니다.

처음 접속 시도시 에러가 난 화면은 MySQL 서버가 최초로 실행되면서 준비작업을 하기 때문에 발생하는 에러입니다. 컨테이너를 실행하면 백그라운드에서 MySQL 서버를 띄우는 시간이 필요하기 때문에 잠시 후에 다시 시도 했을 때 정상적으로 접속된 걸 확인할 수 있습니다.

WordPress container

이번에는 블로그보다 웹용으로 더 흔하게 쓰이는 엔진으로 유명한 워드프레스를 실행합니다. 워드프레스는 데이터베이스가 필요하기 때문에 조금 복잡한 형태를 띄지만 크게 어렵지 않습니다. 바로 전에 생성했던 MySQL 컨테이너에 워드프레스 데이터베이스를 만들고 WordPress 컨테이너를 실행할 때 `--link` 옵션을 이용하여 MySQL 컨테이너를 연결하겠습니다.

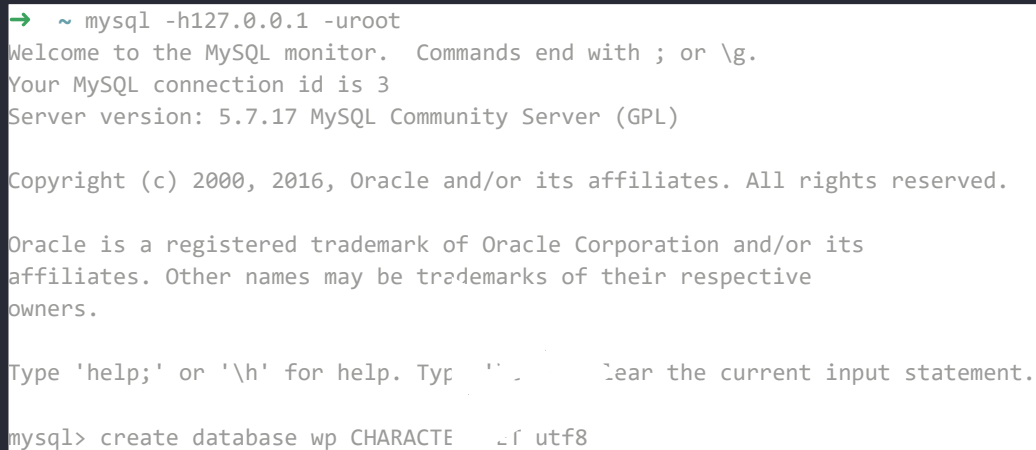
`--link` 옵션은 환경변수와 IP정보를 공유하는데 링크한 컨테이너의 IP정보를 `/etc/hosts`에 자동으로 입력하므로 워드프레스 컨테이너가 MySQL 데이터베이스의 정보를 알 수 있게 됩니다.

마지막으로, 워드프레스용 데이터베이스를 생성하고 워드프레스 컨테이너를 실행합니다. 호스트의 8080포트를 컨테이너의 80포트로 연결하고 MySQL 컨테이너와 연결한 후 각종 데이터베이스 설정 정보를 환경변수로 입력합니다.

```

1  # create mysql database
2  $ mysql -h127.0.0.1 -uroot
3  create database wp CHARACTER SET utf8;
4  grant all privileges on wp.* to wp@'%' identified by 'wp';
5  flush privileges;
6  quit
7
8  # run wordpress container
9  docker run -d -p 8080:80 \
10     --link mysql:mysql \
11     -e WORDPRESS_DB_HOST=mysql \
12     -e WORDPRESS_DB_NAME=wp \
13     -e WORDPRESS_DB_USER=wp \
14     -e WORDPRESS_DB_PASSWORD=wp \
15     wordpress

```



```

→ ~ mysql -h127.0.0.1 -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

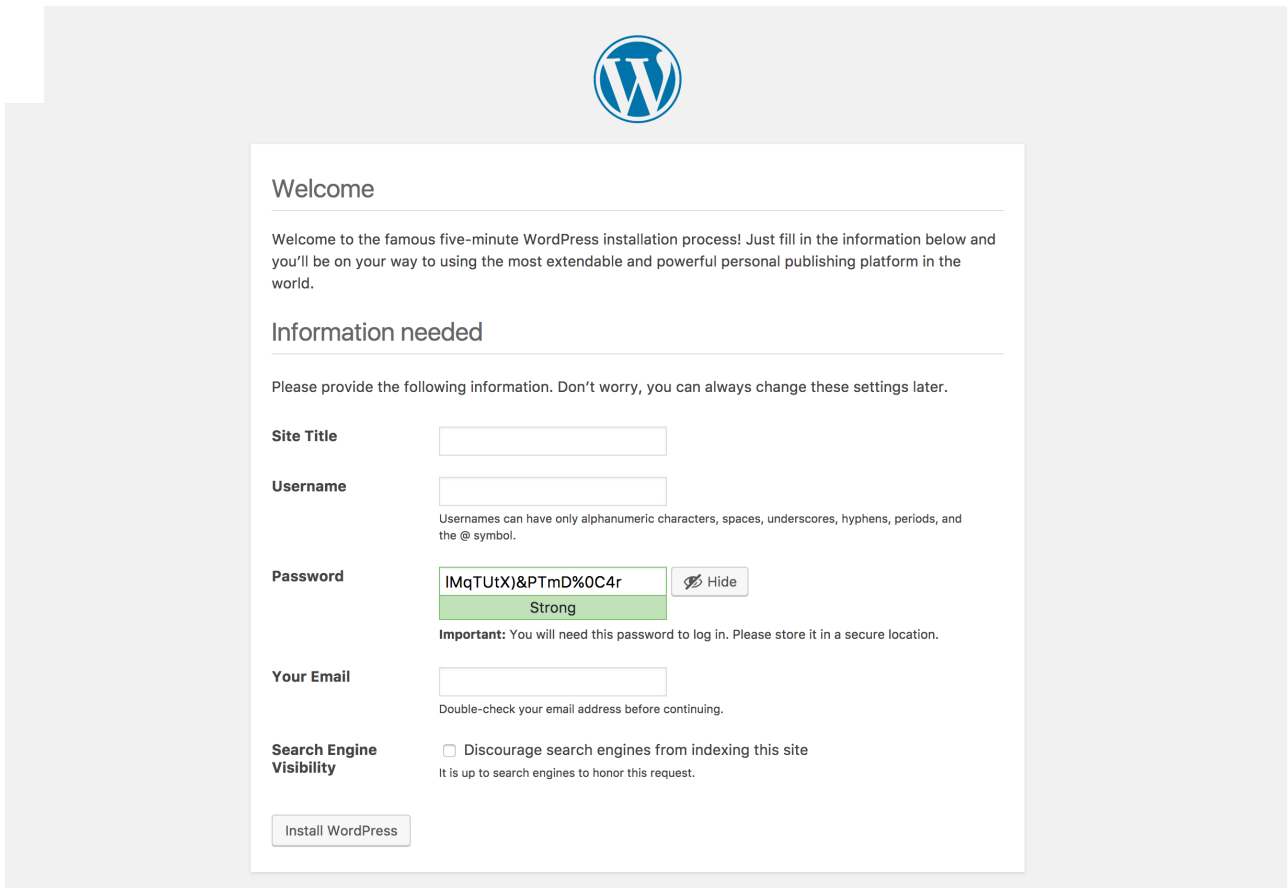
mysql> create database wp CHARACTER SET utf8

```

00:00

run wordpress container

컨테이너가 제대로 실행되었는지 웹 브라우저로 확인해봅니다.



wordpress setup

워드프레스가 실행되었습니다! 단지 이미지를 다운받고 적절한 환경변수를 입력하여 컨테이너를 실행했을 뿐입니다. 워드프레스 컨테이너 내부는 apache2와 php가 설치되어 있지만 추상화되어 있어 실행과정에선 드러나지 않습니다.

이번 예제는 테스트용으로만 사용해야 합니다. 운영 환경에서 사용하려면 추가적인 셋팅이 필요합니다. 이부분은 밑에서 다시 다룹니다.

--link 옵션은 deprecated 되어 곧 사용할 수 없습니다. 대신 Docker network 기능을 이용해야 하지만 쉬운 이해를 돕기 위해 사용하였습니다. ~~참고만 하고 실제 사용은 --~~

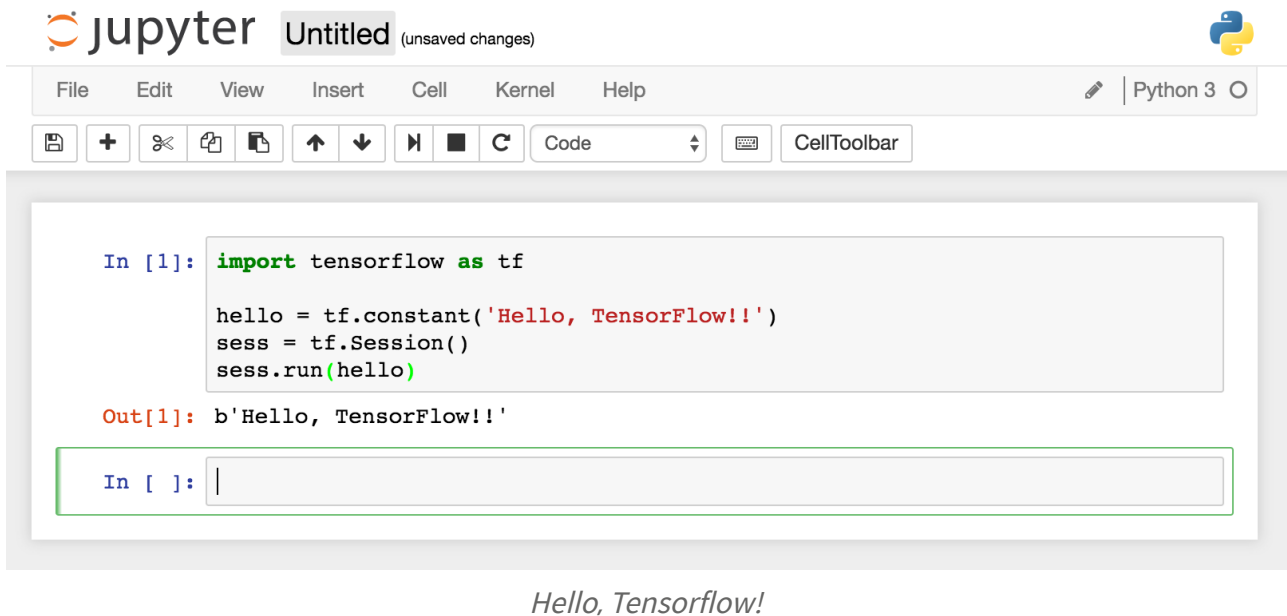
이제, 원하는 서비스가 있다면 이미지를 찾거나 직접 만들고, 어디서나 손쉽게 서비스를 실행할 수 있습니다.

tensorflow

마지막으로 이렇게 활용할 수 있다라는 예제로 tensorflow를 실행해보도록 하겠습니다. [tensorflow](#)는 손쉽게 머신러닝을 할 수 있는 툴입니다. tensorflow는 python으로 만들어져 python과 관련 패키지를 설치해야 합니다. 이번에 설치하는 이미지는 python과 함께 numpy, scipy, pandas, jupyter, scikit-learn, gensim, BeautifulSoup4, Tensorflow가 설치되어 있습니다. 뭔가 복잡해 보이지만 도커라면 손쉽게 실행해 볼 수 있습니다.

```
docker run -d -p 8888:8888 -p 6006:6006 teamlab/pydata-tensorflow:0.1
```

설치된 파일이 많아 다운로드 하는데 시간이 좀 걸립니다. 컨테이너가 실행되면 웹 브라우저에서 jupyter에 접속하여 머신러닝을 시작해 봅시다!



와우! 성공적으로 tensorflow 테스트를 마쳤습니다. 이제 조금 있으면 A.I를 만들 수 있을 것 같습니다.(?!)

여기까지 ubuntu, MySQL, redis, Wordpress, tensorflow를 실행해 보았습니다. 가상머신을 이용해서 동일한 작업을 했다면 컴퓨터가 엄청나게 버벅이기 시작했겠지만 컨테이너 기반의 도커를 이용하여 매우 가볍게 실행하고 있습니다. 내부 구조나 설치과정은 자세히 모르지만, 간단한 도커 명령어로 여러개의 서비스를 순식간에 실행하고 사용할 수 있다니 정말 짱짱맨입니다.

도커 기본 명령어

앞에서 도커의 `run` 명령어를 이용하여 여러개의 컨테이너를 실행했습니다. 이제 컨테이너의 상태를 살펴보고 어떤 이미지가 설치되어 있는지 확인하는 명령어를 알아봅니다.

컨테이너 목록 확인하기 (ps)

컨테이너 목록을 확인하는 명령어는 다음과 같습니다.

```
docker ps [OPTIONS]
```

이 단 기본옵션과 `-a`, `--all` 옵션만 살펴봅니다.

```
1 docker ps
```

output:

	CONTAINER ID	IMAGE	COMMAND	CREATED
1	6a1d027b604f	teamlab/pydata-tensorflow:0.1	"/opt/start"	About 2 minutes ago
2	52a516f87ceb	wordpress	"docker-entrypoint.sh"	8 minutes ago
3	2e2c569115b9	mysql:5.7	"docker-entrypoint.sh"	9 minutes ago
4	56341072b515	redis	"docker-entrypoint.sh"	16 minutes ago

`ps` 명령어는 실행중인 컨테이너 목록을 보여줍니다. detached mode로 실행중인 컨테이너들이 보입니다. 어떤 이미지를 기반으로 만들었는지 어떤 포트와 연결이 되어있는지 등 간단한 내용을 보여줍니다.

이번에는 `-a` 옵션을 추가로 실행해보겠습니다.

```
1 docker ps -a
```

output:

	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
1	6a1d027b604f	teamlab/pydata-tensorflow:0.1	"/opt/start"	2 minutes ago	Up About 2 minutes
2	52a516f87ceb	wordpress	"docker-entrypoint.sh"	9 minutes ago	Up 9 minutes
3	2e2c569115b9	mysql:5.7	"docker-entrypoint.sh"	10 minutes ago	Up 10 minutes
4	56341072b515	redis	"docker-entrypoint.sh"	18 minutes ago	Up 18 minutes
5	e1a00c5934a7	ubuntu:16.04	"/bin/bash"	32 minutes ago	Up 32 minutes

맨 처음 실행했다가 종료된 컨테이너(Exited (0))가 추가로 보입니다. 컨테이너는 종료되어도 삭제되지 않고 남아있습니다. 종료된 건 다시 시작할 수 있고 컨테이너의 읽기/쓰기 레이어는 그대로 존재합니다. 명시적으로 삭제를 하면 깔끔하게 컨테이너가 제거됩니다.

컨테이너 중지하기 (stop)

실행중인 컨테이너를 중지하는 명령어는 다음과 같습니다.

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

↪ 셸은 특별한게 없고 실행중인 컨테이너를 하나 또는 여러개 (띄어쓰기로 구분) 중지할 수 있습니다.

앞에서 실행한 tensorflow 컨테이너는 더이상 필요가 없으니 중지해 보겠습니다. 중지하려면 컨테이너의 ID 또는 이름을 입력하면 됩니다. tensorflow 컨테이너의 ID를 `ps` 명령을 통해 확인하고 중지해 봅니다.

```
1 docker ps # get container ID
2 docker stop ${TENSORFLOW_CONTAINER_ID}
3 docker ps -a # show all containers
```

도커 ID의 전체 길이는 64자리 입니다. 하지만 명령어의 인자로 전달할 때는 전부 입력하지 않아도 됩니다. 예를 들어 ID가 `abcdefgh...` 라면 `abcd` 만 입력해도 됩니다. 앞부분이 겹치지 않는다면 1-2자만 입력해도 됩니다.

잠시 기다리면 tensorflow 컨테이너가 종료됩니다. `ps -a` 명령어를 입력하여 종료되었는지 확인합니다.

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. 종료된 ubuntu 컨테이너와 tensorflow 컨테이너를 삭제해보겠습니다.

```
1 docker ps -a # get container ID
2 docker rm ${UBUNTU_CONTAINER_ID} ${TENSORFLOW_CONTAINER_ID}
3 docker ps -a # check exist
```

컨테이너가 말끔히 삭제되었습니다. 호스트 OS는 아무런 흔적도 남아있지 않고 컨테이너만 격리된 상태로 실행되었다가 삭제되었습니다. 시스템이 꼬일 걱정이 없습니다.

중지된 컨테이너를 일일이 삭제 하는 건 귀찮은 일입니다. `docker rm -v $(docker ps -a -q -f status=exited)` 명령어를 입력하면 중지된 컨테이너 ID를 가져와서 한번에 삭제합니다.

이미지 목록 확인하기 (images)

도커가 다운로드한 이미지 목록을 보는 명령어는 다음과 같습니다.


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

간단하게 도커 이미지 목록을 확인해보겠습니다.

```
1 docker images
```

output:

1	REPOSITORY	TAG	IMAGE ID	CREATED
2	wordpress	latest	b1fe82b15de9	43 hours ago
3	redis	latest	45c3ea2cecac	44 hours ago
4	mysql	5.7	f3694c67abdb	46 hours ago
5	ubuntu	16.04	104bec311bcd	4 weeks ago
6	teamlab/pydata-tensorflow	0.1	7bdf5d7e0191	6 months ago

이미지 주소와 태그, ID, 생성시점, 용량이 보입니다. 이미지가 너무 많이 쌓이면 용량을 차지하기 때문에 사용하지 않는 이미지는 지우는 것이 좋습니다.

이미지 다운로드하기 (pull)

이미지를 다운로드하는 명령어는 다음과 같습니다.

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

ubuntu:14.04 를 다운받아보겠습니다.

```
1 docker pull ubuntu:14.04
```

run 명령어를 입력하면 이미지가 없을 때 자동으로 다운받으니 pull 명령어를 언제 쓰는지 궁금할 수 있는데 pull 은 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...]
```

images 명령어를 통해 얻은 이미지 목록에서 이미지 ID를 입력하면 삭제가 됩니다. 단, 컨테이너가 실행중인 이미지는 삭제되지 않습니다. 컨테이너는 이미지들의 레이어를 기반으로 실행

❗중이므로 당연히 삭제할 수 없습니다.

tensorflow는 더 이상 사용하지 않으니 이미지를 제거해보겠습니다.

```
1 docker images # get image ID
2 docker rmi ${TENSORFLOW_IMAGE_ID}
```

output:

```
1 Untagged: teamlab/pydata-tensorflow:0.1
2 Untagged: teamlab/pydata-tensorflow@sha256:cb5e036afc0aa647a6fe1f384475275aeed946
3 Deleted: sha256:7bdf5d7e0191a6133a385ac69ec6a07be46b08cf5b5e6b826a89b9b47aadabe5
4 Deleted: sha256:9d72165f240630813a39e2f802a75f45caa2bc230881fa73c2f620c4b04686b5
5 Deleted: sha256:5c90ba76fff96b155b8a9446f2fb42d2ae04566832f3929df41dc52c0ae462ae
6 Deleted: sha256:1e23db6b02673d34c54fe87ec958ae8f2e310ca4aa911c7f8488b6c7988bfba0
7 Deleted: sha256:759641367a6788b55361f9eedcbc356f8e464ef794a883b5621fba6e23c6b18
8 Deleted: sha256:b5d017cd48c0d8461230b5ab0aed68e27a4018996344470db3c4b5adba10b49f
9 Deleted: sha256:470cdb158a12a416ffebde8d299fdbde28d56da062c701bf7a51d6484f777e97
10 Deleted: sha256:d136c3aea28597fadb667a084a6e5701e287fd36f03cc9555b0e5bca2e674f9b
11 Deleted: sha256:e62e6cce767baec3711dc1ec8cfcd76f68b88a6ca9ffcd2f3d22345048a0f8d7
12 Deleted: sha256:8e0761516c1c1ad792c9e67d1541d574905f067cb3b67e19a3af58ca7389eee9
13 Deleted: sha256:ad45f6539a6ce95216c9990d26b4b2e44c0b50637c1f0949b6965610b023fd97
14 Deleted: sha256:ebe200261801abef54e60699c098def3a4bd39a7b4833f164b99e23a88e8a98a
15 Deleted: sha256:698c310f0ec8a1f607a59f15a6affa7d9d21e21d9eaf6eb65b6ab6f33fcb62b4
16 Deleted: sha256:e5da0b4fd6d0143d7953e47a244a9be16919d1f4018e128e82fba6df967e790e
17 Deleted: sha256:7d8ffbfef2c1a55581e08b81f34506a746022e5beeb85f5848154b0b7d41208cf
18 Deleted: sha256:854cd556e16f5e5b21cd69f1e7c0d225c9b28eb5cf653aa4aa0937930298d72
19 Deleted: sha256:10e6f47feb1912a4d7a08b7652e360905a6ba8b76f188fd167c8d6afa09951a
20 Deleted: sha256:737ee41ad49072c9ba8301a17751e16d69c1045df74ac77f4ae05f849b08faec
21 Deleted: sha256:62b01cf4b1440085b0b9311372eb8acb509581faf655b2837a2e234c12dfd24
22 Deleted: sha256:7dcdb5fc9a2290d596e3bab8052c3141a69a451a29a98bd12678a7139891094
23 Deleted: sha256:8a9f069492b445dc889f9639572dc289f7c3fc1ac4d597a7468614fa2c624091
24 Deleted: sha256:34a63dcc0c5574597918a8e9df80e364f9fec88c0f4e25a0e5187acd241773ad
25 Deleted: sha256:af6fa43db2e379d3f0f3f36d61a0ece782133314c678d7223872eb7158f102ba
26 Deleted: sha256:efbafc7aa3a87241c10bf0c495672cf4a7b0cc2d23db06120ae1e13528316f82
27 Deleted: sha256:ea0c616be3659f71afce45eff2e28e51e04b25f1ae58b306521e6877f3adf2df
28 Deleted: sha256:f2dcc8902ea3d85aa9a954af894f59100fc00fa78c09b213c76dcf2727e3c3e
29 Deleted: sha256:3fe0adbd3614d696ded87bcb47aed81f49b113770a3778e276cd88abe9010b9f
30 Deleted: sha256:b88d260c0bcff3b4434a39ba33f58a2f9807400d664185057edd393133630242
31 Deleted: sha256:4de08f41d77237b2dbc0660036fe8352fdbacf6e079623d318003ed7e833a40a
32 Deleted: sha256:f32c8f02713c719181c2bf32be13dbd333932411d5fc3e0b6cfb780da8bb8124
33 Deleted: sha256:985d69571ed5d6d7e69bf58232fd8f30f8cd4ac77f41122a80fdab56347ec22d
34 Deleted: sha256:8509bc9969877f784fa6e704098cbcf77d694596644f389c5a2fefe5267878ab
35 Deleted: sha256:0ad5b9ed74fe2eee856b368d426cdd9335a6a6eccec0c4c705b5153c219a82692
36 Deleted: sha256:b159c847c5b9b899c23eba59395a4d7b7b275a8c686191ed6c4ef21ceadad8a2
37 Deleted: sha256:89e9f76552fd1a8d249c1e133e4c8964fe1f9104bb708db1e0ca6f6171ee4c22
38 Deleted: sha256:6d31e1827ca8f5037077314b2403eabd82590f9eae8baf956a2c2a819db68d4e
39 Deleted: sha256:7f4734de8e3dd402f10030f06bcb8781129b1eb6a25c58811a76d015f2a0982f
```

이미지가 삭제되었습니다. 이미지는 여러개의 레이어로 구성되어 있기 때문에 모든 레이어가 삭제된 것을 알 수 있습니다.

컨테이너 둘러보기

도커에 대한 아주아주아주 기본적인 명령어를 살펴보았습니다. 사실 저 명령어들과 이번에 살펴볼 `log`, `exec` 명령어를 익히면 도커에서 사용하는 명령어는 거의 다 익혔다고 할 수 있습니다. 다른 명령어는 필요에 따라 하나하나 살펴보면 됩니다.

컨테이너 로그 보기 (logs)

컨테이너가 정상적으로 동작하는지 확인하는 좋은 방법은 로그를 확인하는 것입니다. 로그를 확인하는 방법은 다음과 같습니다.

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

기본 옵션과, `-f`, `--tail` 옵션을 살펴봅니다.

기존에 생성해 놓은 워드프레스 컨테이너 로그를 확인해 보겠습니다.

```
1 docker ps
2 docker logs ${WORDPRESS_CONTAINER_ID}
```

output:

```
1 WordPress not found in /var/www/html - copying now...
2 Complete! WordPress has been successfully copied to /var/www/html
3 AH00558: apache2: Could not reliably determine the server's fully qualified domain name (yet)
4 AH00558: apache2: Could not reliably determine the server's fully qualified domain name (yet)
5 [Thu Jan 19 16:10:16.507735 2017] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.18 (Ubuntu) configured
6 [Thu Jan 19 16:10:16.507776 2017] [core:notice] [pid 1] AH00094: Command line: 'a
7 172.17.0.1 - - [19/Jan/2017:16:11:54 +0000] "GET / HTTP/1.1" 302 379 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
8 172.17.0.1 - - [19/Jan/2017:16:11:54 +0000] "GET /wp-admin/install.php HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
9 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/css/buttons.min.css HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
10 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/zxcvbn-async.min.js HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
11 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/css/install.min.css?ver=3.5.2 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
12 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/jquery/jquery.min.js?ver=3.1.1 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
13 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/jquery/jquery.js?ver=3.1.1 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
14 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/js/password-strength-meter.js?ver=3.5.2 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
15 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/wp-util.min.js?ver=3.5.2 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
16 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/js/user-profile.min.js?ver=3.5.2 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
17 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/underscore.min.js?ver=1.8.3 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
18 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/css/dashicons.min.css?ver=3.5.2 HTTP/1.1" 200 12345 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"
```

```

19 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/images/wordpress-logo.
20 172.17.0.1 - - [19/Jan/2017:16:12:02 +0000] "GET /wp-includes/js/zxcvbn.min.js HT
21 172.17.0.1 - - [19/Jan/2017:16:12:02 +0000] "GET /favicon.ico HTTP/1.1" 200 229 "
22 ::1 - - [19/Jan/2017:16:12:08 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4
23 ::1 - - [19/Jan/2017:16:12:09 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4
24 ::1 - - [19/Jan/2017:16:12:10 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4

```

컨테이너에서 실행한 로그가 쭉욱 보입니다. 아무 옵션을 주지 않았을 때는 전체 로그를 무식하게 전부 다 출력합니다. 너무 많으니 `--tail` 옵션으로 마지막 10줄만 출력해 보겠습니다.

```
1 docker logs --tail 10 ${WORDPRESS_CONTAINER_ID}
```

output:

```

1 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/wp-util.min.js?v
2 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/js/user-profile.min.js
3 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/js/underscore.min.j
4 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-includes/css/dashicons.min.c
5 172.17.0.1 - - [19/Jan/2017:16:12:01 +0000] "GET /wp-admin/images/wordpress-logo.
6 172.17.0.1 - - [19/Jan/2017:16:12:02 +0000] "GET /wp-includes/js/zxcvbn.min.js HT
7 172.17.0.1 - - [19/Jan/2017:16:12:02 +0000] "GET /favicon.ico HTTP/1.1" 200 229 "
8 ::1 - - [19/Jan/2017:16:12:08 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4
9 ::1 - - [19/Jan/2017:16:12:09 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4
10 ::1 - - [19/Jan/2017:16:12:10 +0000] "OPTIONS * HTTP/1.0" 200 126 "-" "Apache/2.4

```

마지막 10줄만 보니 좀 나아 보입니다. 이제 실시간으로 로그가 생성되는 걸 확인해보겠습니다. `-f` 옵션으로 실행합니다.

```
1 docker logs -f ${WORDPRESS_CONTAINER_ID}
```

```
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-admin/js/user-profile.min.js?ver=4.7.1 HTTP/1.1" 200 2590 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-includes/js/underscore.min.js?ver=1.8.3 HTTP/1.1" 200 6173 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-admin/js/password-strength-meter.min.js?ver=4.7.1 HTTP/1.1" 200 225 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-includes/js/jquery/jquery.js?ver=1.12.4 HTTP/1.1" 200 34120 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-includes/css/buttons.min.css?ver=4.7.1 HTTP/1.1" 200 1698 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
172.17.0.1 - - [19/Jan/2017:16:33:08 +0000] "GET /wp-admin/images/wordpress-logo.svg?ver=20131107 HTTP/1.1" 304 180 "http://localhost:8080/wp-admin/install.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/602.3.12 (KHTML, like Gecko) Version/10.0.2 Safari/602.3.12"
```

▶ 00:00

docker logs -f

로그를 켜 놓은 상태에서 워드프레스 페이지를 새로고침하면 브라우저 접속 로그가 실시간으로 보입니다. 가장 흔하게 사용하는 옵션이고 로그 보기를 중지하려면 `ctrl + c` 를 입력하면 됩니다.

로그에 대해 좀 더 자세히

프로그램마다 로그 파일은 제각각 생길텐데 어떻게 저 로그가 나올까 라는 의문이 생깁니다. 도커는 로그파일을 자동으로 알아채는게 아니라 [표준 스트림](#) Standard streams 중 `stdout` , `stderr` 를 수집합니다. 따라서 컨테이너에서 실행되는 프로그램의 로그 설정을 파일이 아닌 표준출력으로 바꾸어야 합니다. 단지 출력 방식만 바꾸는 것으로 모든 컨테이너는 로그에 대해 같은 방식으로 관리할 수 있게 됩니다.

또하나 중요한 점은 컨테이너의 로그파일은 json 방식으로 어딘가에 저장됩니다. 로그가 많으면 은근히 파일이 차지하는 용량이 커지므로 주의해야합니다. 도커는 다양한 플러그인을 지원하여 json이 아닌 특정 로그 서비스에 스트림을 전달할 수 있습니다. 어느 정도 앱의 규모가 커지면 기본적인 방식 대신 로그 서비스를 이용하는 걸 고려해야 합니다.

컨테이너 명령어 실행하기 (exec)

컨테이너를 관리하다 보면 실행중인 컨테이너에 들어가보거나 컨테이너의 파일을 실행하고 싶을 때가 있습니다. 컨테이너에 `ssh` 를 설치하면 되지 않을까? 라고 생각할 수 있지만 SSH 는 권장하지 않습니다. 하지 말라고 하면 꼭 하는 분들이 있던데 제발.. 예전에는 [nsenter](#) 라는 프로그램을 이용하였는데 docker에 `exec` 라는 명령어로 흡수되었습니다.

컨테이너 명령어를 실행하는 방법은 다음과 같습니다.

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

`run` 명령어와 유사해 보입니다. 차이는 `run` 은 새로 컨테이너를 만들어서 실행하고 `exec` 는 실행중인 컨테이너에 명령어를 내리는 정도입니다.

일단, 가볍게 실행중인 `mysql` 컨테이너에 접속해보겠습니다.

```
1  docker exec -it mysql /bin/bash
2
3  # MySQL test
4  $ mysql -uroot
5
6  mysql> show databases;
7  +-----+
8  | Database          |
9  +-----+
10 | information_schema |
11 | mysql              |
12 | performance_schema |
13 | sys                |
14 | wp                  |
15 +-----+
16 5 rows in set (0.00 sec)
17
18 mysql> quit
19 exit
```

```

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| wp                  |
+-----+
5 rows in set (0.00 sec)

```

▶ 00:00

docker exec bash

키보드 입력이 필요하니 run 명령어와 마찬가지로 `-it` 옵션을 주었고 bash 셸로 접속하여 마치 가상머신에 들어온 것 같은 느낌이 듭니다. 접속한 이후에는 어떤 작업도 할 수 있고 컨테이너를 마음껏 건드릴 수 있습니다.

셸로 완전한 권한을 얻는 방법 말고 바로 `mysql` 명령어를 실행 할 수도 있습니다.

```

1  docker exec -it mysql mysql -uroot
2
3  # MySQL test
4  $ mysql -uroot
5
6  mysql> show databases;
7  +-----+
8  | Database          |
9  +-----+
10 | information_schema |
11 | mysql              |
12 | performance_schema |
13 | sys                |
14 | wp                  |
15 +-----+
16 5 rows in set (0.00 sec)
17
18 mysql> quit

```

```
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| wp |
+-----+
5 rows in set (0.00 sec)

mysql> quit
Bye
```

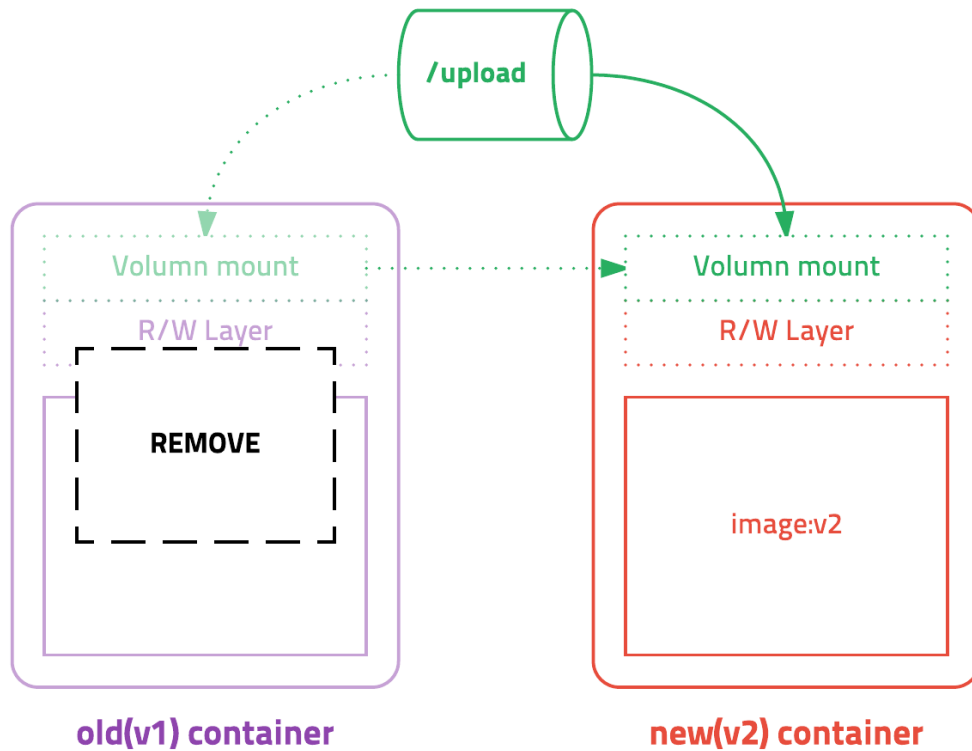
▶ 00:00

docker exec mysql

이제, 호스트 OS에 mysql을 설치하지 않아도 mysql 클라이언트를 사용할 수 있습니다. 굳이 복잡한 작업이 필요 없는 경우는 `-it` 옵션없이 단순히 명령을 실행하고 종료할 수도 있습니다.

컨테이너 업데이트

이제 지금까지 배운 모든걸 정리해서 컨테이너를 새로운 버전으로 업데이트 하는 과정을 살펴 보겠습니다.



도커 컨테이너 업데이트

도커에서 컨테이너를 업데이트 하려면 새 버전의 이미지를 다운(`pull`)받고 기존 컨테이너를 삭제(`stop` , `rm`) 한 후 새 이미지를 기반으로 새 컨테이너를 실행(`run`)하면 됩니다. 배포와 관련된 자세한 사항은 다음글에서 이야기하고 여기선 그냥 그렇구나 하고 이해합시다.

컨테이너를 삭제한다는 건 컨테이너에서 생성된 파일이 사라진다는 뜻입니다. 데이터베이스라면 그동안 쌓였던 데이터가 모두 사라진다는 것이고 웹 어플리케이션이라면 그동안 사용자가 업로드한 이미지가 모두 사라진다는 것입니다. ㄷㄷ

이런 상황도 커 도입했다가 퇴사를 방지하기 위해 컨테이너 삭제시 유지해야하는 데이터는 반드시 컨테이너 내부가 아닌 외부 스토리지에 저장해야 합니다. 가장 좋은 방법은 [AWS S3](#) 같은 클라우드 서비스를 이용하는 것이고 그렇지 않으면 데이터 볼륨 `Data volumes` 을 컨테이너에 추가해서 사용해야 합니다. 데이터 볼륨을 사용하면 해당 디렉토리는 컨테이너와 별도로 저장되고 컨테이너를 삭제해도 데이터가 지워지지 않습니다.

데이터 볼륨을 사용하는 방법은 몇가지가 있는데 여기서는 호스트의 디렉토리를 마운트해서 사용하는 방법에 대해 알아봅니다. `run` 명령어에서 소개한 옵션중에 `-v` 옵션을 드디어 사용해 보겠습니다. MySQL이라면 `/var/lib/mysql` 디렉토리에 모든 데이터베이스 정보가 담기므로 호스트의 특정 디렉토리를 연결해주면 됩니다.

```
1 # before
2 docker run -d -p 3306:3306 \
3   -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
```

```

4      --name mysql \
5      mysql:5.7
6
7      # after
8      docker run -d -p 3306:3306 \
9          -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
10         --name mysql \
11         -v /my/own/datadir:/var/lib/mysql \ # <- volume mount
12         mysql:5.7

```

위 샘플은 호스트의 `/my/own/datadir` 디렉토리를 컨테이너의 `/var/lib/mysql` 디렉토리로 마운트 하였습니다. 이제 데이터베이스 파일은 호스트의 `/my/own/datadir` 디렉토리에 저장되고 컨테이너를 삭제해도 데이터는 사라지지 않습니다. 최신버전의 MySQL 이미지를 다운받고 다시 컨테이너를 실행할 때 동일한 디렉토리를 마운트 한다면 그대로 데이터를 사용할 수 있습니다. **만세!**

Docker Compose

지금까지 도커를 커맨드라인에서 명령어로 작업했습니다. 지금은 간단한 작업만 했기 때문에 명령이 길지 않지만 컨테이너 조합이 많아지고 여러가지 설정이 추가되면 명령어가 금방 복잡해집니다.

도커는 복잡한 설정을 쉽게 관리하기 위해 [YAML](#) 방식의 설정파일을 이용한 [Docker Compose](#)라는 툴을 제공합니다. 깊게 파고들면 은근 기능이 많고 복잡한데 이번에는 아주 가볍게 다루고 지나가도록 하겠습니다.

설치하기

Docker for Mac 또는 Docker for Windows를 설치했다면 자동으로 설치됩니다. 리눅스의 경우 다음 명령어를 입력하여 설치합니다. 그냥 설치파일 하나 다운받으면 됩니다. **GoLang 짱**

```

1      curl -L "https://github.com/docker/compose/releases/download/1.9.0/docker-compose"
2      chmod +x /usr/local/bin/docker-compose
3      # test
4      docker-compose version

```

wordpress 만들기

기존에 명령어로 만들었던 wordpress를 compose를 이용해 만들어 보겠습니다.

먼저 빈 디렉토리를 하나 만들고 `docker-compose.yml` 파일을 만들어 설정을 입력합니다.

```
1  version: '2'
2
3  services:
4    db:
5      image: mysql:5.7
6      volumes:
7        - db_data:/var/lib/mysql
8      restart: always
9      environment:
10        MYSQL_ROOT_PASSWORD: wordpress
11        MYSQL_DATABASE: wordpress
12        MYSQL_USER: wordpress
13        MYSQL_PASSWORD: wordpress
14
15    wordpress:
16      depends_on:
17        - db
18      image: wordpress:latest
19      volumes:
20        - wp_data:/var/www/html
21      ports:
22        - "8000:80"
23      restart: always
24      environment:
25        WORDPRESS_DB_HOST: db:3306
26        WORDPRESS_DB_PASSWORD: wordpress
27  volumes:
28    db_data:
29    wp_data:
```

docker-compose.yml hosted with ❤ by GitHub

[view raw](#)

몇몇 생소해보이는 설정이 눈에 보이지만, 일단 실행해 봅시다.

```
1  docker-compose up
```

*docker-compose*

와우, 아주 손쉽게 워드프레스가 만들어 졌습니다. 단지 명령어를 설정파일로 바꾼거에 불과하지만 가독성과 편리성은 훨씬 향상되었습니다.

Docker Compose의 다른 기능과 생소한 설정내용은 속제로 남겨드립니다. 원래 개발공부라는게 왜만큼 했다고 생각하면 또 다르게 나오고 끊임없이 공부해야 하는 분야입니다. **화이팅!** 도커에 대해 이해를 했다면 Docker Compose 또한 쉽게 사용할 수 있을 것입니다.

정리

여기까지 도커에 대해 기본적인 내용부터 컨테이너를 실행하고 살펴보는 방법까지 알아보았습니다. 도커가 어떤건지, 컨테이너가 뭔지, 이미지가 뭔지 감이 좀 오시나요? 이제 남이 만든 이미지를 사용하는 것이 아니라 직접 이미지를 만들고 컨테이너를 여러 서버로 배포하는 방법을 알아봐야하는데... 다음글에서 알아보도록 하겠습니다.

초보를 위한 도커 안내서 - 인프런

도커를 1도 모르는 입문자, 초보자분들을 위한 도커 안내서 입니다. 복잡한 내용을 제외하고 도커가 왜 인기가 많고 어떻게 사용하는지 빠르게 익힐 수 있도록 집중하였습니다. 입문 서버

<https://www.inflearn.com/course/도커-입문>



docker

초보를 위한 도커안내서

[이제 도커안내서를 영상으로 만나보세요!](#)

READ THIS NEXT

초보를 위한 도커 안내서 - 이미지 만들고 배포하기

이 글은 초보를 위한 도커 안내서 - 설치부터 배포까지 시리즈의 마지막 글입니다. 지난 글에서 도커를...

YOU MIGHT ENJOY

초보를 위한 도커 안내서 - 도커란 무엇인가?

도커를 처음 접하는 시스템 관리자나 서버 개발자를 대상으로 도커 전반에 대해 알고 넓은 지식을 담고...

Subicura's Blog © 2020

[초보를 위한 도커 안내서 - 도커란 무엇인가?](#)

SERIES 1/3

- **초보를 위한 도커 안내서 - 설치하고 컨테이너 실행하기 ✓** SERIES 2/3
- [초보를 위한 도커 안내서 - 이미지 만들고 배포하기](#) SERIES 3/3

[Docker](#)
[DevOps](#)
[Server](#)
[Container](#)
[좋아요 167개](#)
[공유하기](#)


WRITTEN BY

SUPPORTED BY

[subicura](#)

Published 19 Jan 2017

Proudly published with Jekyll

[Feedly 구독하기](#)
[RSS 구독하기](#)
[Email 구독하기](#)

댓글 32개

[정렬 기준](#)
[인기순](#)


댓글 달기...

**정재훈**

정말로 좋은자료 감사합니다. 많은 도움이 되고 있습니다.

[좋아요](#)
[· 답글 달기](#)
[· 24주](#)
**박상희**

선생님,,,진짜 도움 많이 되었습니다...감사합니다ㅠ

[좋아요](#)
[· 답글 달기](#)
[· 25주](#)
**황선희**



빠르게 맛보기할 수 있네요! 쉽게 알려주셔서 감사합니다!!

좋아요 · 답글 달기 · 39주



권혁준

이해하기 엄청 편하네요. 감사합니다.

좋아요 · 답글 달기 · 43주



김정우

안녕하세요! 좋은자료 정말 감사드립니다 ππ 맨 처음 mysql 설치후 워드프레스 예제 작업하다가 오류가있어서 기존 생성된 모든컨테이너 삭제 후에 다시 진행하고 있는데 mysql 예제에서 그대로 포트번호를 -p 3306:3306으로 지정해서 실행시키면 address already in use 에러가 나고 다른 포트로 만드니까 mysql -h127.0.0.1 -uroot 부분에서 접속 거부가 뜨는데 원인을 알수있을까요? π

좋아요 · 답글 달기 · 46주



김지형

좋은 포스팅 잘 읽고 갑니다. 상세한 설명 감사합니다 가능하다면 제 대학 학술동아리 카페에 본 내용을 참고해서 문서를 올려도 될까요?

좋아요 · 답글 달기 · 1년



주강호

죄송하지만 제가 아직 프알못이라, 똑같이 따라 쳤는데 워드프레스를 로컬호스트에서 확인하려고 하니 디비가 없다고 뜨네요 ㅜㅜ localhost:8080에서 확인하는 것이 맞나요?

좋아요 · 답글 달기 · 1년



주강호

정말 감사합니다. 잘보고 갑니다.

좋아요 · 답글 달기 · 1년



Hyunjong Cha

좋은 글 너무 감사드립니다.

라즈베리파이에 docker를 사용 중 인데요.

굉장히 버전이 낮아서요.

현재 사용 중인 Container들에 영향이 없이 Upgrade하는 방법이 있을까요?

좋아요 · 답글 달기 · 1년



최승원

안녕하십니까!! 정말 좋은자료입니다. 최근에 처음 Docker를 이용해서 ELK 구축중인데 많은 도움 됐습니다. 감사합니다.

제가 개인 블로그를 하고있는데 출처를 남기고 블로그에 정리해도 되는지 여쭙보고 싶습니다!

좋아요 · 답글 달기 · 1년

댓글 10개 더 읽어들이기

Facebook 댓글 플러그인