**컨테이너 운영방식.**

동일한 Image를 바탕으로 기동된 Container는 각각의 Container Layer만 추가되며, Image Layer는 공유하여 사용한다는 사실을 알 수 있습니다(실제로 Docker는 동일한 Image에서 기동된 Container들은 Image Layer를 100%공유하여 사용합니다). 또한, Container는 개별적으로 Container Layer를 갖고 해당 Layer에 모든 변경된 내용이 저장되기 때문에, 여러 Container가 동일한 기본 Image를 공유하여 사용하면서 Container 자신의 Data 상태를 가질 수 있습니다. 이러한 특성으로 인해, Docker Container는 더욱 가벼워지고 기동 속도가 빨라질 수 있는 것입니다.

동작중인 컨테이너 확인

docker ps

정지된 컨테이너 확인

docker ps -a

컨테이너 삭제

docker rm [컨테이너ID] OR [이름]

실행 컨테이너 정지

docker stop [컨테이너ID] OR [이름]

-d : 백그라운드 모드

"-d" 옵션이 Docker의 컨테이너를 백그라운드 프로세스로 실행하는 옵션이다.

-e : 환경변수 설정

--name : 컨테이너에 ID 대신 이름을 부여

docker: Error response from daemon: Conflict. The container name "/ubuntu\_test" is already in use by container

docker 컨테이너를 실행한 다음 동일한 명령을 실행하면 다음과 같은 에러가 나타나는데 이것은 docker run 명령이 "create" 와 "start"

명령을 한번에 실행시키는 명령이기 때문에 create 시 이미 동일한 이름의 컨테이너가 존재하기 때문에 발생하는 문제이다.

이 경우 docker rm <id or name>으로 삭제한 후 다시 실행하면 된다.

i : Interactive 모드로 표준입력과 표준출력을 키보드와 화면을 통해 가능하도록 하는 옵션이다.

t: 텍스트 기반의 터미널(TTY)을 애뮬레이션해주는 옵션이다.

-it 명령어를 붙인 Docker를 처음 접하는 개발자는 또 한번 혼란스러움을 겪을 수 있다.

위 예제와 같이 shell 이 나오고 내가 필요한 명령이 사용 가능하게 되면 마치 우분투 서버가 실행되었다는 착각을 하게 된다.

그리고 해당 shell에서 tomcat이나 rails와 같은 애플리케이션 서버를 설치하고, 실행해본다. 잘 돌아간다.

문제는 이렇게 한 다음 shell에서 "exit" 를 입력하여 shell에서 나오는 순간 컨테이너는 다시 중지된다.

명령을 영원히 실행하게 하면 됨.

*$ docker run -d --name ubuntu\_test ubuntu /bin/bash****-c "while true; do echo "still live"; sleep 100; done"***

-c : -c 에 있는 옵션은 무한 루프를 돌면서 100초마다 한번씩 "still live"를 출력하는 기능을 수행하는 shell script 이다.

이 컨테이너에 이제 attach 해보자.

attach 명령을 실행해보면 아무 반응도 없고 100초마다 "still live"만 출력하는 것을 볼 수 있다.

그렇다고 shell prompt가 나타나지도 않는다. 이것은 attach 명령어가 stdout, stderr을 가져오는 것이기 때문에 당연한 것이다.

*$* ***docker exec -it eb3b9e69b18d /bin/bash***

Shell에 접속하기 위해서는 attach가 아닌 exec 명령을 이용해서 컨테이너의 쉘 환경에 접속할 수 있다.

이렇게 접속한 다음 exit 명령을 이용하여 shell을 빠져 나와도 컨테이너는 정상적으로 동작하는 것을 알 수 있다.

지금까지 내용을 종합해보면 Docker의 컨테이너 내에 애플리케이션 서버를 실행하려면 애플리케이션 서버가 무한루프로 동작하게 해야 한다.

하지만 이미 애플리케이션 서버들은 무한루프로 동작하는 프로그램들이다. 따라서 다음 내용만 주의하면 된다.

Tomcat의 경우 예를 들면 일반적으로 다음과 같이 실행하여 백그라운드 모드로 동작하게 한다.

이유는 tomcat 서버를 실행시킨 shell이 종료되더라도 tomcat 서버는 정상적으로 계속 동작하게 하기 위해서이다.

*$ cd $CATALINA\_HOME/bin; $ ./catalina.sh start;*

하지만 docker 환경에서 이렇게 하면 컨테이너가 바로 종료되어 Tomcat 서버가 죽는 것과 동일한 상황을 맞게 된다.

다음과 같이 fore ground로 실행해야 한다.

*$ cd $CATALINA\_HOME/bin; $****./catalina.sh run;***

필자가 선택한 방법은 서비스 개발 후 초기 얼마 동안은 다음과 같은 형태로 컨테이너를 구성하였다.

* 애플리케이션 서버는 백그라운드로 실행
* 애플리케이션 서버 실행 후 바로 shell script로 무한 반복 스크립트 실행

이렇게 하면 설정 변경 후 재시작해도 Docker 컨테이너는 여전히 계속 살아 있고 변경된 옵션에 대해 확인할 수 있다.

이 방법은 서비스 초기 어떤 옵션이 서비스에 맞는지 확인하는 용도로만 사용하고, 정상적인 운영환경에서는 사용하지 않는 것을 권장한다.

**우분투 16.04 실행 ( 우분투 서버를 실행하는 것이 아니라 우분투 환경에서 명령을 실행하는 것임. )**

/bin/bash 명령어를 입력해서 ubuntu:16.04 컨테이너를 실행.

bash 쉘을 실행 키보드 입력을 위해 -it 옵션을 붙임

--rm : 프로세스가 종료되면 컨테이너가 자동으로 삭제되도록 함.

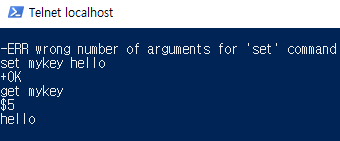
exit로 쉘을 종료하면 컨테이너도 같이 종료됨.

Redis 실행



백그라운드로 실행 후(-d), 포트 연결(-p)

telnet localhost 1234로 접속



Redis에 접속하여 키 설정 후 키 출력.

로컬호스트의 포트만 다르게 하면 하나의 서버에 여러 개의 redis 서버를 띄울 수 있음.

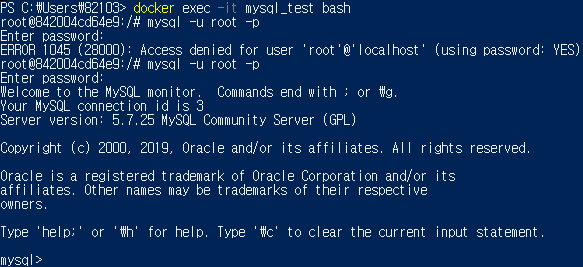
**MYSQL 5.7 container 실행**

1 docker exec -i -t mysql\_test bash

쉘을 사용하려면 exec 명령어를 사용해야 합니다. ex) $ docker exec -i -t wordpressdb /bin/bash

위 명령어를 입력하여 mysql 컨테이너에 접속한다.

mysql컨테이너의 bash쉘에 접속이 된다.



**WordPress container**

바로 전에 생성했던 MySQL 컨테이너에 워드프레스 데이터베이스를 만들고 WordPress 컨테이너를 실행할 때 --link 옵션을 이용하여 MySQL 컨테이너를 연결하겠습니다.

Docker로 이미지를 생성할 때, 하나의 이미지에 웹 서버, 데이터베이스 등 필요한 프로그램을 모두 설치하는 것도 가능하지만,

보통은 프로그램별로 이미지를 따로 생성하는 경우가 많습니다.

이렇게 이미지를 따로 생성하면 나중에 웹 서버를 교체하거나 데이터베이스를 교체하는 일이 있을 때 각 서버간 충격을 완화시키고

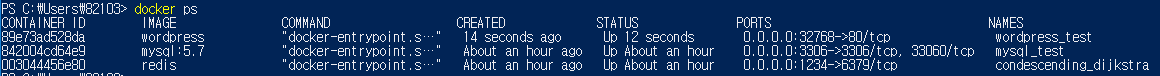
훨씬 더 유연하게 대처할 수 있기 때문입니다.

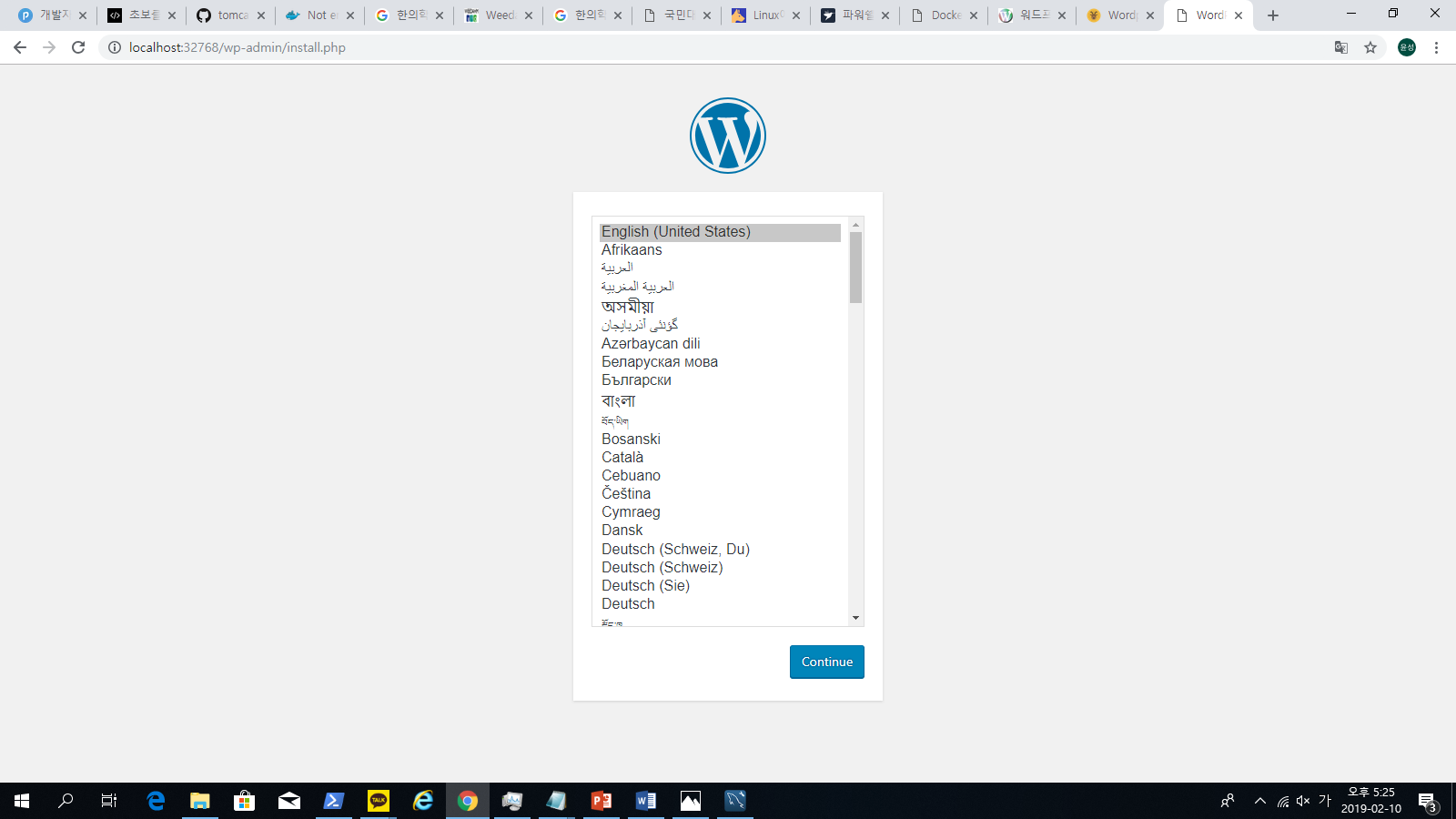
Docker 컨테이너끼리 연결할 때는 docker run 명령어에 --link 옵션을 사용합니다.

$ docker --link [컨테이너 이름]:[별명]

-p 80 80번포트 는 well-kown 포트로 HTTP이다. HTTP에 연결.



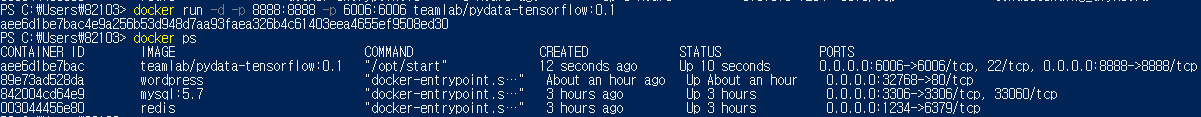


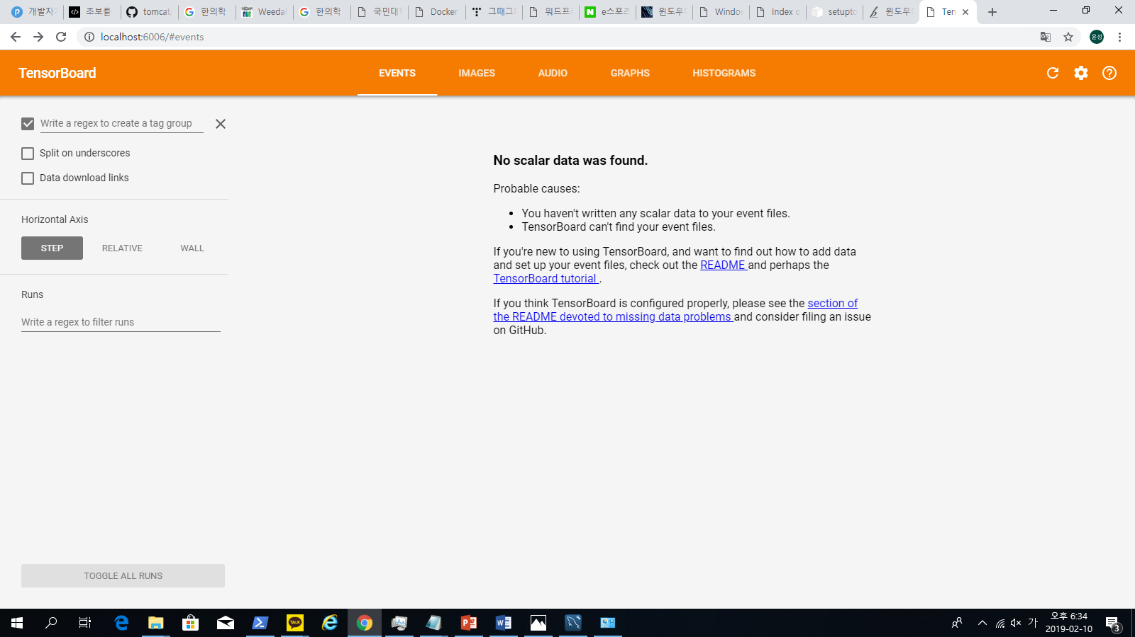


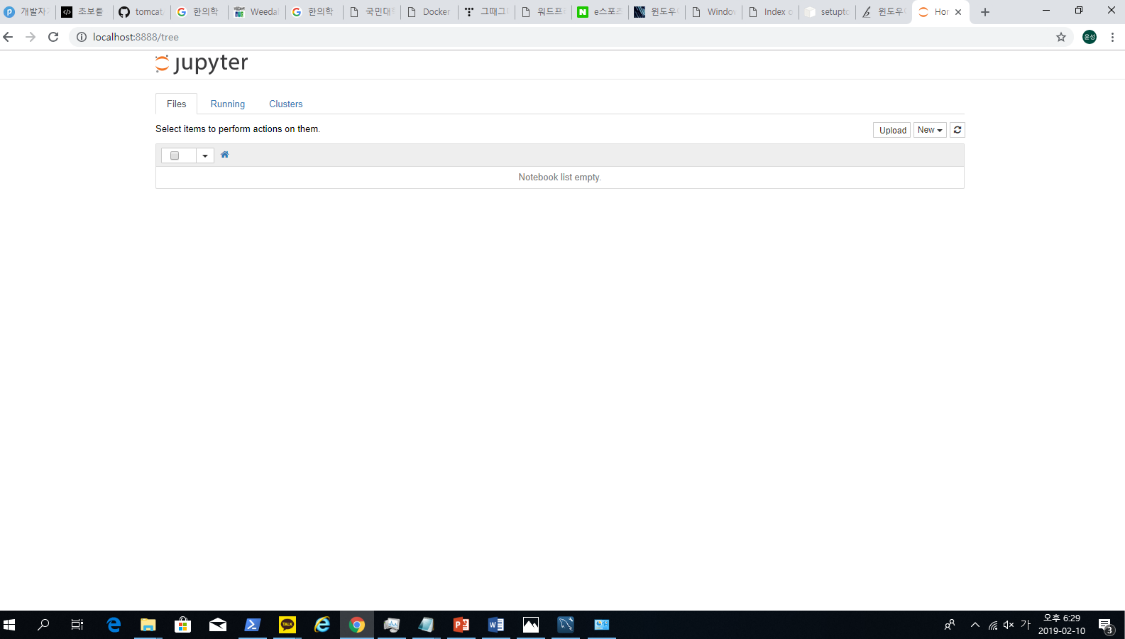
**tensorflow**

tensorflow는 손쉽게 머신러닝을 할 수 있는 툴입니다. tensorflow는 python으로 만들어져 python과 관련 패키지를 설치해야 합니다.

도커라면 손쉽게 실행해 볼 수 있습니다.







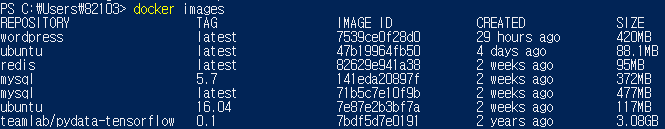
**실행중인 컨테이너 중지.**

도커 ID의 전체 길이는 64자리 입니다. 하지만 명령어의 인자로 전달할 때는 전부 입력하지 않아도 됩니다.

예를 들어 ID가 d9ece...라면 d9ec만 입력해도 됩니다. 앞부분이 다른 컨테이너의 ID와 겹치지 않는다면 1-2자만 입력해도 됩니다.

**지금까지 다운받은 이미지들 확인**

$ docker images

****

지금까지 다운받은 ubuntu, mysql, redis, wordpress, tensorflow 이미지들을 확인할 수 있다.

**이미지 삭제**

$ docker rmi [OPTIONS] image



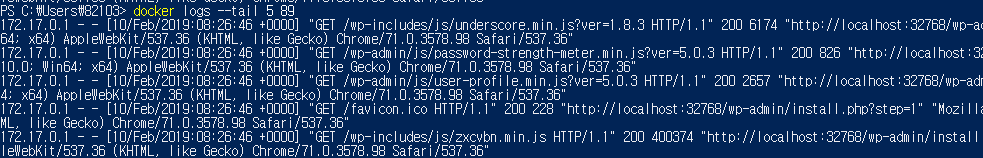
**컨테이너 로그 보기 (logs)**

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

$ docker logs [OPTIONS] CONTAINER

옵션엔 : --tail string(줄수) *- 로그 마지막부터 줄 수 출력*

-f *– 실시간으로 로그 체크.*



**Docker 데이터 볼륨 사용하기**

Docker 데이터 볼륨은 데이터를 컨테이너가 아닌 호스트에 저장하는 방식입니다.

컨테이너를 삭제한다는 건 컨테이너에서 생성된 파일이 사라진다는 뜻입니다.

데이터베이스라면 그동안 쌓였던 데이터가 모두 사라진다는 것이고 웹 어플리케이션이라면

그동안 사용자가 업로드한 이미지가 모두 사라진다는 것입니다.

그걸 방지하기 위해 컨테이너에서 생성된 데이터를 호스트에 저장합니다.

데이터 볼륨을 이용하면 컨테이너가 삭제되어도 호스트에 저장된 데이터가 사라지지 않습니다.

-v <컨테이너 디렉터리>

이 방법으로 컨테이너 여러 개가 데이터를 공유할 수도 있습니다.

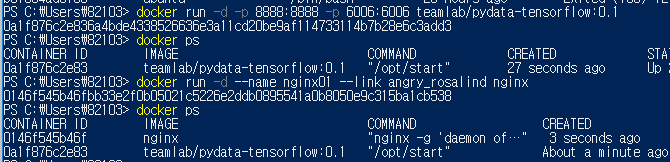
**Docker Compose**

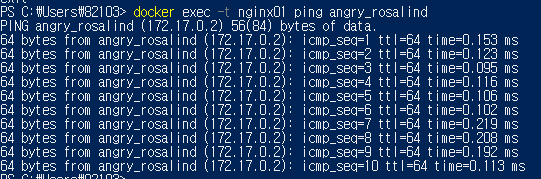
지금까지 도커를 커맨드라인에서 명령어로 작업했습니다.

지금은 간단한 작업만 했기 때문에 명령이 길지 않지만 컨테이너 조합이 많아지고 여러가지 설정이 추가되면 명령어가 금방 복잡해집니다.

Docker-compose.yml 파일에 설정을 입력해 $docker-compose up 명령을 실행하면 손쉽게 컨테이너를 생성할 수 있습니다.

**컨테이너 링크 테스트**

****

****

Nginx 컨테이너에 tensorflow 컨테이너를 링크 시킨 후 ping pong으로 둘 사이의 연결을 보여주는 결과입니다.