Docker CLI

docker images, container를 다루기 위한 Command Line Interface를 알아봅니다.

image

먼저, docker container를 만들기 위한 image에 대하여 알아야 합니다.
이미지는 레지스트리 계정, 레포지토리 이름, 태그 세 가지 정보로 구성되어 있습니다.

Registry

레지스트리는 도커 이미지를 관리하는 공간입니다.
종류로는 Docker Hub, Private Docker Hub, 회사 내부용 레지스트리 등으로 나뉠 수 있습니다.

특별히 다른 것을 지정하지 않는다면, 도커 허브(Docker Hub)를 기본으로 설정합니다.
Docker Hub : https://hub.docker.com/

Repository

레지스트리 내에 도커 이미지가 저장되는 공간으로 GitHub와 유사하게 생각하시면 됩니다.
이미지 이름으로 사용되기도 합니다.

Tag

해당 이미지를 설명하는 버전 정보를 주로 입력합니다.
지정하지 않는다면 `latest` 태그를 붙인 이미지를 가져옵니다.

CLI - 기본 명령어

docker

image

이미지에 대하여 명령합니다.

```
docker image pull
docker pull
docker image ls
docker images
docker image rm
docker rmi
```

container

컨테이너에 대하여 명령합니다.

```
docker container run
docker run
docker container ls
docker ps
docker container rm
docker rm
```

pull

이미지를 받아옵니다.

```
docker image pull docker/whalesay:latest
docker pull <repository/image:tag>
```

run

컨테이너를 실행합니다.

- Start the tutorial

```
docker run [OPTIONS] image [COMMAND] [ARG...]
docker container run --name <container_name> docker/whalesay:latest cowsay boo
```

--rm

하나의 이미지를 받아와 컨테이너로 실행하고, 컨테이너와 관련된 리소스를 삭제하는 작업을 한 번에 해보기.
`--rm` : 컨테이너를 일회성으로 실행합니다.

```
docker container run --name <NAMES> --rm docker/whalesay cowsay boo
```

--it

입력을 허용하고 터미널에 출력을 사용하는 옵션입니다.

```
docker container run -it --rm danielkraic/asciiquarium:latest
```

-d

컨테이너를 백그라운드에서 실행합니다.

```
docker run -d httpd:latest
```

rm

컨테이너를 종료합니다.

```
docker image rm
docker rmi
docker container rm <NAMES>
docker rm <CONTAINER_ID>
```

```
# 컨테이너가 너무 많을 때 (주의 : 종료된 컨테이너는 복구할 수 없습니다.)
docker rm -f $(docker ps -aq)
```

rmi

이미지를 삭제합니다.

```
docker image rm
docker rmi
```

ps

실행중인 컨테이너의 리스트를 출력합니다.

```
docker container ps
docker ps -a
```

`-a` : 중지된 모든 컨테이너를 포함합니다.

CLI - Copy, Dockerfile

로컬에 있는 파일과 도커 이미지를 연결하는 방법

docker

container

cp

실행중인 컨테이너에 로컬의 파일을 복사해 이미지에 넣습니다.

```
docker container cp ./ <NAMES>:/usr/local/apache2/htdocs/
```

commit

실행중인 컨테이너를 이미지로 만듭니다.

```
docker container commit <NAMES> my_pacman:1.0
```

Dockerfile

Dockerfile은 이미지 파일의 설명서라고 생각하면 좋습니다.

```
# Dockerfile
FROM httpd:2.4 # 베이스 이미지를 httpd:2.4로 사용합니다.
COPY ./ /usr/local/apache2/htdocs/ # 호스트의 현재 경로에 있는 파일을 /usr/~/에 포함시킵니다.
```

build

Dockerfile로 도커 이미지 파일을 생성합니다.

```
docker build --tag my_pacman:2.0 .
# `.`을 명시 했듯이 Dockerfile이 위치한 경로를 지정해야 됩니다.
```

컨테이너 내부 bash shell

docker

exec

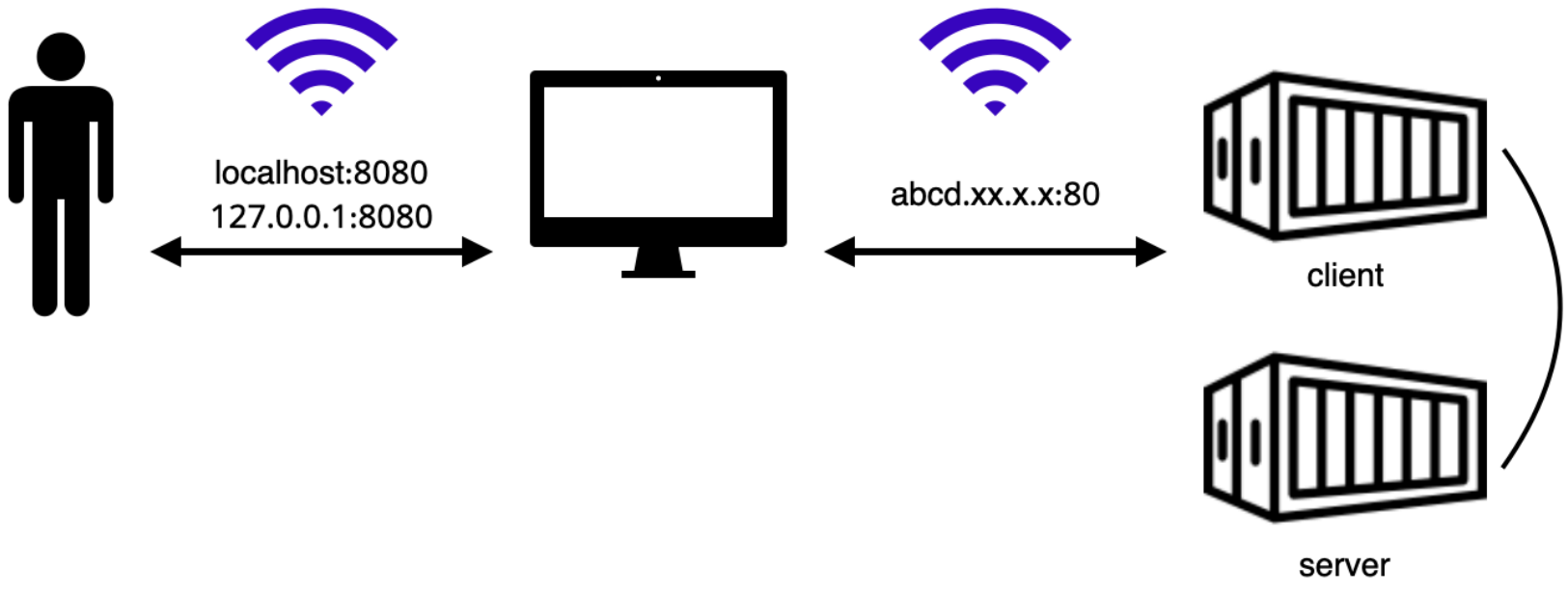
컨테이너 안에서 bash shell을 실행할 수 있습니다.

```
$ docker exec -it <NAMES> bash
root@538de4e5e997:/usr/local/apache2#
$ root@538de4e5e997:/usr/local/apache2# cd /
root@538de4e5e997:/#
$ root@538de4e5e997:/# ls
bin   data  etc   lib    mnt  proc  run   srv  tmp  var
boot  dev   home  media  opt  root  sbin  sys  usr
$ root@538de4e5e997:/# cd data
$ root@538de4e5e997:/data# ls
quiz2.txt
$ root@538de4e5e997:/data# apt update
$ root@538de4e5e997:/data# apt install nano
$ root@538de4e5e997:/data# nano quiz2.txt
"정답은 이겁니다."
```

두 개의 Docker images 다루기

Docker compose의 작동 방식을 이해하고 사용합니다.

nginx 이미지를 기반으로 한 sebcontents/client 이미지를 이용하여 client 컨테이너를 생성합니다.
node 이미지를 기반으로 한 sebcontents/server 이미지를 이용하여 server 컨테이너를 생성합니다.



localhost 의 8080 번 포트로 접속하면, sebcontents/client 이미지를 이용해 배포한 client 컨테이너의 80 번 포트로 연결됩니다.

위 그림은 매우 간략하게 표현되어 있습니다. 도커 네트워크에 대해서 좀 더 궁금하시다면 docker network, docker bridge 를 검색해 더 많은 정보를 찾을 수 있습니다.

docker-compose

- docker compose-file Docs

up

docker-compose.yaml에 정의된 이미지를 컨테이너로 실행합니다.

```
docker-compose up
```

`-d` 를 통하여 백그라운드(`d` =detached)에서 실행할 수 있습니다.

down

docker-compose.yaml에 정의된 이미지를 이용해 실행된 컨테이너를 종료합니다.

```
docker-compose down
```

up {image}

```
docker-compose up {image_name:tag}
```