Docker 4,5

4-1.Bash

Docker가 리눅스 기반이기 때문에 이미지를 생성할 때 Bash(Bourne-again shell)를 주로 사용합니다.

>

출력 리다이렉션. 명령 실행의 표준 출력(stdout)을 파일로 저장합니다. 유닉스계열 운영체제는 장치도 파일로 처리하기 때문에 명령 실행 결과를 특정 장치로 보낼 수도 있습니다.

```
$ echo "hello" > ./hello.txt
$ echo "hello" > /dev/null
```

<

입력 리다이렉션. 파일의 내용을 읽어 명령의 표준 입력(stdin)으로 사용합니다.

```
$ cat < ./hello.txt
```

>>

명령 실행의 표준 출력(stdout)을 파일에 추가합니다. >는 이미 있는 파일에 내용을 덮어쓰지만 >>는 파일 뒷부분에 내용을 추가합니다.

```
$ echo "world" >> ./hello.txt
```

2>

명령 실행의 표준 에러(stderr)를 파일로 저장합니다.

2>>

명령 실행의 표준 에러(stderr)를 파일에 추가합니다.

&>

표준 출력과 표준 에러를 모두 파일로 저장합니다.

1>&2

표준 출력을 표준 에러로 보냅니다. `echo` 명령으로 문자열을 표준 출력으로 출력했지만 표준 에러로 보냈기 때문에 변수에는 문자열이 들어가지 않습니다.

```
$ hello=$(echo "Hello World" 1>&2)
$ echo $hello
```

2>&1

표준 에러를 표준 출력으로 보냅니다. abcd라는 명령은 없으므로 에러가 발생하지만 에러를 표준 출력으로 보낸 뒤 다시 /dev/null로 보냈기 때문에 아무것도 출력되지 않습니다.

```
$ abcd > /dev/null 2>&1
```

|

파이프. 명령 실행의 표준 출력을 다른 명령의 표준 입력으로 보냅니다. 즉 첫 번째 명령의 출력 값을 두 번째 명령에서 처리합니다.

```
$ ls -al | grep .txt
```

$

Bash의 변수입니다. 값을 저장할 때는 $를 붙이지 않고, 변수를 가져다 쓸 때만 $를 붙입니다.

```
$ hello="Hello World"
$ echo $hello
Hello World
```

$()

명령 실행 결과를 변수화합니다. 명령 실행 결과를 변수에 저장하거나 다른 명령의 매개 변수로 넘겨줄 때 사용합니다. 또는 문자열안에 명령의 실행 결과를 넣을 때 사용합니다.

```
$ sudo docker rm $(docker ps -aq)
$ echo $(date)
Tue Sep 9 21:24:30 KST 2014
```

``

$()과 마찬가지로 명령 실행 결과를 변수화합니다.

```
$ sudo docker rm `docker ps -aq`
$ echo `date`
Tue Sep 9 21:24:30 KST 2014
```

&&

한 줄에서 명령을 여러 개 실행합니다. 단 앞에 있는 명령이 에러 없이 실행되어야 뒤에 오는 명령이 실행됩니다.

```
$ make && make install
```

;

한 줄에서 명령을 여러 개 실행합니다. 앞에 있는 명령이 실패를 해도 뒤에 오는 명령이 실행됩니다.

```
$ false; echo "Hello"
Hello
```

4-1.Bash

' '

문자열입니다. ' '안에 들어있는 변수는 처리되지 않고 변수명 그대로 사용됩니다. 또한 ` `와 $()도 처리되지 않고 그대로 사용됩니다.

```
$ echo '$USER'
$USER
```

$USER가 그대로 출력됩니다.

" "

문자열입니다. 명령에 문자열 매개변수를 입력하거나 변수에 저장할 때 주로 사용합니다. ' '와는 달리 " "안에 변수가 들어있으면 변수의 내용으로 바뀝니다. 또한 ` `와 $()도 실행 결과 값이 사용됩니다.

```
$ echo "Hello World"
Hello World
$ echo "$USER"
pyrasis
$ echo "Host name is $(hostname)"
Host name is ubuntu
$ echo "Time: `date`"
Time: Tue Sep  9 21:28:10 KST 2014
```

" ' ' "

" "안에 ' '가 들어갈 수 있습니다. 명령 안에서 다시 명령을 실행하고 매개 변수를 지정할 때 사용합니다.

```
$ bash -c "/bin/echo Hello 'World'"
Hello World
```

₩" ₩$hello

' '안에서 "를 사용할 때는 ₩"처럼 앞에 ₩를 붙여쥽니다.

```
$ bash -c "/bin/echo '{ \"user\": \"$USER\" }'"
{ "user": "pyrasis" }
```

" "안에서 ", $, ` 등의 특수문자를 그대로 사용하려면 앞에 ₩를 붙여쥽니다.

```
$ echo "\$hello \" \`"
$hello " `
```

${}

변수 치환(substitution)입니다. " " 문자열 안에서 변수를 출력할 때 주로 사용합니다. ${} 대신 $만 사용해도 됩니다.

```
$ str="World"
$ echo "Hello ${str}"
Hello World
```

스크립트에서 변수의 기본 값을 설정할 때도 사용합니다. 다음은 HELLO 변수가 있으면 그대로 사용하고 변수가 없으면 기본 값으로 설정한 abcd를 대입합니다.

```
$ HELLO=
$ HELLO=${HELLO-"abcd"}
$ echo $HELLO
```

값이 NULL인 HELLO 변수가 이미 있기 때문에 기본 값을 대입하지 않습니다. 다음은 변수에 값이 있으면 그대로 사용하고, 값이 NULL이면 기본 값으로 설정한 abcd를 대입합니다.

```
$ WORLD=
$ WORLD=${WORLD:-"abcd"}
$ echo $WORLD
abcd
```

변수에 값이 NULL이므로 기본 값을 대입합니다.

₩

한 줄로된 명령을 여러 줄로 표현할 때 사용합니다.

```
$ sudo docker run -d --name hello busybox:latest
$ sudo docker run \
    -d \
    --name hello \
    busybox:latest
```

{1..10}

연속된 숫자를 표현합니다. {시작 숫자..끝 숫자} 형식입니다.

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
```

{문자열1,문자열2}

{}안에 문자열을 여러 개 지정하여 명령 실행 횟수를 줄입니다. 다음은 hello.txt, world.txt 두 파일을 한번에 hello-dir 디렉터리 아래에 복사합니다.

```
$ cp ./{hello.txt,world.txt} hello-dir/
```

4-1.Bash

if

if 조건문입니다. 변수와 변수끼리 또는 문자열과 비교할 때 사용합니다.

```
if [ $a -eq $b ]; then
  echo $a
fi
```

숫자 비교

- -eq: 같다
- -ne: 같지 않다
- -gt: 초과
- -ge: 이상
- -lt: 미만
- -le: 이하

문자열 비교

- =, ==: 같다
- !=: 같지 않다
- -z: 문자열이 NULL일 때
- -n: 문자열이 NULL이 아닐 때

for

for 반복문입니다. 변수안에 있는 값을 반복하거나 범위를 지정하여 반복할 수 있습니다.

```
for i in $(ls)
do
  echo $i
done

for (( i=0; i < 10; i++ ))
do
  echo $i
done

NUM=(1 2 3)
for i in ${NUM[@]}
do
  echo $i
done
```

while

while 반복문입니다.

```
while :
do
  echo "Hello World";
  sleep 1;
done
```

<<<

문자열을 명령(프로세스)의 표준 입력으로 보냅니다.

```
$ cat <<< "User name is $USER"
User name is pyrasis
```

<<EOF
EOF

여러 줄의 문자열을 명령(프로세스)의 표준 입력으로 보냅니다.

```
cat > ./hello.txt <<EOF
Hello World
Host name is $(hostname)
User name is $(USER)
EOF
```

cat은 파일이나 표준 입력의 내용을 출력하는 명령입니다. cat의 표준 출력을 ./hello.txt로 저장하고, <<EOF로 문자열을 cat의 표준 입력으로 보냅니다. 이렇게 하면 문자열 3줄이 ./hello.txt 파일에 저장됩니다.

export

설정한 값을 환경 변수로 만듭니다. `export <변수>=<값>` 형식입니다.

```
$ export HELLO=world
```

printf

지정한 형식대로 값을 출력합니다. 파이프와 연동하여 명령(프로세스)에 값을 입력하는 효과를 낼 수 있습니다.

```
$ printf 80\\nexampleuser\\ny | example-config
Port: 80
User: exampleuser
Save Configuration (y/n): y
```

예를 들어 `example-config`는 Port, User, Save Configuration을 사용자에게 입력을 받습니다. `printf`로 미리 값을 설정하여 파이프로 `example-config`에 넘겨주면 사용자가 입력하지 않아도 자동으로 값이 입력됩니다. 줄바꿈(개행)은 ₩₩n으로 표현합니다.

4-1.Bash

sed	텍스트 파일에서 문자열을 변경합니다. hello.txt 파일의 내용 중에서 hello라는 문자열을 찾아서 world 문자열로 바꾸려면 다음과 같이 실행합니다. `$ sed -i "s/hello/world/g" hello.txt` `sed -i "s/<찾을 문자열>/<바꿀 문자열>/g" <파일명>` 형식입니다. /와 같은 특수 문자는 앞에 ₩를 붙여 ₩/로 입력합니다.
#	주석입니다. 스크립트에 설명을 추가하거나, 명령이 실행되지 않도록 합니다. `# echo "Hello World"`

4-2.Docker file

Dockerfile은 Docker 이미지 설정 파일입니다.
Dockerfile에 설정된 내용대로 이미지를 생성합니다.

example/Dockerfile

```
FROM ubuntu:14.04
MAINTAINER Foo Bar <foo@bar.com>

RUN apt-get update
RUN apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf
RUN chown -R www-data:www-data /var/lib/nginx

VOLUME ["/data", "/etc/nginx/site-enabled", "/var/log/nginx"]

WORKDIR /etc/nginx

CMD ["nginx"]

EXPOSE 80
EXPOSE 443
```

우분투 14.04를 기반으로 nginx 서버를 설치한 Docker 이미지를 생성

- FROM: 어떤 이미지를 기반으로 할지 설정합니다. Docker 이미지는 기존에 만들어진 이미지를 기반으로 생성합니다. `<이미지 이름>:<태그>` 형식으로 설정합니다.
- MAINTAINER: 메인테이너 정보입니다.
- RUN: 셸 스크립트 혹은 명령을 실행합니다.
  - 이미지 생성 중에는 사용자 입력을 받을 수 없으므로 apt-get install 명령에서 `-y` 옵션을 사용합니다(yum install도 동일).
  - 나머지는 nginx 설정입니다.
  - VOLUME: 호스트와 공유할 디렉터리 목록입니다. `docker run` 명령에서 `-v` 옵션으로 설정할 수 있습니다. 예) `-v /root/data:/data`는 호스트의 **/root/data** 디렉터리를 Docker 컨테이너의 **/data** 디렉터리에 연결합니다.
- CMD: 컨테이너가 시작되었을 때 실행할 실행 파일 또는 셸 스크립트입니다.
- WORKDIR: CMD에서 설정한 실행 파일이 실행될 디렉터리입니다.
- EXPOSE: 호스트와 연결할 포트 번호입니다.

4-3.build 명령

Dockerfile을 작성하였으면 이미지를 생성합니다.

```
~/example$ sudo docker build --tag hello:0.1 .
```

`docker build <옵션> <Dockerfile 경로>` 형식입니다.
`--tag` 옵션으로 이미지 이름과 태그를 설정할 수 있습니다. 이미지 이름만 설정하면 태그는 **latest**로 설정됩니다.

그 후에 실행을 합니다.

```
$ sudo docker run --name hello-nginx -d -p 80:80 -v /root/data:/data hello:0.1
```

- `-d` 옵션은 컨테이너를 백그라운드로 실행합니다.
- `-p 80:80` 옵션으로 **호스트의 80번 포트**와 **컨테이너의 80번 포트**를 연결하고 외부에 노출합니다. 이렇게 설정한 뒤 **http://<호스트 IP>:80**에 접속하면 컨테이너의 **80**번 포트로 접속됩니다.
- `-v /root/data:/data`옵션으로 호스트의 **/root/data** 디렉터리를 컨테이너의 **/data** 디렉터리에 연결합니다. **/root/data** 디렉터리에 파일을 넣으면 컨테이너에서 해당 파일을 읽을 수 있습니다.

웹 브라우저를 실행하고, **http://<호스트 IP>:80**으로 접속합니다. **Welcome to nginx!** 페이지가 표시됩니다.

5-1.history 명령

앞에서 생성한 hello:0.1 이미지의 히스토리를 조회

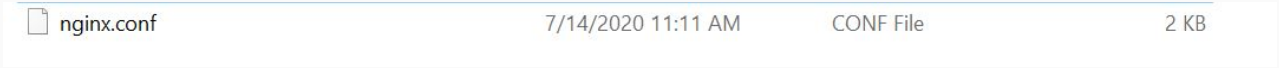
```
C:₩Users₩jpark₩example>docker history hello:0.1
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
fed5410ab7ca        17 minutes ago      /bin/sh -c #(nop)  EXPOSE 443                   0B
95077809b057        17 minutes ago      /bin/sh -c #(nop)  EXPOSE 80                    0B
32a2bba75ee9        17 minutes ago      /bin/sh -c #(nop)  CMD ["nginx"]                0B
f4bc5aafc265        17 minutes ago      /bin/sh -c #(nop) WORKDIR /etc/nginx            0B
5d585092099f        17 minutes ago      /bin/sh -c #(nop)  VOLUME [/data /etc/nginx/…   0B
14a33a704191        17 minutes ago      /bin/sh -c chown -R www-data:www-data /var/l…   0B
0bd1e6c486ab        17 minutes ago      /bin/sh -c echo "₩ndaemon off;" >> /etc/ngin…   1.61kB
a9c8cfec9ce5        17 minutes ago      /bin/sh -c apt-get install -y nginx             21.1MB
fa67699f4a79        18 minutes ago      /bin/sh -c apt-get update                       13.8MB
5c0ab5417648        18 minutes ago      /bin/sh -c #(nop)  MAINTAINER Foo Bar <foo@b…   0B
6e4f1fe62ff1        6 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           6 months ago        /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
<missing>           6 months ago        /bin/sh -c set -xe   && echo '#!/bin/sh' > /…   195kB
<missing>           6 months ago        /bin/sh -c [ -z "$(apt-get indextargets)" ]      0B
<missing>           6 months ago        /bin/sh -c #(nop) ADD file:276b5d943a4d284f8…   196MB
```

`docker history <이미지 이름>:<태그>` 형식입니다.
이미지 이름 대신 이미지 ID를 사용해도 됩니다.

5-2.cp 명령

```
$ sudo docker cp hello-nginx:/etc/nginx/nginx.conf ./
```

`docker cp <컨테이너 이름>:<경로> <호스트 경로>` 형식입니다.



5-3.commit 명령

`docker commit` 명령은 컨테이너의 변경 사항을 이미지 파일로 생성합니다.

```
C:₩Users₩jpark₩example>docker commit -a "Foo Bar <foo@bar.com>" -m "add hello.txt" hello-nginx hello:0.2
sha256:8966e550db46a49e857fea7f01f0fab2e1b2f2b8977fe06701e9dbf25a7f29ef
```

`docker commit <옵션> <컨테이너 이름> <이미지 이름>:<태그>` 형식입니다. 컨테이너 이름 대신 컨테이너 ID를 사용해도 됩니다.

`-a "Foo Bar <foo@bar.com>"`와 `-m "add hello.txt"`옵션으로 커밋한 사용자와 로그 메시지를 설정합니다. **hello-nginx** 컨테이너를 **hello:0.2** 이미지로 생성

```
C:₩Users₩jpark₩example>docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
hello                           0.2                 8966e550db46        3 minutes ago       231MB
hello                           0.1                 fed5410ab7ca        2 hours ago         231MB
jiseong4577/docker101tutorial   latest              91c9e9c83da6        22 hours ago        26.8MB
docker101tutorial               latest              91c9e9c83da6        22 hours ago        26.8MB
```

Tag 가 동일하다면 덮어쓰기

5-4.diff 명령

`docker diff` 명령은 컨테이너가 실행되면서 변경된 파일 목록을 출력합니다. 비교 기준은 컨테이너를 생성한 이미지 내용입니다.

```
C:₩Users₩jpark₩example>docker diff hello-nginx
A /data
C /etc
C /etc/nginx
A /etc/nginx/site-enabled
C /run
A /run/nginx.pid
C /var
C /var/lib
C /var/lib/nginx
A /var/lib/nginx/body
A /var/lib/nginx/fastcgi
A /var/lib/nginx/proxy
A /var/lib/nginx/scgi
A /var/lib/nginx/uwsgi
```

`docker diff <컨테이너 이름>` 형식입니다. 컨테이너 이름 대신 컨테이너 ID를 사용해도 됩니다.

`A`는 추가된 파일, `C`는 변경된 파일, `D`는 삭제된 파일입니다.

5-5.inspect 명령

`docker inspect`명령은 이미지와 컨테이너의 세부 정보를 출력합니다.

`docker inspect <이미지 또는 컨테이너 이름>` 형식입니다. 이미지, 컨테이너 이름 대신 이미지 ID나, 컨테이너 ID를 사용해도 됩니다.