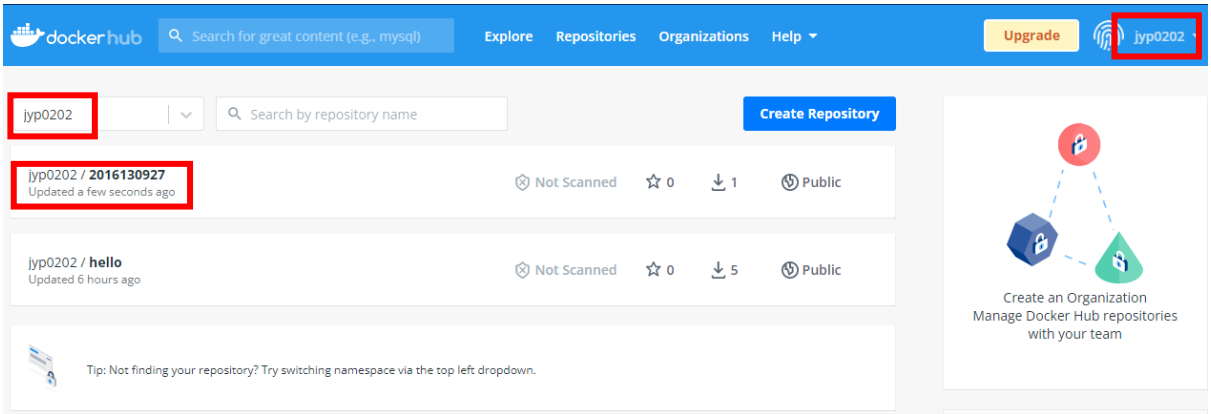
클라우드 컴퓨팅 과제 #2

: Docker 실습

2016130927 박준영

1. Docker Hub 이미지 업로드



Username: jyp0202, Repository: 2016130927

Docker Hub에 이미지를 업로드하는 과정은 다음과 같다.

```
[ec2-user@ip-172-31-84-178 sample1]$ sudo docker image tag hello:0.2 2016130927:
0.1
[ec2-user@ip-172-31-84-178 sample1]$ sudo docker image ls
REPOSITORY           TAG        IMAGE ID       CREATED          SIZE
jyp0202/hello        0.2        ebcb86aaab43   5 hours ago      75.8MB
web                  0.1        ebcb86aaab43   5 hours ago      75.8MB
2016130927           0.1        1b5129eb0afe   5 hours ago      811MB
hello                0.2        1b5129eb0afe   5 hours ago      811MB
hello                0.1        c53870b70767   6 hours ago      811MB
parkjunyeong/hello   0.2        c53870b70767   6 hours ago      811MB
jyp0202/hello        <none>     c53870b70767   6 hours ago      811MB
mongo                4          58a4d0f3f3d7   3 weeks ago      437MB
golang               1.14       21a5635903d6   9 months ago     811MB
node                 8-alpine   2b8fcdc6230a   22 months ago    73.5MB
```

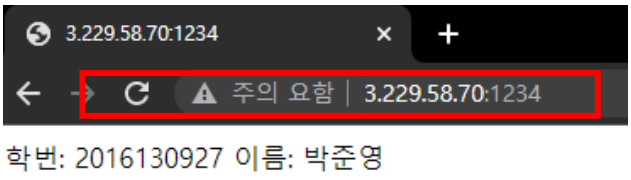
```
[ec2-user@ip-172-31-84-178 sample1]$ sudo docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

먼저 이미지 이름이 학번(2016130927)인 파일을 생성하였다(기존의 hello:0.1 이미지 파일명을 image tag 명령어를 활용, 학번으로 변경 후 업로드하는데 사용했다). 이후 Docker Hub에 접근하기 위해 로그인하였다. 이미지 파일 업로드를 위해 Username:

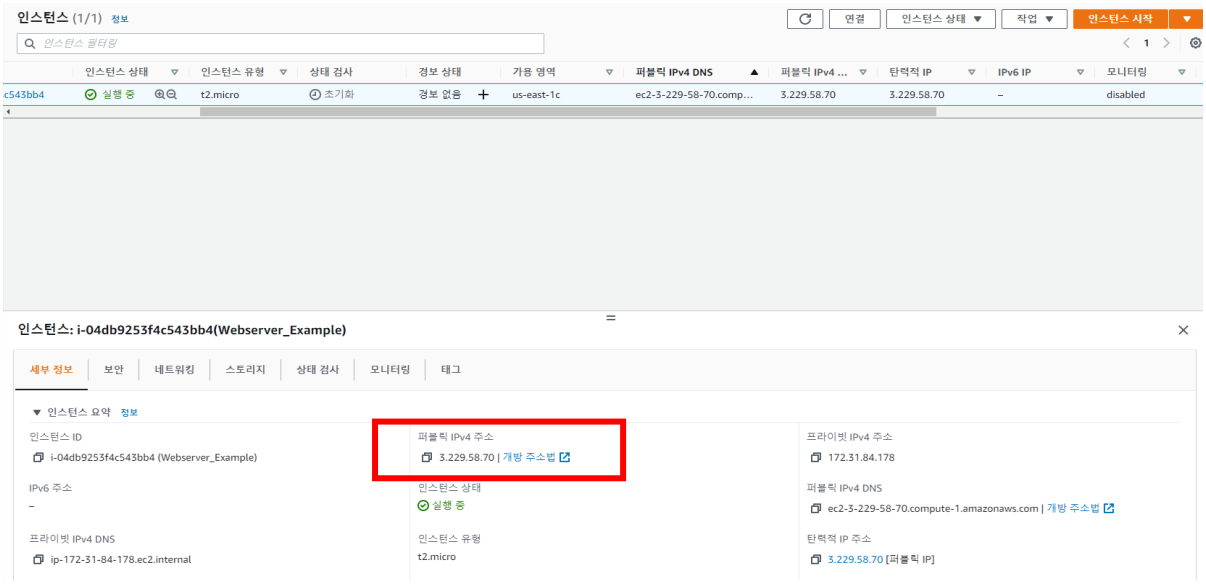
jyp0202로 이 이미지 파일을 build하였고, 이후 pull하여 Docker Hub에 업로드하였다.

```
[ec2-user@ip-172-31-84-178 sample1]$ sudo docker build --tag jyp0202/2016130927:0.1 .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM golang:1.14
 ---> 21a5635903d6
Step 2/3 : WORKDIR /go/src/app
 ---> Using cache
 ---> 2813c36b0c9e
Step 3/3 : COPY . .
 ---> Using cache
 ---> c53870b70767
Successfully built c53870b70767
Successfully tagged jyp0202/2016130927:0.1
[ec2-user@ip-172-31-84-178 sample1]$ sudo docker push jyp0202/2016130927:0.1
The push refers to repository [docker.io/jyp0202/2016130927]
e64446lc513b: Mounted from jyp0202/hello
7cc0e5b9633a: Mounted from jyp0202/hello
660b50dfadeb: Mounted from jyp0202/hello
3eeebffacaf0: Mounted from jyp0202/hello
041459108fb0: Mounted from jyp0202/hello
da654bc8bc80: Mounted from jyp0202/hello
4ef81dc52d99: Mounted from jyp0202/hello
909e93c71745: Mounted from jyp0202/hello
7f03bfe4d6dc: Mounted from jyp0202/hello
0.1: digest: sha256:3a935fa984b075ce68d2cc5d8277ebf47a07c69ca8222ad340be4cdc130b7272 size: 2209
```

2. 웹 서버 컨테이너 실행 및 브라우저 출력 결과



퍼블릭 주소에 포트 번호 1234를 더한 주소를 검색한 결과이다.



인스턴스: i-04db9253f4c543bb4(Webserver_Example)

sg-035325455e8eec81b (Webserver)

▼ 인바운드 규칙

필터 규칙

보안 그룹 규칙 ID	포트 범위	프로토콜	원본	보안 그룹
sgr-064f420617330b232	80	TCP	0.0.0.0/0	Webserver
sgr-0f96be3861522f9be	80	TCP	::/0	Webserver
sgr-05a99f8bf172a3168	22	TCP	0.0.0.0/0	Webserver
sgr-03b57e0c3f5184a9b	1234	TCP	::/0	Webserver
sgr-02042e1a314c891e8	1234	TCP	0.0.0.0/0	Webserver

EC2 인스턴스의 보안 그룹 규칙으로 포트 번호 1234를 추가해 '퍼블릭 주소 +:1234'로 접근 가능하도록 설정하였다.

```
const express = require('express')
const app = express()
const port = 1234

app.get('/', (req, res) => {
        res.send("학 번 : 2016130927 이 름 : 박 준 영 ")
})

app.listen(port, () => {
        console.log("listen : "+port)
})
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"index.js" 11L, 218C
```

sample2의 index.js 파일을 수정해 웹 서버에서 학번과 이름이 출력되도록 수정하였다. 기존 파일은 포트 3000번에 expose하였으나, 포트 1234에 expose하도록 포트 번호 역시 수정하였다.

```
[ec2-user@ip-172-31-84-178 sample2]$ sudo docker run -d --rm -p 1234:1234 web:0.1
e7d9f110befa3f887acb6c4af2cbdc005057cdb5307079a8b8c2d6b105068651
[ec2-user@ip-172-31-84-178 sample2]$ sudo curl http://localhost:1234
학 번 : 2016130927 이 름 : 박 준 영 [ec2-user@ip-172-31-84-178 sample2]$
```

이후 index.js가 들어 있는 이미지 web:0.1를 도커를 통해 run했다. 이미지 파일의 expose 뿐만 아니라 호스트 포트 넘버 1234와 컨테이너 포트 넘버 1234를 -p 옵션을 추가해 포워딩해주는데, 이때 기본값은 TCP이므로 앞서 보안 그룹 규칙에서 명시한 바와 연결된다. 이렇게 로컬 호스트에 1234 포트 넘버를 덧붙여 접근 가능하도록 설정할 수 있다.

3. Volume

```
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker build --tag volume:0.1 .
Sending build context to Docker daemon  14.85kB
Step 1/3 : FROM alpine:latest
latest: Pulling from library/alpine
a0d0a0d46f8b: Pull complete
Digest: sha256:e1c082e3d3c45cccac829840a2594e679c25d438cc8412c2fa221cf1a824e6a
Status: Downloaded newer image for alpine:latest
 ---> 14119a10abf4
Step 2/3 : WORKDIR /volume
 ---> Running in 737a84941c84
Removing intermediate container 737a84941c84
 ---> efc2f82c46e2
Step 3/3 : COPY . .
 ---> c8590e1ce096
Successfully built c8590e1ce096
Successfully tagged volume:0.1
```

Sample3 파일의 Dockerfile을 이용해 volume:0.1 이미지 파일을 build하였다. 이미지 파일을 확인해본 결과는 다음과 같다.

```
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker images
REPOSITORY          TAG                 IMAGE ID       CREATED          SIZE
volume              0.1                 c8590e1ce096   12 seconds ago   5.61MB
web                 0.1                 c56c1623b727   20 minutes ago   75.8MB
myweb               latest              8c85a3eaecad   2 hours ago      198MB
2016130927          0.1                 1b5129eb0afe   9 hours ago      811MB
hello               0.2                 1b5129eb0afe   9 hours ago      811MB
jyp0202/2016130927  0.1                 c53870b70767   9 hours ago      811MB
hello               0.1                 c53870b70767   9 hours ago      811MB
jyp0202/hello       <none>              c53870b70767   9 hours ago      811MB
httpd               latest              1132a4fc88fa   2 weeks ago      143MB
ubuntu              18.04               5a214d77f5d7   5 weeks ago      63.1MB
alpine              latest              14119a10abf4   2 months ago     5.6MB
node                15.12.0-alpine3.12  c9343d22a2a1   7 months ago     112MB
golang              1.14                21a5635903d6   9 months ago     811MB
node                8-alpine            2b8fcdc6230a   22 months ago    73.5MB
```

이후 학번을 이름으로 한 volume을 생성하였고, 이를 도커에서 확인할 수 있었다. Inspect 명령어를 통해 volume 파일을 확인해본 결과, 이 volume 파일은 /var/lib/docker/volumes 안에 위치함을 알 수 있다.

```
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker volume create 2016130927
2016130927
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker volume ls
DRIVER    VOLUME NAME
local     2016130927
local     myvol
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker volume inspect 2016130927
[
    {
        "CreatedAt": "2021-11-10T11:02:05Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/2016130927/_data",
        "Name": "2016130927",
        "Options": {},
        "Scope": "local"
    }
]
```

```
[ec2-user@ip-172-31-84-178 sample3]$ sudo docker run -v 2016130927:/app volume:0.1 touch /app/hello.txt
[ec2-user@ip-172-31-84-178 sample3]$ sudo ls /var/lib/docker/volumes/2016130927/_data
hello.txt
```

-v 명령어를 통해 volume:0.1 이미지 파일을 컨테이너에서 run할 때 2016130927 volume을 마운트시키면서 touch 명령어를 통해 컨테이너 내 app 디렉토리에 hello.txt라는 파일을 생성하였다. 이때 -v 명령어는 콜론(:)을 기준으로 마운트할 volume의 이름과 컨테이너의 경로를 연결시킨다. 다른 것은 일반적인 컨테이너를 run하는 명령어이다.

만일 컨테이너와 volume이 정상적으로 mount되었다면, 컨테이너에서 생성한 hello.txt가 volume 내에서도 존재할 것이다. 이를 위해 ls 명령어를 통해 volume명 2016130927 내 data가 있는 주소를 검색했고, hello.txt를 확인할 수 있었다.