9장

Docker Image 생성하기

전체 내용

Docker Image
생성하기

Docker Image 생성하기

RUN

CMD, ENTRYPOINT

EXPOSE

VOLUME

WORKDIR

COPY, ADD

ENV, ARG

Docker Image 생성하기

USER

LABEL

HEALTHCHECK

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - 지원하는 2가지 형식
    - **/bin/sh** 형식(Default)
      - FROM centos:7
        RUN yum -y install httpd
    - **exec** 형식(JSON 배열)
      - FROM centos:7
        RUN ["/bin/bash", "-c", "yum -y install nginx"]
    - 이미지에 /bin/sh가 없는 경우에는 꼭 exec 형식을 사용해야 하며, exec 형식을 사용할 때는 Shell을 경유하지 않고 직접 실행되기 때문에 환경 변수 등을 사용할 수 없다
  - docker image 생성하는 방법
    - Dockerfile(또는 dockerfile)을 만들어 내용을 입력하여 이미지를 생성한다
    - **docker build -t 이미지이름 .**
      - 보통 Dockerfile이 저장된 곳에서 실행한다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **RUN**
    - 각종 리눅스 명령어(Package 설치 등)를 실행하며, 이 때 **각 RUN 당 이미지 레이어를 만들게 된다**
      - Container를 만들어 실행할 때는 사용되지 않고, 단지 이미지를 만들 때 사용한다
    - RUN 명령은 되도록 **한 줄로 작성**하는 것이 좋다
      - RUN 명령은 image layer를 생성해 나가는 과정이기 때문에, 같은 결과를 가져오더라도 RUN을 여러 줄로 작성하면 image layer가 여러 개 생성되고, RUN을 한 줄로 작성하면 image layer가 1개로 생성된다
    - 앞으로 각 실습할 때 Dockerfile을 생성하기 위해서 필요한 Directory를 /lab 하위에 생성한다
      - **mkdir -p /lab/run**
      - **cd /lab/run**
      - **vi Dockerfile**

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **RUN**
    - 한 줄 예제
      - FROM centos:7
        RUN yum -y install **httpd php mysql**
    - **docker build -t img_run_2 .**
    - **docker history img_run_2**

```
[root@docker3 ~]# docker history img_run_2
IMAGE          CREATED         CREATED BY                                      SIZE
8b79dba71a43   2 minutes ago   /bin/sh -c yum -y install httpd php mysql       329MB
eeb6ee3f44bd   14 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      14 months ago   /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>      14 months ago   /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4…   204MB
```

    - **docker image inspect img_run_2 --format='{{.Size}}'**

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **RUN**
    - 여러 줄 예제
      - FROM centos:7
        RUN yum -y install **httpd**
        RUN yum -y install **php**
        RUN yum -y install **mysql**
    - **docker build -t img_run_1 .**
    - **docker history img_run_1**

```
[root@docker3 ~]# docker history img_run_1
IMAGE          CREATED              CREATED BY                                      SIZE
f995597a4566   About a minute ago   /bin/sh -c yum -y install mysql                 279MB
13963f8fc5cf   About a minute ago   /bin/sh -c yum -y install php                   206MB
2f2bc799e3ea   2 minutes ago        /bin/sh -c yum -y install httpd                 221MB
eeb6ee3f44bd   14 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      14 months ago        /bin/sh -c #(nop)  LABEL org.label-schema.sc…   0B
<missing>      14 months ago        /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4…   204MB
```

    - **docker image inspect img_run_1 --format='{{.Size}}'**
    - **docker images**

```
[root@docker3 ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
img_run_2           latest    8b79dba71a43   23 minutes ago       533MB
img_run_1           latest    f995597a4566   24 minutes ago       909MB
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **CMD, ENTRYPOINT**
    - 두 명령 모두 Image를 바탕으로 생성된 **Container에서 사용된다**
      - **docker run** 또는 **docker start**
    - mkdir /lab/nginx ; cd /lab/nginx ; vi Dockerfile
    - centos7에 nginx를 설치하는 이미지 생성(Dockerfile)
      - **FROM centos:7**
      - **RUN** touch /etc/yum.repos.d/nginx.repo && echo -e '[nginx]\nname=nage repo\nbaseurl=http://nginx.org/packages/centos/7/$basearch/\ngpgcheck=0\nenabled=1' > /etc/yum.repos.d/nginx.repo
      - **RUN** yum install nginx -y
      - **CMD** ["nginx", "-g", "daemon off;"]
      - EXPOSE 80
    - **docker build -t nginx_centos7 .**
    - RUN 이하는 **Image를 생성**할 때 실행되는 것이며,
      CMD 이하는 **Container를 생성**할 때 실행되는 것이다
      - nginx 웹서비스는 **"docker run -d -p 80:80 nginx_centos7"**을 실행할 때 구동되는 것이다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **CMD, ENTRYPOINT**
    - CMD
      - **명령과 인자는 변경될 수 있고**, Container에서 명령 설정하지 않을 때는 CMD에 기재된 명령을 default로 실행된다
    - ENTRYPOINT
      - **기본적으로 명령 변경은 불가능하며**(사용자에 의해 변경되지 않고), 고정적으로 실행될 명령은 ENTRYPOINT를 사용하는 것이 좋다
      - 하지만 docker run의 **--entrypoint="실행명령"** 옵션을 사용하여 Dockerfile에 정의한 ENTRYPOINT를 무시할 수 있다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **CMD, ENTRYPOINT**
    - CMD 예제
      - mkdir /lab/cmd ; cd /lab/cmd
      - vi Dockerfile
        - **FROM** centos:7
        - **CMD** ["/bin/ls", "-lh", "/root"]
      - **docker build -t img_cmd_1 .**
      - **docker run -it img_cmd_1**

```
[root@docker3 cmd]# docker run -it img_cmd_1
total 4.0K
-rw------- 1 root root 3.4K Nov 13  2020 anaconda-ks.cfg
```

      - **docker run -it img_cmd_1 ls -la /tmp**
      - **docker run -it img_cmd_1 ps -ef**
      - **docker run -it img_cmd_1 bash**
      - 명령과 인자를 다르게 하여 각각 실행된 container의 결과를 확인해 본다.
        사용자가 입력한 명령과 인자의 결과가 나타난다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **CMD, ENTRYPOINT**
    - ENTRYPOINT 예제
      - mkdir /lab/entrypoint ; cd /lab/entrypoint
      - vi Dockerfile
        - **FROM** centos:7
        - **ENTRYPOINT** ["/bin/ls", "-lh", "/root"]
      - **docker build -t img_ entrypoint_1 .**
      - **docker run -it img_entrypoint_1**

```
[root@docker3 entrypoint]# docker run -it img_entrypoint_1
total 4.0K
-rw------- 1 root root 3.4K Nov 13  2020 anaconda-ks.cfg
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **CMD, ENTRYPOINT**
    - ENTRYPOINT 예제
      - **docker run -it img_entrypoint_1 cat /etc/passwd**
        - "cat /etc/passwd"의 결과는 나오지 않고, **원래 것인 "ls -lh /root"만 나온다**
      - **docker run -it img_entrypoint_1 ps -ef**
        - 오류가 나온다
      - **docker run -it img_entrypoint_1 /bin/bash**
      - **docker run -it --entrypoint=cat img_entrypoint_1 /etc/passwd**
        - 이것은 정상적으로 결과가 나온다. entrypoint로 만든 이미지일지라도 **"--entrypoint=명령어** 이미지이름 **인자"**를 사용하면 사용자가 입력한 명령이 정상적으로 실행된다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **EXPOSE**
    - 해당 Container가 런타임에 지정된 네트워크 포트에서 수신 대기중 이라는것을 알려준다
    - 보통 Dockerfile을 작성하는 사람과 Container를 직접 실행할 사람 사이에서 공개할 포트를 알려주기 위해 문서 유형으로 작성할 때 사용한다
    - 이 명령 자체가 작성된 포트를 실행하여 listening 상태로 올려주지 않기 때문에, 실제로 Port를 열기 위해선 container run 에서 -p 옵션을 사용해야 한다
    - 이미지에서 사용하는 Port 확인
      - **docker image history nginx_centos7 | grep -i expose**

```
[root@docker3 healthcheck]# docker history nginx_centos7 | grep -i expose
b591509f27f9   5 hours ago     /bin/sh -c #(nop)  EXPOSE 80
```

    - 이 Port를 확인 후에 docker run -p 8080:80을 해주게 된다
      - **docker run --name mynginxweb -d -p 8080:80 nginx_centos7**
      - **curl http://localhost:8080**

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **VOLUME**
    - Container 안에 있는 data는 Container를 삭제하면 모든 데이터가 같이 삭제(휘발성 데이터) 되기 때문에 데이터를 보존하기 위해 VOLUME을 사용
    - VOLUME 명령은 설정한 Container의 data를 호스트 OS에 저장하거나, **Container들간의 데이터를 공유**를 할 수 있다
    - Dockerfile에 사용하는 VOLUME는 docker host의 /var/lib/docker/volumes에 생성되며, Docker에서 자동 생성한 hash값으로 디렉토리가 생긴다
      - 파일을 찾기가 어렵다
    - mkdir /lab/volume ; cd /lab/volume
    - vi Dockerfile
      - FROM centos:7
      - **VOLUME** ["/var/log/", "/shared/"]
    - **docker build -t img_volume_1 .**
    - docker run --name volume1 -itd img_volume_1
    - docker run --name volume2 -itd img_volume_1

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **VOLUME**
    - **docker exec** -it volume1 bash
    - cd /shared
    - touch GodBlessYou.file
    - exit
    - **docker exec** -it volume2 bash
    - cd /shared
    - touch JesusLovesYou.file
    - exit
    - find / -name "GodBlessYou.file"
    - find / -name "JesusLovesYou.file"

두 파일의 저장 위치가 다르다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **VOLUME**
    - docker volume은 **docker run 명령을 실행할 때 사용하는 것을 권장한다**
    - 이렇게 하면 다양한 위치(docker host, nfs, 외부 storage 등)에 저장할 수도 있고, Container간의 파일 공유도 가능하다
      - **docker volume create** shared
      - docker run --name volume3 -itd **-v shared:**/data img_volume_1
      - docker run --name volume4 -itd **-v shared:**/data img_volume_1
      - **docker exec** -it volume3 bash
      - cd /data
      - touch CheerUp.file
      - exit
      - **docker exec** volume4 **ls -l /data** (여기에 동일한 CheerUp.file이 존재한다)

```
[root@docker3 volume]# docker exec volume4 ls -l /data
total 0
-rw-r--r-- 1 root root 0 Dec  4 16:22 CheerUp.file
```

      - find / -name "CheerUp.file"
      - **docker volume inspect shared**

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **WORKDIR**
    - **명령을 실행하기 위한** 디렉토리를 지정하는 것이다
    - 리눅스의 cd와 비슷하지만, cd에 더하여 **작업할 디렉토리를 지정하는 것이다**
      - 아래 예제는 **imsi.sh 파일을 실행할 디렉터리가 /root/**임을 Image 생성할 때 지정하는 것이다. 그러므로 미리 /root/에 imsi.sh 파일이 존재해야 한다
    - Docker Host에서 생성한 imsi.sh 파일을 Container 안의 /root/imsi.sh로 복사하는 Dockerfile을 생성하여 본다
      - mkdir /lab/**workdir** ; cd /lab/**workdir**
      - **vi imsi.sh**
        - #!/bin/bash
pwd
echo "Peace be with you!"
      - **chmod +x** imsi.sh
      - **vi Dockerfile**
        - FROM centos:7
COPY imsi.sh /root/imsi.sh
WORKDIR /root/
CMD ./imsi.sh

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **WORKDIR**
    - **docker build -t img_workdir_1 .**
    - docker run --name workdir1 img_workdir_1

```
[root@docker3 workdir]# docker run --name workdir1 img_workdir_1
/root
Peace be with you!
```

      - Container를 실행하면 기본적으로 WORKDIR로 경로를 이동하고, WORKDIR로 복사된 imsi.sh 파일이 자동으로 실행된다
      - imsi.sh 파일을 실행하는데, 이 파일은 WORKDIR 경로에 있어야 한다
    - docker run --name workdir2 **-it** img_workdir_1 **/bin/bash**
      - **pwd** (## 현재 위치가 /root/ 임을 알 수 있다)
      - 이렇게 하면 CMD의 imsi.sh 파일을 실행하지 않고 /bin/bash를 실행하는 것이다
      - 하지만 Container를 실행하면 기본적으로 WORKDIR 위치로 이동된 것을 알 수 있다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **WORKDIR**
    - WORKDIR 명령은 Dockerfile의 **"RUN, CMD, ENTRYPOINT, COPY, ADD 명령"**을 실행하기 위한 작업할 디렉토리를 지정하는 것이다
    - Docker 이미지 안의 파일은 **절대 경로로 지정**하거나, 아니면 **WORKDIR 명령에서** 지정한 디렉토리를 기점으로 한 경로로 지정한다(**상대 경로**)
    - **/home_dir/web** 디렉토리에 host.html 파일을 복사하기 위한 명령 예제
      - **WORKDIR** /home_dir
      - **ADD** host.html web/

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **COPY, ADD**
    - COPY와 ADD는 Docker host의 파일 또는 디렉토리를 Container 안의 경로로 복사한다
    - COPY는 Docker host에서 Container 안으로 복사만 가능하다
    - ADD는 복사 기능 뿐 아니라 **원격 파일 다운로드** 또는 **압축 해제** 등과 같은 기능도 갖고 있다
      - **ADD** http://down.cloudshell.kr/docker/supermario.sh **/root/mario.sh**
    - mkdir /lab/add ; cd /lab/add
    - vi Dockerfile
      - FROM centos:7
        ADD http://down.cloudshell.kr/docker/supermario.sh **/root/mario.sh**
        WORKDIR /root/
        CMD ["/bin/bash"]

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **COPY, ADD**
    - **docker build -t img_add_1 .**
    - docker run --name add1 -it img_add_1
    - pwd
    - ls -l

```
[root@docker3 add]# docker run --name add1 -it img_add_1
[root@267a6f9d53b9 ~]# pwd
/root
[root@267a6f9d53b9 ~]# ls -l
total 8
-rw------- 1 root root 3416 Nov 13  2020 anaconda-ks.cfg
-rw------- 1 root root   85 May 20  2022 mario.sh
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **ENV, ARG**
    - ENV는 **Dockerfile 또는 Container 안에서** 환경 변수(WORKDIR 사용)로 사용이 가능하다
    - ARG는 **Dockerfile에서만** 환경 변수로 사용이 가능하다
    - mkdir /lab/**env** ; cd /lab/**env ; cp** /lab/workdir/imsi.sh /lab/env/imsi.sh
    - vi Dockerfile
      - FROM centos:7
        COPY imsi.sh /root/env/imsi.sh
        ENV DIR=/root/env/
        RUN echo **${DIR}**
        CMD **${DIR}/**imsi.sh
    - **docker build** -t img_env_1 .
    - **docker run** --name env1 img_env_1

```
Step 2/5 : COPY imsi.sh /root/env/imsi.sh
 ---> d89d19730052
Step 3/5 : ENV DIR=/root/env/
 ---> Running in 712c98fb4b37
Removing intermediate container 712c98fb4b37
 ---> 08de0e887475
Step 4/5 : RUN echo ${DIR}
 ---> Running in 7a282b23c7f0
/root/env/
Removing intermediate container 7a282b23c7f0
 ---> 76be001d3e5c
Step 5/5 : CMD ${DIR}/imsi.sh
 ---> Running in 726f918699d9
```

```
[root@docker3 env]# docker run --name env1 img_env_1
/
Peace be with you!
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **ENV, ARG**
    - mkdir /lab/**arg** ; cd /lab/**arg ; cp** /lab/workdir/imsi.sh /lab/arg/imsi.sh
    - vi Dockerfile
      - FROM centos:7
        COPY imsi.sh /root/arg/imsi.sh
        ARG DIR=/root/arg/
        RUN echo **${DIR}**
        CMD **${DIR}/**imsi.sh
    - **docker build** -t img_arg_1 .
    - **docker run** --name arg1 img_arg_1

```
[root@docker3 arg]# docker run --name arg1 img_arg_1
/bin/sh: /imsi.sh: No such file or directory
```

      - CMD ${DIR}/imsi.sh 이 명령이 제대로 실행되지 않는다. ARG의 환경변수는 container를 실행할 때는 적용되지 않기 때문이다

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **USER**
    - "RUN, CMD, ENTRYPOINT 명령"을 실행하기 위한 **특정 사용자를 지정해야** 하는 상황에서 사용한다
    - USER 명령에서 지정하는 사용자는 RUN 명령으로 미리 생성해 놓아야 한다
    - mkdir /lab/user ; cd /lab/user
    - vi Dockerfile
      - FROM centos:7
        RUN useradd **dockerstudent**
        RUN whoami
        USER dockerstudent
        RUN whoami
        CMD ["/bin/bash"]
    - **docker build** -t img_user_1 .
    - **docker run** --name user1 **-it** img_user_1

```
Step 2/7 : RUN useradd dockerstudent
 ---> Running in 027cfc0bb91e
Removing intermediate container 027cf
 ---> eaf82f97bc0f
Step 3/7 : RUN whoami
 ---> Running in cdf6a865b33f
root
Removing intermediate container cdf6a
 ---> c9f05d1f5f51
Step 4/7 : USER dockerstudent
 ---> Running in 8a231e53c3b7
Removing intermediate container 8a231
 ---> 451042162518
Step 5/7 : RUN whoami
 ---> Running in c994b78bfea0
dockerstudent
```

```
[root@docker3 user]# docker run --name user1 -it img_user_1
[dockerstudent@cf982aa34adb /]$ cd ~
[dockerstudent@cf982aa34adb ~]$ ls -l
total 0
-rw-r--r-- 1 dockerstudent dockerstudent 0 Dec  5 13:48 myfile.txt
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **LABEL**
    - 도커 이미지에 **버전 정보, 작성자 정보, 설명** 등과 같은 정보를 제공할 때 사용한다
    - mkdir /lab/**label** ; cd /lab/**label**
    - vi Dockerfile
      - **FROM jesuswithme/nginxdemos**
        LABEL maintainer "Yongshik Lee<jesuswithme@gmail.com>"
        LABEL title="WebApp"
        LABEL version="1.0"
        LABEL description="This image is WebApp for education"
    - **docker build** -t img_label_1 .
    - **docker image inspect** --format="{{ .Config.Labels }}" img_label_1

```
[root@docker3 label]# docker image inspect --format="{{ .Config.Labels }}" img_label_1
map[description:This image is WebApp for education maintainer:Yongshik Lee<jesuswithme@gmail.com> title:WebApp
 version:1.0]
```

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **HEALTHCHECK**
    - Container의 Process 상태를 점검하는 것이다
    - Dockerfile에서 HEALTHCHECK는 **하나의 명령만이 유효하고**, 만약 여러 개가 있다면 **가장 마지막에 선언된 HEALTHCHECK가 적용된다**
    - 두 가지 방식으로 사용된다
      - HEALTHCHECK [OPTIONS] CMD command
        - Container 내부에서 명령을 실행하여 Container 상태 확인
        - 이 방법을 통해 웹페이지 등을 확인할 수 있다
      - HEALTHCHECK NONE
        - Base Image에서 상속된 상태 확인을 비활성화
    - centos7에서 nginx를 실행하는 이미지를 만들 때 HEALTHCHECK 명령을 사용하여 본다
      - mkdir /lab/**healthcheck** ; cd /lab/**healthcheck** ; **cp /lab/centos/Dockerfile .**
      - vi Dockerfile (##아래 내용을 제일 아래에 추가한다)
        - HEALTHCHECK --interval=10s --timeout=3s CMD curl -f localhost || exit 1
          - 10초마다(interval) HEALTHCHECK를 하고, 3초 이상이 소요(timeout)되면서 3번의 재시도(retries, 초기값3)가 실패하면 unhealthy 상태로 변경됨

Docker Image Layer란?

- Dockerfile 생성할 때 사용하는 명령어
  - **HEALTHCHECK**
    - **docker build -t img_healthcheck_1 .**
    - **docker run -d img_healthcheck_1**

    - Container에서 실행중인 Web Service의 상태 확인하기
      - **docker ps**

```
[root@docker3 healthcheck]# docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS
   NAMES
c69c1ea1691f   img_healthcheck_1    "nginx -g 'daemon of…"   17 seconds ago   Up 16 seconds (healthy)
```

      - **docker container inspect** c69 | grep health

```
[root@docker3 healthcheck]# docker container inspect c69 | grep health
                "Status": "healthy",
            "Image": "img_healthcheck_1",
```