EXPERIMENT-2: VOLUMES

Volume is simply a directory inside our container. Firstly, We have to declare this as a volume and then share Volume.

Even if we stop container, still we can access volume. Volume will be created in one container. We can declare a directory as a volume only while creating Container. You can't Create Volume from existing container. You can share one volume across any number of container. Volume wil not be included when you update an image and make a new container of that image. Volume can be mapped container to container and Host to container and vice-versa.

Benefits:

- Decoupling Cintainers from storage.
- Share Volume among different Containers.
- Attach Volume to Containers.
- On deleting containers volume does not delete.

Lets create a volume using Dockerfile.

Step-1

Create dockerfile and type below command

Vi Dockerfile //to open dockerfile

In dockerfile use below command:

FROM ubuntu

VOLUME ["/myvolume"]

Now to execute this dockerfile to make an image of your own use command.

docker build –t <image_name>

docker build –t bydockerfile5

```
[root@ip-172-31-3-238 ec2-user]# vi Dockerfile
[root@ip-172-31-3-238 ec2-user]# docker build -t bydockerfile5 .
Sending build context to Docker daemon  38.91kB
Step 1/2 : FROM ubuntu
 ---> f63181f19b2f
Step 2/2 : VOLUME ["/myvolume"]
 ---> Running in de758334b7f2
Removing intermediate container de758334b7f2
 ---> 171e4b36e124
Successfully built 171e4b36e124
Successfully tagged bydockerfile5:latest
```

Step-2:

Create a container of the above image we created.

Command:

docker run -it –name <container_name> <image_name> /bin/bash

docker run -it –name volumecontainer2 bydockerfile5 /bin/bash

After this you will be redirect to your bash of the container and you will see your volume directory with a name myvolume. Make some files like file.txt, file.sh, file inside myvolume container and do exit from that container.

```
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer2 bydockerfile5 /bin/bash
root@67786b1d5f9a:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  myvolume  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@67786b1d5f9a:/# cd myvolume
root@67786b1d5f9a:/myvolume# ls
root@67786b1d5f9a:/myvolume# touch file1 file2.txt file3.sh
root@67786b1d5f9a:/myvolume# ls
file1  file2.txt  file3.sh
root@67786b1d5f9a:/myvolume# exit
exit
```

Step-3: Now we have to create a new container which we can have of any image while giving the access of the volume directory.

Command:

docker run -it --name <new_container_name> --privileged=true --volume-from <container_consist_volume_Directory> <image_name_any> /bin/bash

docker run -it --name volumecontainer4 –privileged=true --volume-from volumecontainer2 bydockerfile /bin/bash

```
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer4--privileged --volume-from volumecontainer2 bydockerfile /bin/bash
unknown flag: --volume-from
See 'docker run --help'.
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer4--privileged --volumes-from volumecontainer2 bydockerfile /bin/bash
root@769bd4417e3a:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  myvolume  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@769bd4417e3a:/# cd tmp
root@769bd4417e3a:/tmp# ls
MyLogFile
root@769bd4417e3a:/tmp# cd ..
root@769bd4417e3a:/# cd myvolume
root@769bd4417e3a:/myvolume# ls
file1  file2.txt  file3.sh  file4.txt
root@769bd4417e3a:/myvolume#
```

Now, since we can create n number of containers giving access of volume directory.

If any container create a file in “myvolume” directory will reflect changes in all the containers having volume directory privileged.

Let’s create a volume using command line

Step-1:

Commad:

docker run -it --name volumecontainer5 -v /volume2 ubuntu /bin/bash

-v is used to declare a volume directory.

```
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer5 -v /volume2 ubuntu /bin/bash
root@6d0cee830f22:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volume2
root@6d0cee830f22:/# cd volume2
root@6d0cee830f22:/volume2# ls
root@6d0cee830f22:/volume2# touch vol1 vol2 vol3
root@6d0cee830f22:/volume2# ls
vol1  vol2  vol3
root@6d0cee830f22:/volume2# exit
exit
```

Step-2: Create another container on any image and give volume privileged.

Command: docker run -it --name volumecontainer6 --privileged=true --volumes-from volumecontainer5 ubuntu /bin/bash

```
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer6 --priviledge=true --volume-from volumecontainer5 ubuntu /bin/bash
unknown flag: --priviledge
See 'docker run --help'.
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer6 --privileged=true --volume-from volumecontainer5 ubuntu /bin/bash
unknown flag: --volume-from
See 'docker run --help'.
[root@ip-172-31-3-238 ec2-user]# docker run -it --name volumecontainer6 --privileged=true --volumes-from volumecontainer5 ubuntu /bin/bash
root@9fe70e0f06c3:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volume2
root@9fe70e0f06c3:/# cd volume2
root@9fe70e0f06c3:/volume2# ls
vol1  vol2  vol3
```

How to share Volume Between host and container

Step-1:

Check your present directory using ‘pwd’ command. Create some file, if already present no need to make.

/home/ec2-user

Command to create file: touch file1 file2.txt

We can map our container and host volume while creating a container using ‘:’ symbol and ‘-v’ is used to define a volume we will create.

Command:

docker run -it --name <container_name> -v /home/ec2-user:/<volume name> --privileged=true <image_name> /bin/bash

docker run -it --name hostcontainer -v /home/ec2-user:/bansal --privileged=true ubuntu /bin/bash

We will see all the file present in our host is mapped to a container volume named ‘bansal’.

```
[ec2-user@ip-172-31-3-238 ~]$ sudo su
[root@ip-172-31-3-238 ec2-user]# pwd
/home/ec2-user
[root@ip-172-31-3-238 ec2-user]# ls
Dockerfile  Dopckerfile  file1  file2  test  testfile1  test.tar.gz
[root@ip-172-31-3-238 ec2-user]# docker run -it --name hostcontainer -v /home/ec2-user:/bansal --privileged=true ubuntu /bin/bash
root@930b085a6bf9:/# ls
bansal  bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@930b085a6bf9:/# cd bansal
root@930b085a6bf9:/bansal# ls
Dockerfile  Dopckerfile  file1  file2  test  test.tar.gz  testfile1
root@930b085a6bf9:/bansal# exit
exit
```

Step-2:

To verify exit the container again start the container and add some files in a volume directory and exit the container. Check the update pwd of the host.

```
[root@ip-172-31-3-238 ec2-user]# docker start hostcontainer
hostcontainer
[root@ip-172-31-3-238 ec2-user]# docker attach hostcontainer
root@930b085a6bf9:/# ls
bansal  bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@930b085a6bf9:/# cd bansal
root@930b085a6bf9:/bansal# ls
Dockerfile  Dopckerfile  file1  file2  test  test.tar.gz  testfile1
root@930b085a6bf9:/bansal# ^C
root@930b085a6bf9:/bansal#
root@930b085a6bf9:/bansal# touch abcd.txt
root@930b085a6bf9:/bansal# ls
Dockerfile  Dopckerfile  abcd.txt  file1  file2  test  test.tar.gz  testfile1
root@930b085a6bf9:/bansal# exit
exit
[root@ip-172-31-3-238 ec2-user]# pwd
/home/ec2-user
[root@ip-172-31-3-238 ec2-user]# ls
abcd.txt  Dockerfile  Dopckerfile  file1  file2  test  testfile1  test.tar.gz
[root@ip-172-31-3-238 ec2-user]#
```