Table of contents

Using Volume to persist data of the container

- In some scenarios, you have to access the files on the host system, or if you want to persist the data even if the container is killed, or share data across containers.
- Some Database or Files that container and other applications should access.
- In above scenarios, `Docker Volumes` is used for managing files outside the lifecycle of the container.
- Although this is something where containers are dependent on a compabitible file or DB Mount, its a useful feature.

Bind Mounts

- When you use a bind mount, a file or directory on the host machine is mounted into a container.
- The file or directory is referenced by its absolute path on the host machine.
- It helps to access files on Host from within a container.
- Use Docker's volume flag to access host files from within the container. illustrates the use of a volume flag to interact with the host's filesystem.
  - Lets create a small script and schedule in crontab in Host Linux to write some content in a file
  - Add below content to Shell Script : `test-cron-job.sh`

```
#!/bin/bash
echo “Cron ran successfully at : $(date)” >> /tmp/test-file.html
```

- Schedule crontab with

```
crontab -e

# Add below in crontab editor
* * * * * bash /home/ec2-user/test-cron-job.sh
```

- The following command shows the host's `/tmp` directory being mounted on `/var/data1`, and it could be run to start the container.

```
docker run -v /tmp:/var/data1 -it ubuntu:20.04 bash
cat /var/data1/test-file.html
```

```
cd /var/data1/
echo "written from docker container with Hostname as $HOSTNAME" >> test-file.html
```

- In the case of bind mounts, the first field is the path to the file or directory on the host machine.
- The second field is the path where the file or directory is mounted in the container.
- Similarly, even if multiple containers are launched with above same command, the Host directory will be mounted on those containers.

Docker Volumes:

- Creates a new volume that containers can consume and store data in.
- Use below to Create a volume and then configure the container to use it:
- Let's pull the latest nginx image from the docker hub and run the container and load the home page which listens on port 80.

```
docker run -it --name=WebApp -d -p 80:80 nginx
netstat -nltp
```

- Access the nginx home page in the browser
- Go inside the container and edit the content of `/usr/share/nginx/html`

```
docker ps
docker exec -it WebApp bash
cat /usr/share/nginx/html/index.html
echo "Changing the content of the home page from Hostname as $HOSTNAME" >
/usr/share/nginx/html/index.html
```

- `docker container stop <CONTAINER_ID>`
- `docker container rm <CONTAINER_ID>`
- We can use `docker stop` and `docker start` to check if content that is changed is accessible.
- If this container is stopped or somehow gets killed, and another container is loaded, the changes made in the previous container are not accessible anymore.

For example uses of this command, refer to the examples section below.

- Mount a same path on Host Directory and creating multiple containers and write the hostname using CMD in a file that is mounted.

Creating Docker Volumes

- Volumes are saved in the host filesystem `/var/lib/docker/volumes/` which is owned and maintained by docker.

- Any other `non-docker` process can't access it, also as checked above other docker processes/containers can still access the data even container is stopped since it is isolated from the container file system.

```
# Check for any volumes
sudo ls /var/lib/docker/volumes/
#create docker volume
docker volume create datavolume
#list volumes
docker volume ls
#inspect volumes
docker volume inspect datavolume
##removing docker volumes
docker volume rm datavolume
```

- Stop and remove previously launched containers if any

```
docker container stop ContainerID
docker container rm ContainerID
```

- Now above scenario can be seen using docker volume to persist the changes in one container and access using another

Mounting a Data Volume

- To mount a data volume to a container add the `--mount` flag to the `docker run` command.
- To run a container and mount a data volume to it, follow the basic syntax: `docker run --mount source=[volume_name],destination=[path_in_container] [docker_image]`
- Replace `[path_in_container]` with the path where you want to place the data volume in the container.Everything stored in that directory automatically gets saved on the data volume on the host as well.

```
docker run -d --name=Container1 --mount
source=datavolume,destination=/usr/share/nginx/html -p 80:80 nginx
```

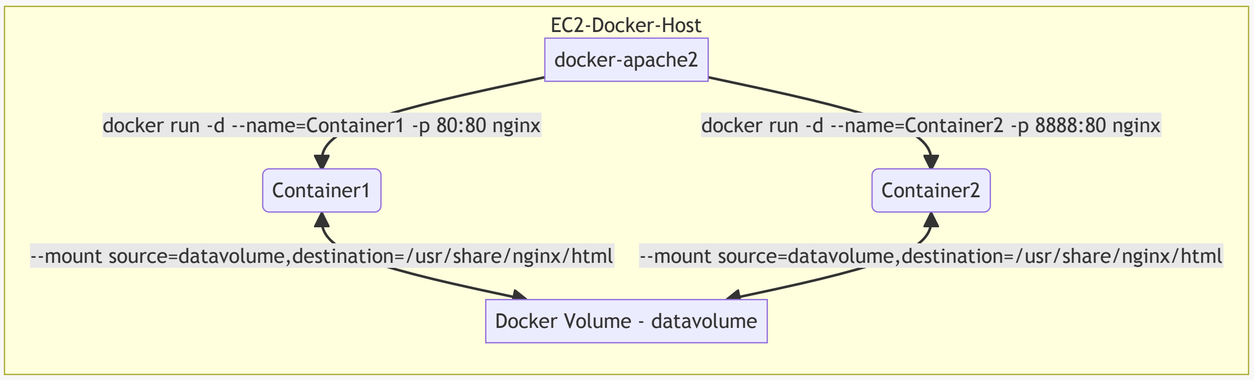
- Access the nginx home page in the browser

```
docker exec -it Container1 bash
ls /usr/share/nginx/html
echo "Changing the content of the home page from Hostname as $HOSTNAME" >>
/usr/share/nginx/html/index.html
```

- Check the content of the file from host : `sudo cat /var/lib/docker/volumes/datavolume/_data/index.html`
- Now if we launched another container and check the content, since Volume is shared by both containers the same file is accessible from this new container.

```
docker run -d --name=Container2 --mount
source=datavolume,destination=/usr/share/nginx/html -p 8888:80 nginx
docker exec -it Container2 bash
ls /usr/share/nginx/html
echo "Changing the content of the home page from Hostname as $HOSTNAME" >>
/usr/share/nginx/html/index.html
```

- Access the container on browser with port 8888 or using `curl localhost:8888`


EC2-Docker-Host
docker-apache2
docker run -d --name=Container1 -p 80:80 nginx
docker run -d --name=Container2 -p 8888:80 nginx
Container1
Container2
--mount source=datavolume,destination=/usr/share/nginx/html
--mount source=datavolume,destination=/usr/share/nginx/html
Docker Volume - datavolume

- Display containers information associated with an image

```
docker container ls -a --filter "ancestor=ubuntu:20.04"
```

- Display only container id associated with an image

```
docker container ls -a --filter "ancestor=ubuntu:20.04" -q
```