

Table of contents

- [Table of contents](#)
 - [Container Files](#)
 - [bash container files](#)
 - [nginx container files](#)
 - [Storage Overview](#)
 - [1.Bind Mounts](#)
 - [2.Docker Volumes](#)
 - [Creating Docker Volumes](#)
 - [Mounting a Data Volume](#)
 - [Docker Stats](#)
-

Container Files

bash container files

```
docker run -it bash:latest bash
bash-5.1# echo devops > file.txt && cat file.txt

# The output of the above command is 'devops'

# if we run the same image with just the command to output the file's contents:
docker run bash:latest bash -c "cat file.txt"
# The output of the above command will be as below
cat: can't open 'file.txt': No such file or directory

# This is because, each time u are launching a new container, the files are
created in container space,
# and are not directly available for other containers to read.
# The second run of the container runs on a clean file system, so the file is not
found.
```

- Two containers are not able to see the same files.
- If the container gets terminated/killed, the files created by that container are lost
- Containers is not able to access files that are on Host File System

nginx container files

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

- Access the nginx home page in the browser using the IP of the host and port mapping.
- Go inside the container and edit the content of `/usr/share/nginx/html`

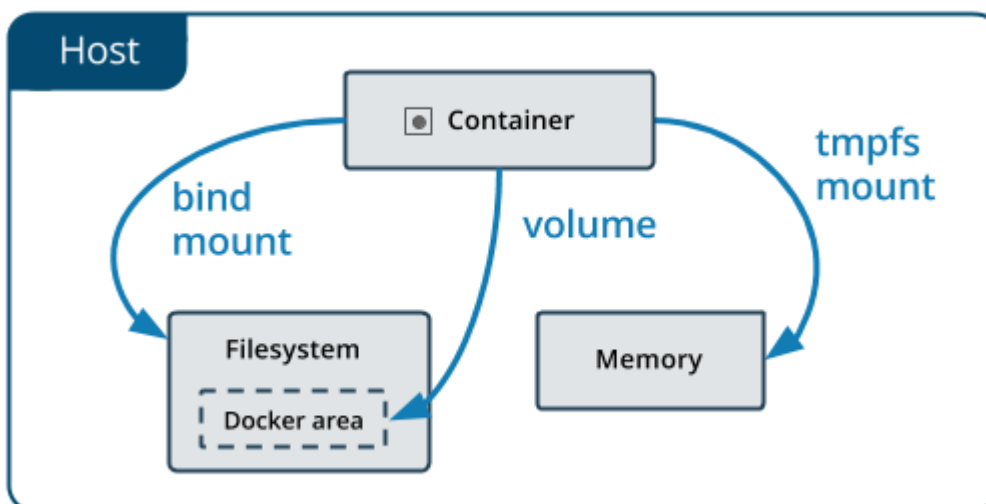
```
docker ps
docker exec -it WebApp bash
cat /usr/share/nginx/html/index.html
echo "Changing the content of the home page from Hostname as $HOSTNAME" >
/usr/share/nginx/html/index.html
# Escape from Container
docker container stop WebApp
docker container rm WebApp
```

- 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.

```
docker run --name=WebAppNew -d -p 80:80 nginx
docker exec -it WebAppNew bash
cat /usr/share/nginx/html/index.html
```

Storage Overview

- By default all files created inside a container are stored on a writable container layer. This means that:
 - The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops: **bind mounts** and **volumes**.



1.Bind Mounts

- A Docker **bind mount** is a connection from the container to a directory on the host machine.
- It allows the **host to share its own file system with the container**, which can be made **read-only** or **read-write**.
- This allows to use a container to work with our files on the host.
- The file or directory is referenced by its **absolute path** on the host machine.

--

- Below are the two options using which directory on the host can be mounted:
 - **-v** or **--volume** : The **-v** option contains three components, separated by colons:
 - **Source directory** i.e `$(pwd)`
 - Mount point within the container i.e `/var/data`
 - (Optional) **ro** if the mount is to be read-only.
 - **--mount** : Consists of multiple key-value pairs, separated by commas and each consisting of a **<key>=<value>**
 - The **type** of the mount, which can be **bind**, **volume**.
 - The **source** of the mount. For bind mounts, this is the path to the file or directory on the Docker daemon host. Can be specified as **source** or **src**.
 - The **destination** takes as its value the path where the file or directory is mounted in the container. Can be specified as **destination**, **dst**, or **target**.

--

`pwd`

```
docker run -v $(pwd):/var/data -it bash:latest
# Inside the container
cd /var/data/
echo "File created from Containers" > /var/data/file.txt
# Enter the escape sequence: Ctrl+P and Ctrl+Q
# Access this file from host
# The files created inside the container can be accessible from host.
```

```
# launch another container
docker run -v $(pwd):/var/data -it bash:latest
cd /var/data/
cat file.txt
```

OR

```
docker run --mount type=bind,source="$(pwd)",target=/var/data -it bash:latest
ls /var/data/
cat /var/data/file.txt
```

```
# To view the Mount of the existing Container
docker inspect CONTAINER
#
"Mounts": [
```

```

    {
        "Type": "bind",
        "Source": "/root/mount_testing",
        "Destination": "/var/data",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
]

```

--

- If containers are lost/killed, the files are still accessible on the host path.
- After running this command, we will have `file.txt` in the working directory of our host machine.
- Similarly, even if multiple containers are launched with above same command, the same host directory will be mounted on those containers.

2.Docker Volumes

- A bind mount uses the host file system, but **Docker volumes** are native to **Docker**.
- **Docker Volumes** can be shared between containers.
- **Docker Volumes** is used for managing files outside the lifecycle of the container.
- The data is kept on storage attached to the host – often the local filesystem.
- The volume itself has a lifecycle that's longer than the container's, allowing it to persist until no longer needed.

--

Creating Docker Volumes

- Create a new volume that containers can consume and store data in.
- Use below command to Create a volume and then configure the container to use it:
- 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, other docker **processes/containers** can still access the data even container is stopped since it is isolated from the container file system.

```

# create docker volume
docker volume create datavolume
# list volumes
docker volume ls
# Check for any volumes
sudo ls /var/lib/docker/volumes/
# 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 volume ls
# create docker volume
docker volume create datavolume
docker run -d --name=Container1 --mount
type=volume,source=datavolume,destination=/usr/share/nginx/html -p 80:80 nginx

#Inspect the container
docker inspect Container1

"Mounts": [
  {
    "Type": "volume",
    "Name": "datavolume",
    "Source": "/var/lib/docker/volumes/datavolume/_data",
    "Destination": "/usr/share/nginx/html",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
]
```

--

- Access the nginx home page in the browser

```
docker exec -it Container1 bash
ls /usr/share/nginx/html
```

```
cat /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 this **Volume** i.e **datavolume** is shared by both containers the same file is accessible from this new container.

--

```
docker ps
docker run -d --name=Container2 --mount
type=volume,source=datavolume,destination=/usr/share/nginx/html -p 8888:80 nginx

#Inspect the container
docker inspect Container2
"Mounts": [
  {
    "Type": "volume",
    "Name": "datavolume",
    "Source": "/var/lib/docker/volumes/datavolume/_data",
    "Destination": "/usr/share/nginx/html",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
]

docker exec -it Container2 bash
ls /usr/share/nginx/html
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

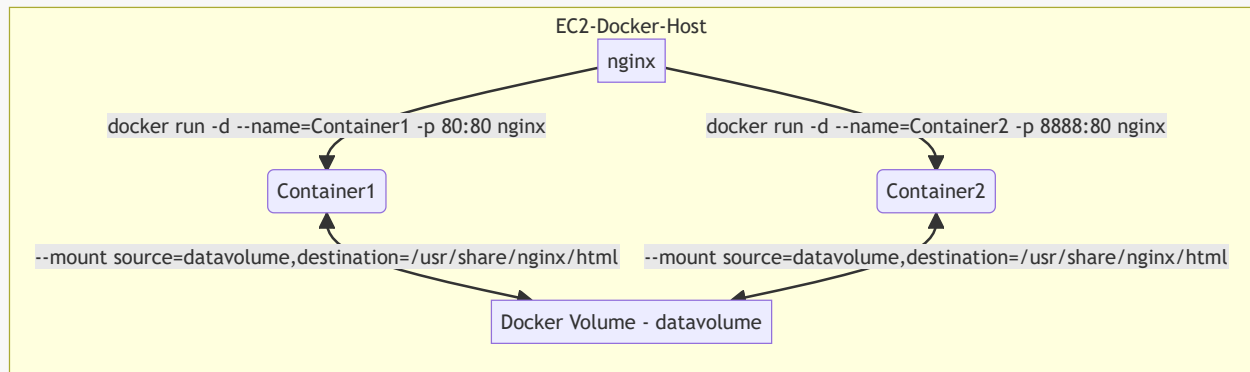
# To remove volume
docker volume ls
docker volume rm datavolume
[ec2-user@ip-172-31-24-162 ~]$ docker volume rm datavolume
Error response from daemon: remove datavolume: volume is in use -
[c1b796f7a48c0b877189ba8802b75fe122d82dea3ff26add770d23e4c9599b6e, 8ec7e7f84355
04f8ea83523074f2b11059067e7f4cca2b8b5f115653e6ceafa4]

docker stop Container1 Container2
docker rm Container1 Container2

docker volume rm datavolume
docker volume ls
```

--

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



Docker Stats

- `docker stats` : Display a live stream of container(s) resource usage statistics

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET
I/O	BLOCK I/O	PIDS			
ff70f0a5ae9f	great_bartik	0.00%	768KiB / 965.8MiB	0.08%	1kB /
0B	0B / 0B	1			
c559ac5e30c9	frosty_newton	0.00%	772KiB / 965.8MiB	0.08%	1.33kB
/ 0B	0B / 0B	1			
a20bf7d5fc8e	sad_jennings	0.00%	764KiB / 965.8MiB	0.08%	1.4kB
/ 0B	0B / 0B	1			
03071a5fd70d	elastic_sinoussi	0.00%	780KiB / 965.8MiB	0.08%	1.4kB
/ 0B	0B / 0B	1			
d5086de0b615	stupefied_dirac	0.00%	3.113MiB / 965.8MiB	0.32%	1.84kB
/ 0B	3.18MB / 0B	1			

--

- To remove all unused volumes and free up space:

```
$ docker volume prune
```

- Display containers information associated with an image

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