Managing Data for Containers

The file system management of a container.

We'll cover the following

In the last lesson, we looked at how to commit images to the Docker Hub and access it anywhere. Whenever you create a container from an image, it creates a new container without any data except the image data. We created the date-project image, which is a very small image. It has only one project file which we created in the container file system. If the container is removed before committing the changes, we will lose the data. So, it is always good practice to separate your data's file system from the container's file system.

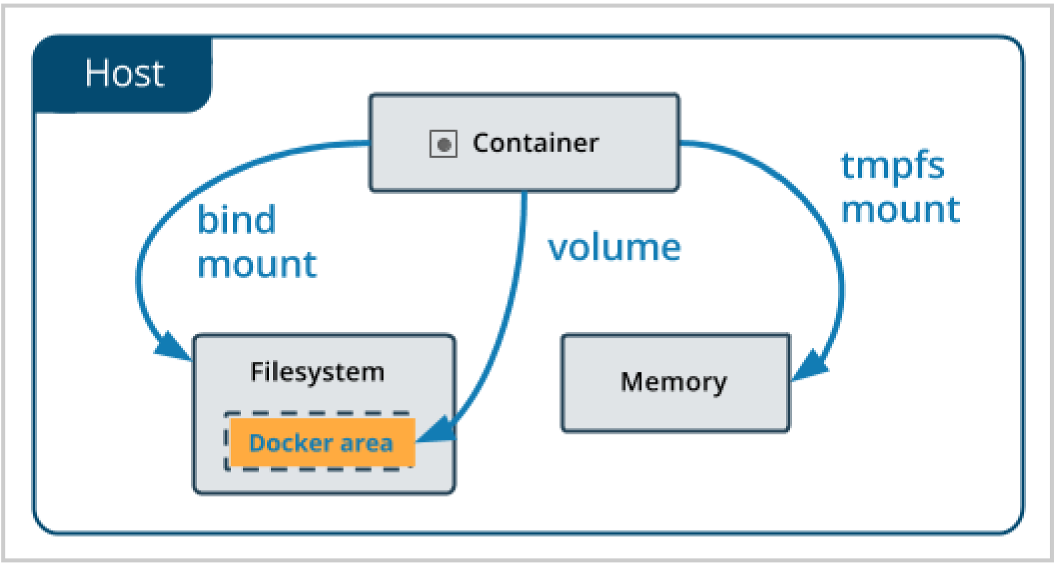
Whenever a container is created, a file system is also created with it, which is a default Linux filesystem. Although Docker shares the OS's kernel, there is a separation between file systems.

Most times, you need to access the host files in the container for faster access to data and while coding as well, because you cannot code, build, and then check your code.

Docker's bind mount and volumes can be used in such cases. Let's check them out one by one.

Commands used in this lesson

- docker volume --help: to get the volume help
- docker volume create: to create a new volume
- docker inspect volume: to inspect the created volume
- docker run -v: to mount a volume



Bind mount And volume. Ignore the temporary file system mount for now.

Bind mount

This is available from a very early version of Docker. In bind mount, you use the host filesystem and mount it on the container using -v flag with the run command.

For Linux and Mac:

`$docker run -it -v <absolute_path>:<folder path or new folder name> date_project:1.0`

```
docker run -it -v /Users/venkateshachintalwar/Desktop/:/desktop date_project:1.0
root@00c55886cc27:/# ls
bin  boot  desktop  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys
root@00c55886cc27:/#
```

On `line 3`, you can see the desktop folder. If you `cd` into it, you will be able to access the folder from inside the container. You will also get access to any changes created in the container from the container itself or the host will be reflected on both sides.

> To mount the file system as read-only, use `ro` flag. `$docker run -it -v <absolute_path>:<folder path or new folder name>:ro date_project:1.0`

```
root@00c55886cc27:/# cd desktop/
root@00c55886cc27:/desktop# ls
'Screen Recording'                 'Screenshot 2019-09-20 at 5.15.21 PM.png'    'S
```

```
creenshot 2019-10-08 at 5.57.55 PM.png'
'Screenshot 2019-07-30 at 12.08.13 PM.png'  'Screenshot 2019-09-24 at 11.46.0
6 AM.png'  'Screenshot 2019-10-08 at 5.58.12 PM.png'
'Screenshot 2019-07-30 at 12.32.35 PM.png'  'Screenshot 2019-09-29 at 9.44.4
0 PM.png'   'Screenshot 2019-10-09 at 7.56.35 PM.png'
'Screenshot 2019-08-05 at 3.21.05 PM.png'   'Screenshot 2019-10-04 at 4.59.5
4 PM.png'   'Screenshot 2019-10-09 at 7.57.29 PM.png'
'Screenshot 2019-08-05 at 3.35.48 PM.png'   'Screenshot 2019-10-06 at 7.59.5
9 PM.png'    plan_items.csv
'Screenshot 2019-08-05 at 3.41.06 PM.png'   'Screenshot 2019-10-07 at 12.39.4
4 AM.png'   query_result.csv
'Screenshot 2019-08-27 at 5.13.37 PM.png'   'Screenshot 2019-10-08 at 11.10.2
6 PM.png'   site_visit_view.csv
'Screenshot 2019-09-17 at 8.50.59 PM.png'   'Screenshot 2019-10-08 at 11.10.4
1 PM.png'
'Screenshot 2019-09-20 at 11.45.27 AM.png'  'Screenshot 2019-10-08 at 11.11.0
9 PM.png'
```

Volumes

Bind mount has some limitations and is dependent on the host's file system. If a folder is accidentally deleted from the host, Docker can't do anything.

On the other hand, volumes are created in Docker space which provides more control over using the Docker CLI.

> Docker volumes are mostly created to share data within different containers, rather than sharing data with host and container.

Let's create one volume below.

`$docker volume --help` to see all available options.

```
(base) Administrators-MacBook-Pro:~ venkateshachintalwar$ docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes


Run 'docker volume COMMAND --help' for more information on a command.
```

Let’s create one volume: `$docker volume create <volume_name>`

If you don’t provide the volume name, Docker will assign a random unique one.

```
(base) Administrators-MacBook-Pro:~ venkateshachintalwar$ docker volume create project_directory
project_directory
(base) Administrators-MacBook-Pro:~ venkateshachintalwar$ docker volume ls
DRIVER              VOLUME NAME
local               832d0428f56fcdf8aca95768c0a0e4481ea373a7c37d7c9e2a7ef7f8b1a002fd
local               e553eed0ac69ab332c39fef0afebbe63bd6eae5fd1c8d37f9f29d7cd1728ece8
local               project_directory
```

If you inspect the Docker volume,

```
(base) Administrators-MacBook-Pro:~ venkateshachintalwar$ docker volume inspec
t project_directory
[
    {
        "CreatedAt": "2019-10-12T09:30:57Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/project_directory/_data",
        "Name": "project_directory",
        "Options": {},
        "Scope": "local"
    }
]
```

From the code above, you can see the mount point, however you cannot access it directly if you are using Mac and Windows.

Let’s run the container using this volume. `$ docker run -it -v project_directory:/project date_project:1.0`

```
(base) Administrators-MacBook-Pro:lib venkateshachintalwar$ docker run -it -v project_directory:/p
root@86c2fe895444:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  project  root  run  sbin  srv  sys
root@86c2fe895444:/# cd project/
root@86c2fe895444:/project# ls
root@86c2fe895444:/project# mkdir app
root@86c2fe895444:/project# ls
```

```
app
```

We have successfully mounted the volume onto the container file system. Now, let’s share this volume with another container so that the newly-created app folder should be accessible to the newly-created container.

```
(base) adminisatorsmbp:Desktop venkateshachintalwar$ docker run -it -v project_directory:/project_
root@4a95ba56b066:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  project_in_second  root  run  sbin
root@4a95ba56b066:/# cd project_in_second/
root@4a95ba56b066:/project_in_second# ls
app
```

So, using volumes, we can share data in different containers. Volumes are more reliable than bind mounts.

Congratulations! With this, you have completed the Docker fundamentals. pull images, run containers, create and share images, and work with shared data volumes.

In the next lesson, we will see some frequently-used commands so that it will be easy to work with Docker. You will also get all the commands in one place.