Docker Data Volumes

If you have been following the series so far, we have launched Containers from various images and even created a basic image for our own use. To reiterate the point, Containers are Ephemeral and once a container is removed, it is gone. What about scenarios where you want the applications running inside the container to write to some files/data and then ensure that the data is still present. For e.g. let’s say that you are running an application that is generating data and it creates files or writes to a database and so on. Now, even if the container is removed and in the future you launch another container, you would like that data to still be there.

In other words, the fundamental thing that we are trying to get over here is to separate out the container lifecycle from the data. Ideally we want to keep these separate so that the data generated is not destroyed or tied to the container lifecycle and can thus be reused. This is done via Volumes, which we shall see via several examples.

As per the official documentation, there are 2 ways in which you can manage data in Docker:

- Data volumes
- Data volume containers

Let us work through examples for both the above cases.

Few points to keep in mind about Data Volumes

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated. Data volumes can be shared across containers too. They could be mounted in read-only mode too.

Mounting a Data volume

Let us begin first with the most basic operation i.e. mounting a data volume in one of our containers. We will be working with the **busybox** image to keep things simple.

We are going to use the -v [/VolumeName] as an option to mount a volume for our container. Let us launch a container as given below:

```
$ docker run -it -v /data --name container1 busybox
```

This will launch a container (named container1) in interactive mode and you will be at the prompt in the container.

Give the ls command as shown below:

```
$ docker run -it -v /data --name container1 busybox
/ # ls
bin   data  dev   etc   home  proc  root  sys   tmp   usr   var
/ #
```

Notice that a volume named **data** is visible now.

Let us a do a cd inside the data volume and create a file named file1.txt as shown below:

```
/ # cd data
/data # touch file1.txt
/data # ls
file1.txt
/data #
```

So what we have done so far is to mount a volume /data in the container. We navigated to that directory (/data) and then created a file in it.

Now, let us exit the container by typing exit and going back to the terminal.

```
/data # exit
$
```

Now, if we do a docker ps -a , we should see our container (container1) currently in the exited state as shown below:

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
22ab6644ea6b busybox "/bin/sh" 3 minutes ago Exited (0) 23 seconds ago container1
```

Now, let us inspect the container and see what Docker did when we started this container.

Give the following command:

```
$ docker inspect container1
```

This will give out a JSON output and you should look for Mounts attribute in the output. A sample output from my machine is shown below:

```
"Mounts": [
{
"Type": "volume",
"Name": "568258a7940182c5ac52f0637c60c1d1f81e9ec77f3e4694647b4879c2ff003c",
"Source": "/var/lib/docker/volumes/568258a7940182c5ac52f0637c60c1d1f81e9ec77f3e4694647b4879c2ff003c/_data",
"Destination": "/data",
"Driver": "local",
"Mode": "",
"RW": true,
"Propagation": ""
}],
```

This tells you that when you mounted a volume (**/data**), it has created a folder **/var/lib/….** for you, which is where it puts all the files, etc that you would have created in that volume. Note that we had created a **file1.txt** over there (we will come to that in a while).

Also notice that the **RW** mode is set to true i.e. Read and Write.

Now that the container is stopped i.e. exited, let us restart the container (**container1**) and see if our volume is still available and that **file1.txt** exists.

```
$ docker restart container1
container1
$ docker attach container1
/ # ls
bin  data dev  etc  home proc root sys  tmp  usr  var
/ # cd data
/data # ls
file1.txt
/data #
```

The sequence of steps above are as follows:

1. We restarted the container (**container1**)

2. We attached to the running container (**container1**)
3. We did a **ls** and found that our volume **/data** is still there.
4. We did a **cd** into the **/data** volume and did a **ls** there. And our file is still present.

Hopefully this explains to you how you can persist your data in volumes.

Now, let's do an interesting thing. Exit the container and remove the container.

```
/data # exit
$ docker rm container1
container1
$ docker volume ls
DRIVER              VOLUME NAME
local               568258a7940182c5ac52f0637c60c1d1f81e9ec77f3e4694647b4879c2ff003c
```

This shows to you that though you have removed the **container1**, the data volume is still present on the host. This is a dangling or ghost volume and could remain there on your machine consuming space. Do remember to clean up if you want. Alternatively, there is also a **-v** option while removing the container as shown in the help:

```
$ docker rm --help
Usage: docker rm [OPTIONS] CONTAINER [CONTAINER...]
Remove one or more containers
```

```
-f, --force=false Force the removal of a running container (uses SIGKILL)
 --help=false Print usage
 -l, --link=false Remove the specified link
 -v, --volumes=false Remove the volumes associated with the container
```

This will always remove your volumes when the container is also removed.

Exercise: What happens if you launch another container with the same /data volume. Is the file still there or does each container get its own file system? Try it out.

Mounting a Host Directory as a Data volume

Now that we have seen how to mount a volume in the container, the next step is to look at the same process of mounting a volume but this time we will mount an existing host folder in the Docker container. This is an interesting concept and is very useful if you are looking to do some development where you regularly modify a file in a folder outside and expect the container to take note or even sharing the volumes across different containers.

To mount a host volume while launching a Docker container, we have to use the following format for volume **-v** :

```
-v HostFolder:ContainerVolumeName
```

So, let us start a busybox container as shown below:

```
$ docker run -it --name container1 -v /Users:/datavol busybox
/ #
```

What we have done here is that we have mapped the host folder /Users to a volume /datavol that will be mounted inside our container (container1).

Now, if we do a **ls** , we can see that the /datavol has been mounted. Do a cd into that folder and a ls, and you should be able to see the folder contents of C:\Users on your machine.

```
/ # cd datavol
/datavol # ls
/datavol #
```

Hope this makes it clear on how to use host folders.

Exercise:

1. Try adding a file directly from your laptop/machine in **/Users/<username>** folder and then go back to your running container and do a ls there. You should be able to see that new file there.
2. From the container shell, go to the **/datavol** folder and then add a file there. Then go back to your machine/laptop and do a ls. You should see the new files there.

Additional Note: Optionally if you require, you can mount a single host file too as a data volume.

Start thinking now of how you would use host folders that have been mounted as data volumes. Assume you are doing development and have the Apache Web Server or any other Web Server running in the container. You could have started the container and mounted a host director that the Web Server can use. Now on your host machine, you could make changes using your tools and those would then get reflected directly into your Docker container.

Data volume containers

We now come to the next part i.e. creating a Data volume container. This is very useful if you want to share data between containers or you want to use the data from non-persistent containers. The process is really two step:

1. You first create a Data volume container
2. Create another container and mount the volume from the container created in Step 1.

Let us see that in action:

We will first create a container (**container1**) and mount a volume inside of that:

```
$ docker run -it -v /data --name container1 busybox
```

Now, let us go into the /data volume and create two dummy files in it as shown below:

```
/ # cd data
/data # ls
/data # touch file1.txt
/data # touch file2.txt
/data #
```

Now launch another terminal.

Now, if we do a **docker ps**, we should see our running container:

```
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
006f7ba16783 busybox "/bin/sh" 27 seconds ago Up 26 seconds
```

Now, if we execute a command on the running **container1** i.e. see the contents of our **/data** volume, you can see that the two files are present.

```
$  docker exec container1 ls /data
file1.txt
file2.txt
```

Great ! Now, let us launch another container (**container2**) but it will mount the data volume from container1 as given below:

```
$ docker run -it --volumes-from container1 --name container2 busybox
```

Notice above that we have launched it in interactive mode and have used a new parameters— **volumes-from** <**containername**> that we have specified. This tells **container2** to mount the volumes that **container1** mounted.

Now, if we do a **ls**, we can see that the data folder is present and if we do a **ls**inside of that, we can see our two files: **file1.txt** and **file2.txt**

```
/ # ls
bin dev home lib64 media opt root sbin tmp var
data etc lib linuxrc mnt proc run sys usr
/ # cd data
/data # ls
file1.txt file2.txt
/data #
```

You can launch multiple containers too , all using the same data volume from **container1**. For e.g.

```
$ docker run -it --volumes-from container1 --name container3 busybox
$ docker run -it --volumes-from container1 --name container4 busybox
```