DOCKER VOLUMES

In general, Docker containers are ephemeral, running just as long as it takes for the command issued in the container to complete. Sometimes, however, applications need to share access to data or persist data after a container is deleted. Databases, user-generated content for a web site, and log files are just a few examples of data that is impractical or impossible to include in a Docker image but which applications need to access. Persistent access to data is provided with Docker Volumes.

Docker Volumes can be created and attached in the same command that creates a container, or they can be created independently of any containers and attached later. In this article, we'll look at four different ways to share data between containers.

1. Creating an Independent Volume.

Introduced in Docker's 1.9 release, the ***docker volume create*** command allows you to create a volume without relating it to any particular container. We'll use this command to add a volume named ***mydata***

$ docker volume create --name mydata

```
[ec2-user@ip-172-31-3-162 ~]$ docker volume create --name mydata
mydata
[ec2-user@ip-172-31-3-162 ~]$
```

The name is displayed, indicating that the command was successful.

To make use of the volume, we'll create a new container from the Ubuntu image, using the **--rm** flag to automatically delete it when we exit. We'll also use **-v** to mount the new volume. **-v** requires the name of the volume, a colon, then the absolute path to where the volume should appear inside the container. If the directories in the path don't exist as part of the image, they'll be created when the command runs. If they do exist, the mounted volume will hide the existing content.

$ docker run -ti --rm -v mydata:/c_mydata ubuntu

```
[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --rm -v mydata:/c_mydata ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
38e2e6cd5626: Pull complete
705054bc3f5b: Pull complete
c7051e069564: Pull complete
7308e914506c: Pull complete
Digest: sha256:945039273a7b927869a07b375dc3148de16865de44dec8398672977e050a072e
Status: Downloaded newer image for ubuntu:latest
```

While in the container, let's write some data to the volume.

root@4655a7fa1db9:/# echo "this is sample data written in container " > /c_mydata/Example.txt

```
root@4655a7fa1db9:/# echo "this is sample data written in  container " > /c_myda
ta/Example.txt
root@4655a7fa1db9:/#
```

Because we used the --rm flag, our container will be automatically deleted when we exit. Our volume, however, will still be accessible.

root@4655a7fa1db9:/# exit

```
root@4655a7fa1db9:/# exit
exit
[ec2-user@ip-172-31-3-162 ~]$
```

We can verify that the volume is present on our system with ***docker volume inspect.***

[ec2-user@ip-172-31-3-162 ~]$ docker volume inspect mydata

```
[ec2-user@ip-172-31-3-162 ~]$ docker volume inspect mydata
[
    {
        "CreatedAt": "2019-01-31T11:15:05Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/mydata/_data",
        "Name": "mydata",
        "Options": {},
        "Scope": "local"
    }
]
```

Next, let's start a new container and attach ***mydata***.

[ec2-user@ip-172-31-3-162 ~]$ docker run --rm -ti -v mydata:/mydata1 ubuntu

Let's Verify the contents

root@a3582d7cc24a:/# cat /mydata1/Example.txt

```
root@a3582d7cc24a:/# cat /mydata1/Example.txt
this is sample data written in  container
```

Exit the container:

```
root@a3582d7cc24a:/# exit
exit
[ec2-user@ip-172-31-3-162 ~]$
```

In this example, we created a volume, attached it to a container, and verified its persistence.

2. Creating a Volume that Persists when the Container is Removed.

In our next example, we'll create a volume at the same time as the container, delete the container, then attach the volume to a new container.

We'll use the docker run command to create a new container using the base Ubuntu image. **-t** will give us a terminal, and **-i** will allow us to interact with it. For clarity, we'll use **--name** to identify the container.

The **-v** flag will allow us to create a new volume, which we'll call **mydata2**. We'll use a colon to separate this name from the path where the volume should be mounted in the container. Finally, we will specify the base Ubuntu image and rely on the default command in the Ubuntu base image's Docker file, bash, to drop us into a shell.

[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --name=Container2 -v mydata2:/c_mydata2 ubuntu

```
[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --name=Container2 -v mydata2:/c_myd
ata2 ubuntu
```

While in the container, we'll write some data to the volume.

Note: The -v flag is very flexible. It can bindmount or name a volume with just a slight adjustment in syntax. If the first argument begins with a / or ~/ you're creating a bindmount. Remove that, and you're naming the volume. For example:

- -v /path:/path/in/container mounts the host directory, /path at the /path/in/container
- -v path:/path/in/container creates a volume named path with no relationship to the host.

root@e6bf91a62f5e:/# echo "this data written into expmale2 file" > /c_mydata2/Example2.txt

```
root@e6bf91a62f5e:/# echo "this data written into expmale2 file" > /c_mydata2/Ex
ample2.txt
root@e6bf91a62f5e:/#
```

root@e6bf91a62f5e:/# cat /c_mydata2/Example2.txt

```
root@e6bf91a62f5e:/# cat /c_mydata2/Example2.txt
this data written into expmale2 file
```

Let's exit the container

```
root@e6bf91a62f5e:/# exit
exit
```

When we restart the container, the volume will mount automatically.

[ec2-user@ip-172-31-3-162 ~]$ docker start –ai Container2

```
[ec2-user@ip-172-31-3-162 ~]$ docker start -ai Container2
root@e6bf91a62f5e:/#
```

Let's verify that the volume has indeed mounted and our data is still in place.

root@e6bf91a62f5e:/# cat /c_mydata2/Example2.txt

```
root@e6bf91a62f5e:/# cat /c_mydata2/Example2.txt
this data written into expmale2 file
```

Finally, let's exit and clean up:

```
root@e6bf91a62f5e:/# exit
exit
[ec2-user@ip-172-31-3-162 ~]$
```

Docker won't let us remove a volume if it's referenced by a container. Let's see what happens when we try:

[ec2-user@ip-172-31-3-162 ~]$ docker volume rm mydata2

The message tells us that the volume is still in use and supplies the long version of the container ID:

Error response from daemon: remove mydata2: volume is in use - [e6bf91a62f5eb88a33a4cdec7cc7a80426b6c3695bf1103959f94260f09ce87c]

```
[ec2-user@ip-172-31-3-162 ~]$ docker volume rm mydata2
Error response from daemon: remove mydata2: volume is in use - [e6bf91a62f5eb88a
33a4cdec7cc7a80426b6c3695bf1103959f94260f09ce87c]
[ec2-user@ip-172-31-3-162 ~]$
```

We can use this ID to remove the container

```
[ec2-user@ip-172-31-3-162 ~]$ docker rm e6bf91a62f5eb88a33a4cdec7cc7a80426b6c369
5bf1103959f94260f09ce87c
e6bf91a62f5eb88a33a4cdec7cc7a80426b6c3695bf1103959f94260f09ce87c
[ec2-user@ip-172-31-3-162 ~]$
```

Removing the container won't affect the volume. We can see it's still present on the system by listing the volumes with ***docker volume ls***

[ec2-user@ip-172-31-3-162 ~]$ docker volume ls

```
[ec2-user@ip-172-31-3-162 ~]$ docker volume ls
DRIVER              VOLUME NAME
local               mydata
local               mydata2
[ec2-user@ip-172-31-3-162 ~]$
```

And we can use ***docker volume rm*** to remove it.

[ec2-user@ip-172-31-3-162 ~]$ docker volume rm mydata2

```
[ec2-user@ip-172-31-3-162 ~]$ docker volume rm mydata2
mydata2
[ec2-user@ip-172-31-3-162 ~]$
```

In this example, we created an empty data volume at the same time that we created a container. In our next example, we'll explore what happens when we create a volume with a container directory that already contains data.

3. Creating a Volume from an Existing Directory with Data

Generally, creating a volume independently with docker volume create and creating one while creating a container are equivalent, with one exception. If we create a volume at the same time that we create a container and we provide the path to a directory that contains data in the base image, that data will be copied into the volume.

As an example, we'll create a container and add the data volume at **/var**, a directory which contains data in the base image.

[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --rm -v mydata3:/var ubuntu

All the content from the base image's **/var** directory is copied into the volume, and we can mount that volume in a new container.

```
[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --rm -v mydata3:/var ubuntu
root@6d3b0587f742:/#
```

Exit the current container:

root@6d3b0587f742:/# exit

```
root@6d3b0587f742:/# exit
exit
[ec2-user@ip-172-31-3-162 ~]$
```

This time, rather than relying on the base image's default bash command, we'll issue our own **ls** command, which will show the contents of the volume without entering the shell.

[ec2-user@ip-172-31-3-162 ~]$ docker run --rm -v mydata3:/c_mydata3 ubuntu ls c_mydata3

```
[ec2-user@ip-172-31-3-162 ~]$ docker run --rm -v mydata3:/c_mydata3 ubuntu ls c_
mydata3
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
```

The directory **c_mydata3** now has a copy of the contents of the base image's **/var** directory.

4. Sharing Data Between Multiple Docker Containers.

So far, we've attached a volume to one container at a time. Often, we'll want multiple containers to attach to the same data volume. This is relatively straightforward to accomplish, but there's one critical caveat: at this time, Docker doesn't handle file locking. If you need multiple containers writing to the volume, the applications running in those containers must be designed to write to shared data stores in order to prevent data corruption.

Create Container4 and mydata4

Use **docker run** to create a new container named **Container4** with a data volume attached:

[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --name=Container4 -v mydata4:/c_mydata4 ubuntu

```
[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --name=Container4 -v mydata4:/c_myd
ata4 ubuntu
```

Next we'll create a file and add some text:

root@0951110e9e64:/# echo "This file is shared between containers" > /c_mydata4/Example4.txt

```
root@0951110e9e64:/# echo "This file is shared between containers" > /c_mydata4/
Example4.txt
```

Then, we'll exit the container:

root@0951110e9e64:/# exit

```
root@0951110e9e64:/# exit
exit
```

This returns us to the host command prompt, where we'll make a new container that mounts the data volume from Container4.

Create Container5 and Mount Volumes from Container4

We're going to create **Container5**, and mount the volumes from **Container4**

[ec2-user@ip-172-31-3-162 ~]$ docker run -ti --name=Container5 --volumes-from Container4 ubuntu

Let's check the data persistence:

root@4ccbfb6b9ff2:/# cat /c_mydata4/Example4.txt

```
root@4ccbfb6b9ff2:/# cat /c_mydata4/Example4.txt
This file is shared between containers
```

Now let's append some text from Container5.

```
root@4ccbfb6b9ff2:/# echo "Both containers can write to DataVolume4" >> /c_mydat
a4/Example4.txt
```