Docker 03: Handling Docker Volumes with Hands-on

Purpose of this tutorial is to explain how to handle volumes in Docker containers.

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

Volumes are easier to back up or migrate than bind mounts.

You can manage volumes using Docker CLI commands or the Docker API.

Volumes work on both Linux and Windows containers.

Volumes can be more safely shared among multiple containers.

Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.

New volumes can have their content pre-populated by a container.

Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

In addition, volumes are often a better choice than persisting data in a container’s writable layer, because a volume does not increase the size of the containers using it, and the volume’s contents exist outside the lifecycle of a given container.

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: ***volumes***, and ***bind mounts***. If you’re running Docker on Linux you can also use a *tmpfs mount*.

* **Volumes** are stored in a part of the host file system which is *managed by Docker* (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the file system. Volumes are the best way to persist data in Docker.
* **Bind mounts** may be stored *anywhere* on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
* **tmpfs mounts** are stored in the host system’s memory only, and are never written to the host system’s file system.

If your container generates non-persistent state data, consider using a tmpfs mount to avoid storing the data anywhere permanently, and to increase the container’s performance by avoiding writing into the container’s writable layer.

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.

Docker volumes, which are special directories in a container, store files in the host machine so that the files are persisted even after the container stops.

Volumes are created and managed by Docker. You can create a volume explicitly using the docker volume create command.

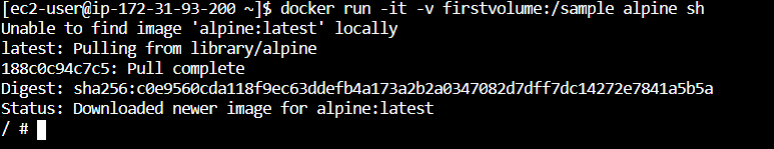


When you create a volume, it is stored within a directory on the Docker host. When you mount the volume into a container, this directory is what is mounted into the container. Look at the Mountpoint.



### **Declaration of volumes**

Volumes can be declared on the command-line, with the --volume or -v flag for docker run. Let's create an alpine container.



**Tip:**

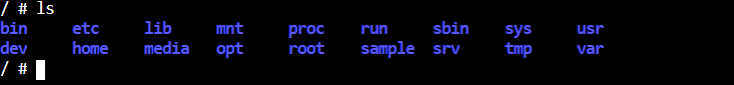
-v or --volume: Consists of three fields, separated by colon characters (:). The fields must be in the correct order.

* the first field is the name of the volume, and is unique on a given host machine. In this example volume name is firstvolume.
* The second field is the path where the file or directory are mounted in the container. In this example folder in container is /sample.
* The third field is optional, and is a comma-separated list of options, such as ro (read only).

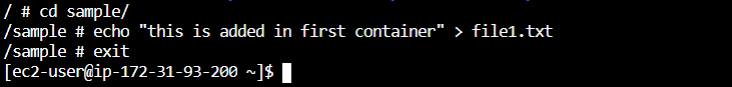
**Alpine:**

* Alpine Linux is an independent, non-commercial, general purpose Linux distribution designed for power users who appreciate security, simplicity and resource efficiency.
* Because of its small size, it is commonly used in containers providing quick boot-up times.

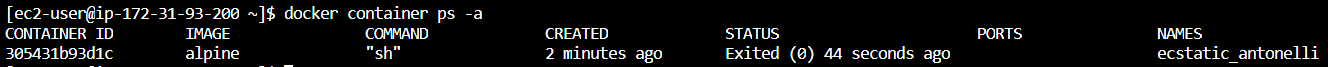
When we type ls command in alpine terminal, we can see the sample folder.



We create a file in the sample folder and exit.



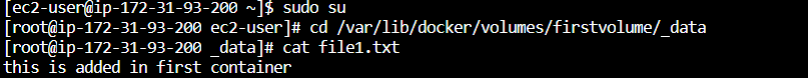
We remove the alpine container.





Let's check the file1.txt.

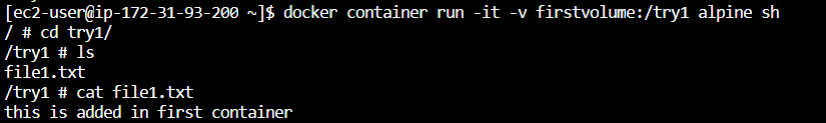




As we see above, file1.txt is still there even if we remove the container.

### **Usage volume with different containers**

Let's run an alpine image and this time we will create try1 folder instead of sample folder.

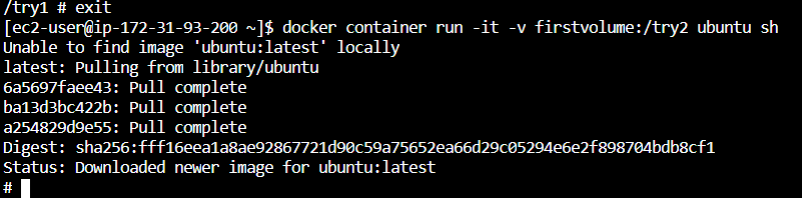


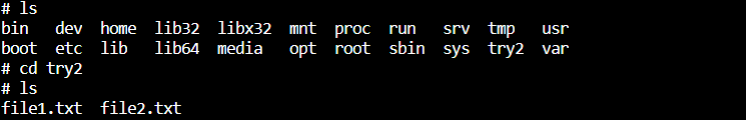
As we see, we can reach file1.txt via a new container.

We can add a new file to the try1 folder.



We create an ubuntu image.

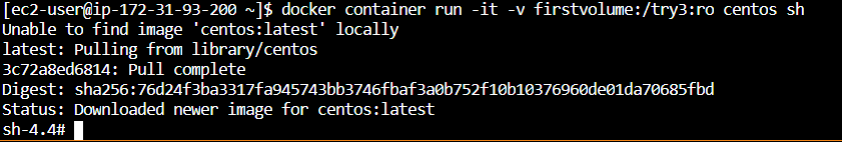


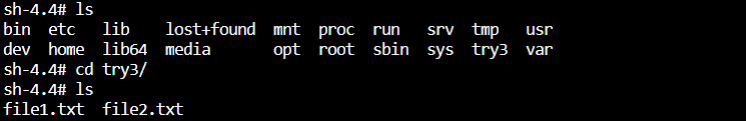


We can use the same volumes with different containers.

### **Use a read-only volume**

For some development applications, the container needs to write into the bind mount so that changes are propagated back to the Docker host. At other times, the container only needs read access to the data. Remember that multiple containers can mount the same volume, and it can be mounted read-write for some of them and read-only for others, at the same time.





Let's try to add a file to the volume.

