This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Docker Volumes

8/20/2025

Table of Contents

Docker Volumes

Understanding Volumes 4

Managing Volumes 8

Search the table of all articles in this guide, listed in alphabetical order, to find the article you need.

Understanding Docker Volumes

Understanding Volumes

Contents

- 1 Choose the -v or --mount flag
- 2 Escape Values from Outer CSV Parser
- 3 Differences between -v and --mount behavior

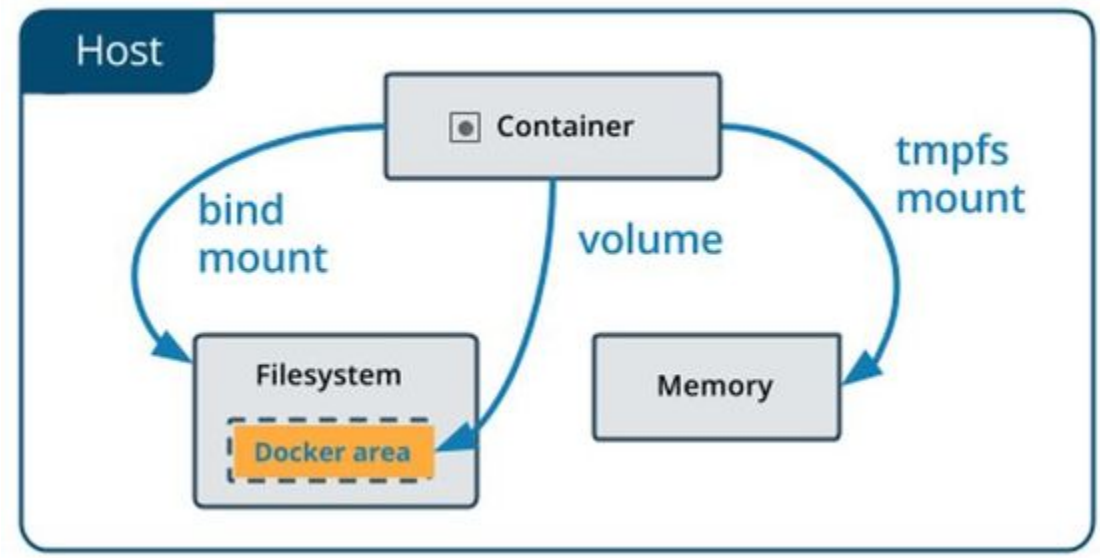
This section explains what docker volumes are all about.

> ## Warning
>
> The following content has been deprecated and is maintained for reference only.

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure 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.

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 life cycle of a given container.



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. Volumes use rprivate bind propagation, and bind propagation is not configurable for volumes.

Choose the -v or --mount flag

Originally, the **-v** or **--volume** flag was used for standalone containers and the **--mount** flag was used for swarm services. However, starting with Docker 17.06, you can also use **--mount** with standalone containers. In general, **--mount** is more explicit and verbose. The biggest difference is that the -v syntax combines all the options together in one field, while the **--mount** syntax separates them. Here is a comparison of the syntax for each flag. New users should try **--mount** syntax which is simpler than **--volume** syntax. If you need to specify volume driver options, you must use **--mount**.

- **-v** or **--volume**: Consists of three fields, separated by colon characters (:). The fields must be in the correct order, and the meaning of each field is not immediately obvious.
  - In the case of named volumes, the first field is the name of the volume, and is unique on a given host machine. For anonymous volumes, the first field is omitted.
  - The second field is the path where the file or directory are mounted in the container.
  - The third field is optional, and is a comma-separated list of options, such as **ro**. These options are discussed below.
- **--mount**: Consists of multiple key-value pairs, separated by commas and each consisting of a **key=value** tuple. The **--mount** syntax is more verbose than **-v** or **--volume**, but the order of the keys is not significant, and the value of the flag is easier to understand.
  - The **type** of the **mount**, which can be **bind**, **volume**, or **tmpfs**. This topic discusses volumes, so the type is always **volume**.
  - The **source** of the mount. For named volumes, this is the name of the volume. For anonymous volumes, this field is omitted. May be specified as **source** or **src**.
  - The **destination** takes as its value the path where the file or directory is mounted in the container. May be specified as **destination**, **dst**, or **target**.
  - The **read-only** option, if present, causes the bind mount to be mounted into the container as read-only.
  - The volume-opt option, which can be specified more than once, takes a key-value pair consisting of the option name and its value.

Escape Values from Outer CSV Parser

If your volume driver accepts a comma-separated list as an option, you must escape the value from the outer CSV parser. To escape a volume-opt, surround it with double quotes (") and surround the entire mount parameter with single quotes (').

For example, the local driver accepts mount options as a comma-separated list in the o parameter. This example shows the correct way to escape the list.

```
$ docker service create \
     --mount 'type=volume,src=,dst=,volume-driver=local,volume-opt=type=nfs,volume-
opt=device=:,"volume-opt=o=addr=,vers=4,soft,timeo=180,bg,tcp,rw"'
    --name myservice \
```

The examples show that both **--mount** syntax and **-v** syntax are possible, and that **--mount** is presented first.

Differences between -v and --mount behavior

As opposed to bind mounts, all options for volumes are available for both **--mount** and **-v** flags.When using volumes with services, only **--mount** is supported.

Managing Volumes

Contents

- 1 Create and Manage Volumes
- 2 Using a Volume Driver
- 3 Create a Volume using a Volume Driver
- 4 Start a Container Which Creates a Volume Using a Volume Driver
- 5 Backup, Restore, or Migrate Data Volumes
  - 5.1 Backup a Container
  - 5.2 Restore Container from Backup
  - 5.3 Remove Volumes
  - 5.4 Remove Anonymous Volumes
  - 5.5 Remove All Volumes

This section explains how you can create and manage volumes outside the scope of any container.

> ## Warning
>
> The following content has been deprecated and is maintained for reference only.

Create and Manage Volumes

Note: At the end of this topic, you will be provided with a terminal to an environment that has all the prerequisites (such as Docker and Kubernetes) up and running. You can practice your commands in this tutorial without any need to setup your own environment.

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

Create a Volume

```
$ docker volume create my-vol
```

List Volumes

```
$ docker volume ls

local               my-vol
```

Inspect Volumes

```
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

Remove a Volume

```
$ docker volume rm my-vol
```

Start a Service with Volumes

When you start a service and define a volume, each service container uses its own local volume. None of the containers can share this data if you use the local volume driver. However, some volume drivers do support shared storage. Docker for AWS and Docker for Azure both support persistent storage using the Cloudstor plugin.

The following example starts a **nginx** service with four replicas, each of which uses a local volume called "myvol2**.**

```
$ docker service create -d \
  --replicas=4 \
  --name devtest-service \
  --mount source=myvol2,target=/app \
  nginx:latest
```

Use docker service **ps devtest-service** to verify that the service is running:

```
$ docker service ps devtest-service

ID                  NAME                IMAGE               NODE                DESIRED
STATE       CURRENT STATE            ERROR               PORTS
4d7oz1j85wwn        devtest-service.1   nginx:latest        moby
Running             Running 14 seconds ago
```

Remove the service to stop all its tasks:

```
$ docker service rm devtest-service
```

Removing the service does not remove volumes created by the service.

Using a Volume Driver

When you create a volume using docker volume create, or when you start a container which uses a **not-yet-created** volume, you can specify a volume driver. The following examples use the **vieux/sshfs** volume driver, first when creating a standalone volume, and then when starting a container which creates a new volume.

Initial set-up

This example assumes that you have two nodes, the first of which is a Docker host and can connect to the second using SSH.

On the Docker host, install the vieux/sshfs plugin:

```
$ docker plugin install --grant-all-permissions vieux/sshfs
```

Create a Volume using a Volume Driver

This example specifies a SSH password, but if the two hosts have shared keys configured, you can omit the password. Each volume driver may have zero or more configurable options, each of which is specified using an -o flag.

```
$ docker volume create --driver vieux/sshfs \
  -o sshcmd=test@node2:/home/test \
  -o password=testpassword \
  sshvolume
```

Start a Container Which Creates a Volume Using a Volume Driver

This example specifies a SSH password, but if the two hosts have shared keys configured, you can omit the password. Each volume driver may have zero or more configurable options. If the volume driver requires you to pass options, you must use the --mount flag to mount the volume, rather than -v.

```
$ docker run -d \
  --name sshfs-container \
  --volume-driver vieux/sshfs \
  --mount src=sshvolume,target=/app,volume-opt=sshcmd=test@node2:/home/test,volume-
opt=password=testpassword \
  nginx:latest
```

Backup, Restore, or Migrate Data Volumes

Volumes are useful for backups, restores, and migrations. Use the --volumes-from flag to create a new container that mounts that volume.

Backup a Container

For example, in the next command, we:

- Launch a new container and mount the volume from the dbstore container
- Mount a local host directory as /backup
- Pass a command that tars the contents of the dbdata volume to a backup.tar file inside our /backup directory.

```
$ docker run --rm --volumes-from dbstore -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar
/dbdata
```

When the command completes and the container stops, we are left with a backup of our dbdata volume.

Restore Container from Backup

With the backup just created, you can restore it to the same container, or another that you made elsewhere.

For example, create a new container named dbstore2:

```
$ docker run -v /dbdata --name dbstore2 ubuntu /bin/bash
Then un-tar the backup file in the new container`s data volume:
$ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar
xvf /backup/backup.tar --strip 1"
```

You can use the techniques above to automate backup, migration and restore testing using your preferred tools.

Remove Volumes

A Docker data volume persists after a container is deleted. There are two types of volumes to consider:

- Named volumes have a specific source form outside the container, for example awesome:/bar.
- Anonymous volumes have no specific source so when the container is deleted, instruct the Docker Engine daemon to remove them.

Remove Anonymous Volumes

To automatically remove anonymous volumes, use the --rm option. For example, this command creates an anonymous /foovolume. When the container is removed, the Docker Engine removes the /foo volume but not the awesome volume.

```
$ docker run --rm -v /foo -v awesome:/bar busybox top
```

Remove All Volumes

To remove all unused volumes and free up space:

```
$ docker volume prune
```

```
$ docker volume prune
```

You can practice the above-mentioned commands using the following widget: