Docker Volumes

Managing Volumes

12/18/2025

Contents

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: