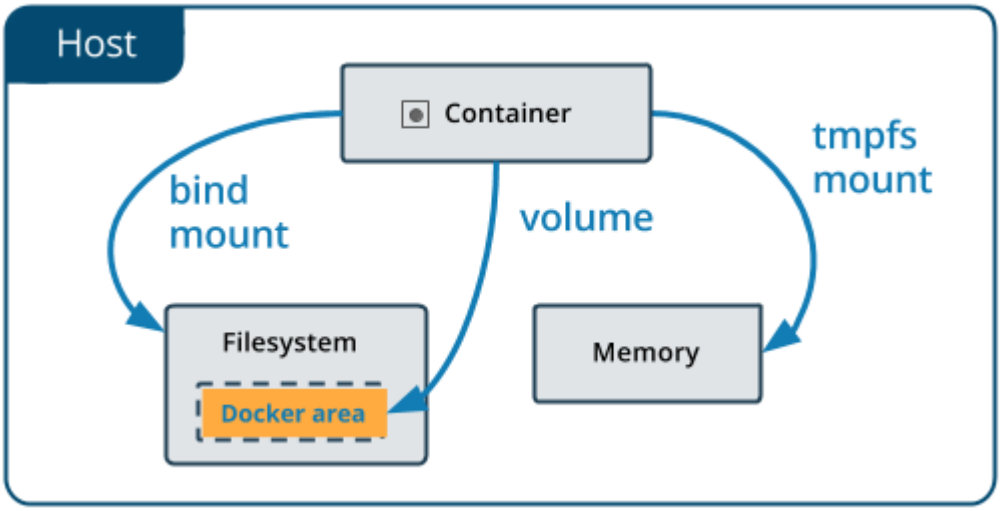
Using volumes

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 lifecycle 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 `readonly` 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=<VOLUME-NAME>,dst=<CONTAINER-PATH>,volume-
> driver=local,volume-opt=type=nfs,volume-opt=device=<nfs-server>:<nfs-
> path>,"volume-opt=o=addr=<nfs-address>,vers=4,soft,timeo=180,bg,tcp,rw"'
>     --name myservice \
>     <IMAGE>
> ```
>
> {: .warning}

The examples below show both the `--mount` and `-v` syntax where possible, and `--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.

Create and manage volumes

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 a volume:

```
$ 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 container with a volume

If you start a container with a volume that does not yet exist, Docker creates the volume for you. The following example mounts the volume `myvol2` into `/app/` in the container.

The `-v` and `--mount` examples below produce the same result. You can't run them both unless you remove the `devtest` container and the `myvol2` volume after running the first one.

- mount
- -v

- mount

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2,target=/app \
  nginx:latest
```

- -v

```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```

Use `docker inspect devtest` to verify that the volume was created and mounted correctly. Look for the `Mounts` section:

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "myvol2",
        "Source": "/var/lib/docker/volumes/myvol2/_data",
        "Destination": "/app",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

This shows that the mount is a volume, it shows the correct source and destination, and that the mount is read-write.

Stop the container and remove the volume. Note volume removal is a separate step.

```
$ docker container stop devtest

$ docker container rm devtest

$ docker volume rm myvol2
```

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, but 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, which stops all its tasks:

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

Removing the service does not remove any volumes created by the service. Volume removal is a separate step.

Syntax differences for services

The `docker service create` command does not support the `-v` or `--volume` flag. When mounting a volume into a service's containers, you must use the `--mount` flag.

Populate a volume using a container

If you start a container which creates a new volume, as above, and the container has files or directories in the directory to be mounted (such as `/app/` above), the directory's contents are copied into the volume. The container then mounts and uses the volume, and other containers which use the volume also have access to the pre-populated content.

To illustrate this, this example starts an `nginx` container and populates the new volume `nginx-vol` with the contents of the container's `/usr/share/nginx/html` directory, which is where Nginx stores its default HTML content.

The `--mount` and `-v` examples have the same end result.

- mount
- -v

- mount

```
$ docker run -d \
  --name=nginxtest \
  --mount source=nginx-vol,destination=/usr/share/nginx/html \
  nginx:latest
```

- -v

```
$ docker run -d \
  --name=nginxtest \
  -v nginx-vol:/usr/share/nginx/html \
  nginx:latest
```

After running either of these examples, run the following commands to clean up the containers and volumes. Note volume removal is a separate step.

```
$ docker container stop nginxtest

$ docker container rm nginxtest

$ docker volume rm nginx-vol
```

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.

This example modifies the one above but mounts the directory as a read-only volume, by adding `ro` to the (empty by default) list of options, after the mount point within the container. Where multiple options are present, separate them by commas.

The `--mount` and `-v` examples have the same result.

- mount
- -v

- mount

```
$ docker run -d \
  --name=nginxtest \
  --mount source=nginx-vol,destination=/usr/share/nginx/html,readonly \
  nginx:latest
```

- -v

```
$ docker run -d \
  --name=nginxtest \
  -v nginx-vol:/usr/share/nginx/html:ro \
  nginx:latest
```

Use `docker inspect nginxtest` to verify that the readonly mount was created correctly. Look for the `Mounts` section:

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "nginx-vol",
        "Source": "/var/lib/docker/volumes/nginx-vol/_data",
        "Destination": "/usr/share/nginx/html",
        "Driver": "local",
        "Mode": "",
        "RW": false,
        "Propagation": ""
    }
],
```

Stop and remove the container, and remove the volume. Volume removal is a separate step.

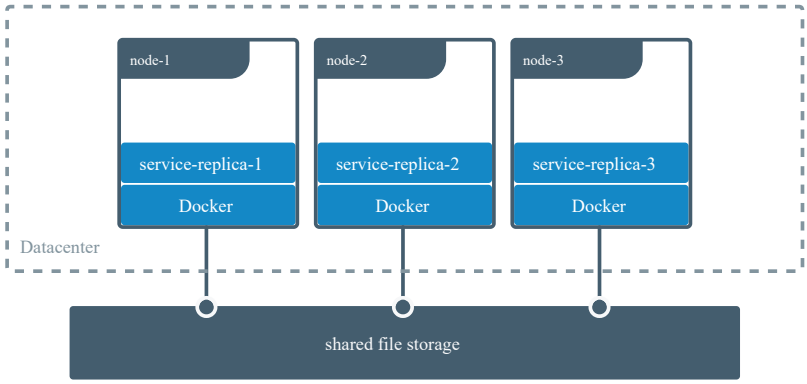
```
$ docker container stop nginxtest

$ docker container rm nginxtest

$ docker volume rm nginx-vol
```

Share data among machines

When building fault-tolerant applications, you might need to configure multiple replicas of the same service to have access to the same files.



There are several ways to achieve this when developing your applications. One is to add logic to your application to store files on a cloud object storage system like Amazon S3. Another is to create volumes with a driver that supports writing files to an external storage system like NFS or Amazon S3.

Volume drivers allow you to abstract the underlying storage system from the application logic. For example, if your services use a volume with an NFS driver, you can update the services to use a different driver, as an example to store data in the cloud, without changing the application logic.

Use 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
```

Create a service which creates an NFS volume

This example shows how you can create an NFS volume when creating a service. This example uses `10.0.0.10` as the NFS server and `/var/docker-nfs` as the exported directory on the NFS server. Note that the volume driver specified is `local`.

NFSv3

```
$ docker service create -d \
  --name nfs-service \
  --mount 'type=volume,source=nfsvolume,target=/app,volume-driver=local,volume-
opt=type=nfs,volume-opt=device=:/var/docker-nfs,volume-opt=o=addr=10.0.0.10' \
  nginx:latest
```

NFSv4

```
docker service create -d \
    --name nfs-service \
    --mount 'type=volume,source=nfsvolume,target=/app,volume-driver=local,volume-
opt=type=nfs,volume-opt=device=:/var/docker-nfs,"volume-
opt=o=10.0.0.10,rw,nfsvers=4,async"' \
    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, create a new container named `dbstore`:

```
$ docker run -v /dbdata --name dbstore ubuntu /bin/bash
```

Then 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 from 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 `/foo` volume. 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
```