**Volumes:**

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. 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. 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.

**Uses of volumes:**

Volumes are the preferred way to persist data in Docker containers and services. Some use cases for volumes include:

1. Sharing data among multiple running containers. If you don’t explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.
2. When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you decouple the configuration of the Docker host from the container runtime.
3. When you want to store your container’s data on a remote host or a cloud provider, rather than locally.
4. When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice. You can stop containers using the volume, then back up the volume’s directory (such as /var/lib/docker/volumes/<volume-name>).

**Uses of bind mounts:**

In general, you should use volumes where possible. Bind mounts are appropriate for the following types of use case:

1. Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting /etc/resolv.conf from the host machine into each container.
2. Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.
3. If you use Docker for development this way, your production Dockerfile would copy the production-ready artifacts directly into the image, rather than relying on a bind mount.
4. When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

**Create a volume:**

Command: **docker volume create <vol-name>**

**List Volumes:**

Command: **docker volume ls**

**Remove volume:**

Command**: docker Volume rm <vol-name>**

**Inspect a volume:**

Command: **docker volume inspect <vol-name>**

**Start Container with volume:**

**Command:** **docker run -d \**

**--name <container\_name>\**

**-v <vol\_name>:/app \**

**nginx:latest**

**Stop the container and remove the volume:**

Command: **docker container stop devtest**

**docker container rm devtest**

**docker volume rm myvol2**

**Start a service with volumes:**

Command: **docker service create -d \**

**--replicas=4 \**

**--name devtest-service \**

**--mount source=<volume\_name>,target=/app \**

**nginx:latest**

**Check if the service is working or not:**

**Command: docker service ps devtest-service**

**Remove Service:**

Command: **docker service rm devtest-service**

**Remove All Volumes:**

Command: **docker volume prune**

**Populate volume Using Container:**

Command**: docker run -d \**

**--name=nginxtest \**

**-v nginx-vol:/usr/share/nginx/html \**

**nginx:latest**

**Use a read-only volume:**

Command: **docker run -d \**

**--name=nginxtest \**

**-v nginx-vol:/usr/share/nginx/html:ro \**

**nginx:latest**