Manage data in Docker

By default all files created inside a container are stored on a writable container layer. This means that:

**1. Data Doesn't Persist After the Container is Gone:**

* **Explanation**: When you create files or store data inside a Docker container, by default, that data is saved in a special writable layer that is part of the container. However, this data is temporary. If the container is deleted or stopped, all the data in this layer is also lost. Additionally, if you need to access the data from outside the container (for example, from another container or process), it can be challenging because the data is stuck inside that specific container's layer.

**2. Data is Tied to the Host Machine:**

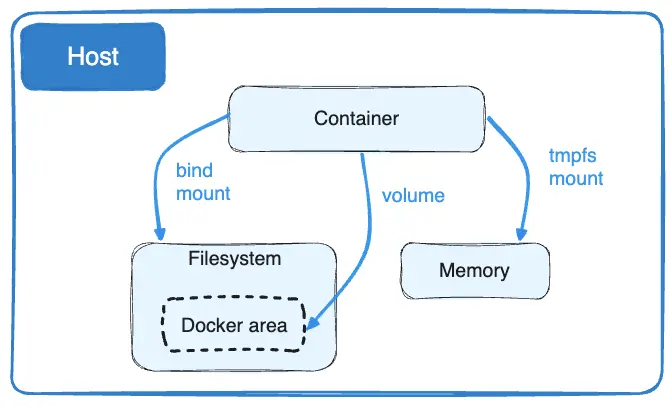
* **Explanation**: The writable layer where the container stores its data is closely linked to the host machine (the computer or server) where the container is running. This means the data is not easily transferable. If you move the container to a different host machine, the data won't move with it. This makes it hard to share or backup the data if you need to move the container around.
* Iska mtlb yeh haka data itna tightly close/linked hota hai container and host machine ka darmayan kay jasa hi container remove hota hai wo layer be lost hojati hai.

**3. Performance Impact Due to Storage Driver:**

* **Explanation**: To write data into the container’s writable layer, Docker uses a special software component called a "storage driver." This driver manages how the data is stored on the filesystem, using something called a "union filesystem" provided by the Linux kernel. However, because this storage driver adds an extra layer of management and complexity, it can slow down the performance of data writing and reading compared to using Docker volumes, which directly access the host machine’s filesystem without this extra layer.
* Iska mtlb hai basically within container jo layers kay darmayan data sharing hoti hai that is done by **Storage driver** so agar isi mechanism ka through data ko different containers ka darmayan be share kreinga toh slow process hoga , isi lia Data accessing through direct host filesystem is much good than it and that is done by volume.

Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.

Docker also supports containers storing files in-memory on the host machine. Such files are not persisted. If you're running Docker on Linux, tmpfs mount is used to store files in the host's system memory. If you're running Docker on Windows, named pipe is used to store files in the host's system memory.



1. Volumes are stored in a part of the host filesystem which is *managed by Docker* (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
   * Iska mtlb haka Volume host machine may asi jaga store hota hai which can be only managed by Docker, otherwise koi be asa process/program jo Docker related nhi hai wo usko modify nhi krskta.
2. 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.
   * In Bind mount host system pa kahi be mounting hojati hai that can be modify later on.
3. tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.
   * Iska toh simply mtlb yeh haka agar yeh mounting krtay hain toh wo RAM pa data store hoga which is not persistent. This type of mounting is done to store temporary info like sensitive keys etc.

**In detail**

[**Volumes**](https://docs.docker.com/engine/storage/#volumes)

Volumes are created and managed by Docker. You can create a volume explicitly using the docker volume create command, or Docker can create a volume during container or service creation.

When you create a volume, it's stored within a directory on the Docker host. When you mount the volume into a container, this directory is what's mounted into the container. This is similar to the way that bind mounts work, except that volumes are managed by Docker and are isolated from the core functionality of the host machine.

A given volume can be mounted into multiple containers simultaneously. When no running container is using a volume, the volume is still available to Docker and isn't removed automatically. You can remove unused volumes using docker volume prune.

When you mount a volume, it may be named or anonymous. Anonymous volumes are given a random name that's guaranteed to be unique within a given Docker host. Just like named volumes, anonymous volumes persist even if you remove the container that uses them, except if you use the --rm flag when creating the container, in which case the anonymous volume is destroyed. See [Remove anonymous volumes](https://docs.docker.com/engine/storage/volumes/#remove-anonymous-volumes). If you create multiple containers after each other that use anonymous volumes, each container creates its own volume. Anonymous volumes aren't reused or shared between containers automatically. To share an anonymous volume between two or more containers, you must mount the anonymous volume using the random volume ID.

A/c to above explanation its mean that jo anonymous volume hotay hain wo different containers ka darmayan share nhi hosktay , but jo named volume hain wo different volumes ka darmayan share hosktay hain.

Volumes also support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers, among other possibilities.

**Bind Mount**

* **Definition**: Bind mounts map a specific directory or file on the host machine directly into a container. The path is defined explicitly by the user.
* **Use Case**: Bind mounts are useful when you need direct access to files on the host machine, such as during development when you want to edit code on the host and have those changes immediately reflected in the container.
* **Advantages**:
  + Full control over the exact location of files on the host system.
  + Useful for accessing specific host directories and files.

One important fact that we should consider before using **bind mount.**

One side effect of using bind mounts is that you can change the host filesystem via processes running in a container, including creating, modifying, or deleting important system files or directories. This is a powerful ability which can have security implications, including impacting non-Docker processes on the host system.

* + Iska mtlb yeh hai kay we use Bind mount in Docker, so basically jis file or folder ko hum apni host machine say link krwatay hain container may, Toh jis tarah agar hum apni uss local file or folder may changes krtay hain toh that will reflect in container, so isi tarah jab hum container may be kuch change kreinga toh it will reflect in local system , which means kay agar kisi nay container say koi file may error inject krdia ya etc toh it can affect our files at host machine as well.

### [**tmpfs**](https://docs.docker.com/engine/storage/#tmpfs)

A tmpfs mount isn't persisted on disk, either on the Docker host or within a container. It can be used by a container during the lifetime of the container, to store non-persistent state or sensitive information. For instance, internally, Swarm services use tmpfs mounts to mount [secrets](https://docs.docker.com/engine/swarm/secrets/) into a service's containers.

Now basically we have two term use which are Named Volumes and Named Pipe:

**Understanding Named Pipes and Named Volumes:**

**Named Pipes:**

* **What They Are**: Named pipes are a method of inter-process communication (IPC) that allows data to be exchanged between processes. In the context of Docker, a named pipe is a special type of file that acts as a conduit for communication between the Docker host and a container.
* **Use Case**: Named pipes are commonly used to allow a container to interact with the Docker Engine API directly. For example, you might run a third-party tool inside a container that needs to communicate with the Docker Engine on the host machine. The named pipe provides a way for this communication to happen securely and efficiently.
* **Example**: On Windows, Docker uses named pipes (e.g., \\.\pipe\docker\_engine) to allow containers to communicate with the Docker daemon. Its mean in the context of Docker **named pipe** are used to make communication b/w containers and docker daemon.

**Named Volumes:**

* **What They Are**: Named volumes in Docker are a way to persist data that is generated and used by Docker containers. Unlike bind mounts, which link to a specific directory on the host, named volumes are managed by Docker and stored in Docker’s volume directory on the host machine.
* **Use Case**: Named volumes are typically used to store persistent data that needs to be shared between containers or preserved across container restarts. They abstract away the underlying storage, making it easier to manage and use persistent data.
* **Example**: If you create a named volume using docker volume create my\_volume, Docker will manage the storage location, and you can mount it in a container using -v my\_volume:/container/path.

[**Good use cases for volumes**](https://docs.docker.com/engine/storage/#good-use-cases-for-volumes)

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

* 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.
* 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. Iska mtlb haka later on jab container different machine pay jayega for testing, deployment toh jo folder hai wagera hai wo thoray bht changes aa sktay hain to uss cheex ko prevent krnay ka liya Volume ko use krtay hain.
* When you want to store your container's data on a remote host or a cloud provider, rather than locally.
* 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>).
* When your application requires high-performance I/O on Docker Desktop. Volumes are stored in the Linux VM rather than the host, which means that the reads and writes have much lower latency and higher throughput.
* When your application requires fully native file system behavior on Docker Desktop. For example, a database engine requires precise control over disk flushing to guarantee transaction durability. Volumes are stored in the Linux VM and can make these guarantees, whereas bind mounts are remoted to macOS or Windows, where the file systems behave slightly differently.

[Good use cases for bind mounts](https://docs.docker.com/engine/storage/#good-use-cases-for-bind-mounts)

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

* 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.
* 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.

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.

* When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.

[**Good use cases for tmpfs mounts**](https://docs.docker.com/engine/storage/#good-use-cases-for-tmpfs-mounts)

tmpfs mounts are best used for cases when you do not want the data to persist either on the host machine or within the container. This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.