Storage

All files created inside a container are stored on a writable layer by default. Which means that:

- The data does not persist when that container no longer exists
- It is challenging to get the data out of the container if another process needs it.
- Performance is impacted because the writable layer requires a kernel driver to manage the filesystem.

Docker has four options to store files with data persistency in the host machine:

- **Volumes** are stored in a part of the host filesystem, which is managed by Docker (on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.tmp.
    - Sharing data among multiple running containers can be using the same volume simultaneously.
    - Volumes are only removed when you explicitly remove them, ensuring data retention.
    - To keep using the same information in different containers.
    - 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.
    - When your application requires high-performance I/O. 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.
- **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.
    - Sharing configuration files from the host machine to containers.
    - Sharing source code between a development environment on the Docker host and a container.
    - When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require.
- **tmpfs mounts** are available only if you are hosting in Linux, stored in the host system's memory, and never written to the host system's filesystem.
    - 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.
- **named pipes**: An `npipe` mount can be used only in Windows hosts. The typical use case is to run a third-party tool inside of a container and connect to the Docker Engine API using a named pipe.

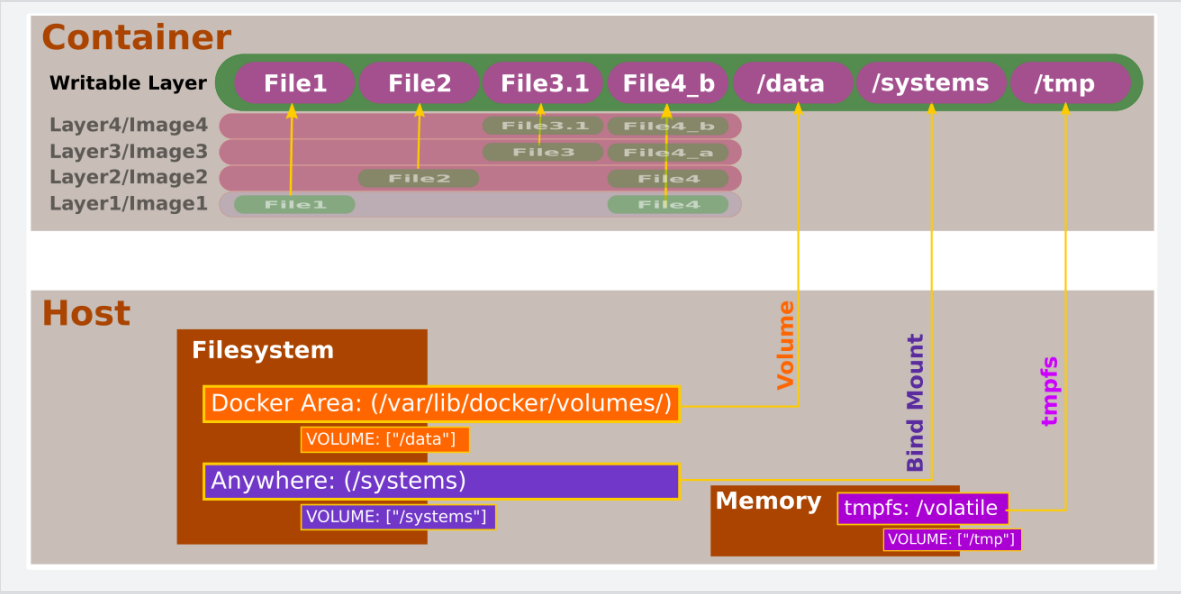

Fig. 1 - How the local host shares a filesystem and is mounted inside a container.

No matter which type of mount you choose to use, the data looks the same from within the container. Either as a file or a directory.

Volume mounts

Volumes are preferred to manage storage in containers because Docker fully manages them. You can create a volume explicitly using the following command or during container or service creation:

```
$ docker volume create <VOLUME-NAME>
```

When you create a volume, it is stored within a directory on the Docker host. (`/var/lib/docker/volumes/`)

You can list your volumes:

```
$ docker volume ls
DRIVER    VOLUME NAME
local     17bd068350750355781a2d2ff392512ff7a860cec3fcb05
local     45f28929947948aaa21f471e17a098ac34d00248490a7f6
local     79b0f357817f633ed0df5e1eb87a779205ed05789d9aeb2
```

And you can see the details:

```
$ docker volume inspect
17bd066f8bef966e4beedbd8350750355781a2d2ff392512ff7a860cec3fcb05
[
    {
        "CreatedAt": "2020-04-08T15:23:34-05:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint":
"/var/lib/docker/volumes/17bd068350750355781a2d2ff392512ff7a860cec3fcb05
/_data",
        "Name": "17bd068350750355781a2d2ff392512ff7a860cec3fcb05",
        "Options": null,
        "Scope": "local"
    }
]
```

When you mount the volume into a container, this directory is mounted into the container.

You can remove unused volumes:

```
$  docker volume prune
WARNING! This will remove all local volumes not used by at least one
container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
17bd066f8bef966e4beedbd8350750355781a2d2ff392512ff7a860cec3fcb05
79b0f357817f63adafe8996c8ac2e683ed0df5e1eb87a779205ed05789d9aeb2
45f289299479481cb6441103c605983aaa21f471e17a098ac34d00248490a7f6
aa246f643e27692ca018e69acd3e20cb4a8de937a3b8449a383ff322272cf77e
```

When you mount a volume, it may be **named** or **anonymous** and you have two options to mount:

```
--volume, -v: Combines all in a single line. Options separated by colon
(:).
        - Name of the volume
        - Mounting Path
        - Comma separated list of options.
--mount: Multiple key:pairs separated by commas.
        - type: volume
        - source or src: Name of the volume
        - destination, dst or target: Mounting path
        - readonly: if present mounts in read-only mode
        - volume-opt: Key-value pairs of the options.
        - volume-driver: local, etc..
$ 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>
```

To start a container with a volume:

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

To use it as read only:

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

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 mounts

Bind mounts have limited functionality compared to volumes:

- When you use a bind mount, a file or directory on the *host machine* is mounted into a container.
- The file or directory is referenced by its full path on the host machine.
- The file or directory does not need to exist on the Docker host already.
- It is created on-demand if it does not yet exist.
- Bind mounts are remarkably performant, but they rely on the host machine's filesystem having a specific directory structure available.
- You cannot use Docker CLI commands to manage bind mounts directly.

> Bind mounts allow access to sensitive files
>
> One side effect of using bind mounts, for better or for worse, 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 powerful ability can have security implications, including impacting non-Docker processes on the host system.

You have two options to mount:

```
--volume, -v: Combines all in a single line. Options separated by colon
(:).
        - Directory on the host machine
        - Mounting Path
        - Comma separated list of options.
--mount: Multiple key:pairs separated by commas.
        - type: bind
        - source or src: Directory on the host machine
        - destination, dst or target: Mounting path
        - readonly: if present mounts in read-only mode
        - bind-propagation: Changes the bind propagation. [rprivate,
private, rshared, shared, rslave, slave].
```

To start a container:

```
$ docker run -d -it --name devtest --mount
type=bind,source="$(pwd)"/target,target=/app nginx:latest
```

Use `docker inspect devtest` to verify that the bind mount was created correctly.

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/tmp/source/target",
        "Destination": "/app",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

The above code shows that the mount is a `bind` mount, it shows the correct source and destination, it shows that the mount is read-write and that the propagation is set to `rprivate.`

If you bind-mount into a non-empty directory on the container, the directory's existing contents are obscured by the bind mount.

tmpfs mounts

A `tmpfs` mount is not persisted on disk, either on the Docker host or within a container. A container can use it during the container's lifetime to store non-persistent state or sensitive information.

- you cannot share `tmpfs` mounts between containers.
- This functionality is only available if you are running Docker on Linux.

You have two options to mount:

```
--tmpfs: Does not allow to specify any configurable options and can only be used with standalone containers
--mount: Multiple key:pairs separated by commas.
        - type: tmpfs
        - destination, dst or target: Mounting path
        - tmpfs-size: Size of the tmpfs mount in bytes. Unlimited by default.
        - tmpfs-mode: File mode of the tmpfs in octal. For instance, 700 or 0770. Defaults to 1777 or world-writable.
```

To start a container:

```
$ docker run -d -it --name tmptest --mount type=tmpfs,destination=/app nginx:latest
```

The following example sets the `tmpfs-mode` to `1770`, so that it is not world-readable within the container.

```
$ docker run -d -it --name tmptest --mount type=tmpfs,destination=/app,tmpfs-mode=1770 nginx:latest
```