Data & Volumes

Images are read-only - once they're created, they can't change (you have to rebuild them to update them).

Containers on the other hand can read and write - they add a thin "**read-write layer**" on top of the image. That means that they can make changes to the files and folders in the image without actually changing the image.

But even with read-write Containers, **two big problems** occur in many applications using Docker:

1. **Data written in a Container doesn't persist**: If the Container is stopped and removed, all data written in the Container is lost
2. **The container Container can't interact with the host filesystem**: If you change something in your host project folder, those changes are not reflected in the running container. You need to rebuild the image (which copies the folders) and start a new container

Problem 1 can be solved with a Docker feature called "**Volumes**". **Problem 2** can be solved by using "**Bind Mounts**".

Volumes

Volumes are folders (and files) managed on your host machine which are connected to folders / files inside of a container.

There are **two types of Volumes:**

- **Anonymous Volumes**: Created via `-v /some/path/in/container` and **removed automatically** when a container is removed because of `--rm` added on the `docker run` command
- **Named Volumes**: Created via `-v some-name:/some/path/in/container` and **NOT removed** automatically

With Volumes, **data can be passed into a container** (if the folder on the host machine is not empty) and it can be saved when written by a container (changes made by the container are reflected on your host machine).

Volumes are created and managed by Docker - as a developer, you don't necessarily know where exactly the folders are stored on your host machine. Because the data stored in there is **not meant to be viewed or edited by you** - use "Bind Mounts" if you need to do that!

Instead, especially **Named Volumes** can help you with **persisting data**.

Since data is not just written in the container but also on your host machine, the **data survives even if a container is removed** (because the Named Volume isn't removed in that case). Hence you can use Named Volumes to persist container data (e.g. log files, uploaded files, database files etc)-

Anonymous Volumes can be useful for ensuring that some Container-internal folder is **not overwritten** by a "Bind Mount" for example.

By default, **Anonymous Volumes are removed** if the Container was started with the `--rm` option and was stopped thereafter. They are **not removed** if a Container was started (and then removed) without that option.

Named Volumes are never removed, you need to do that manually (via `docker volume rm VOL_NAME`, see reference below).

Bind Mounts

Bind Mounts are very similiar to Volumes - the key difference is, that you, the developer, **set the path on your host machine** that should be connected to some path inside of a Container.

You do that via `-v /absolute/path/on/your/host/machine:/some/path/inside/of/container`.

The path in front of the `:` (i.e. the path on your host machine, to the folder that should be shared with the container) has to be an absolute path when using `-v` on the `docker run` command.

Bind Mounts are very useful for **sharing data with a Container** which might change whilst the container is running - e.g. your source code that you want to share with the Container running your development environment.

Don't use Bind Mounts if you just want to persist data - Named Volumes should be used for that (exception: You want to be able to inspect the data written during development).

In general, **Bind Mounts are a great tool during development** - they're not meant to be used in production (since you're container should run isolated from it's host machine).

Key Docker Commands

- `docker run -v /path/in/container IMAGE` : Create an **Anonymous Volume** inside a Container
- `docker run -v some-name:/path/in/container IMAGE` : Create a **Named Volume** (named `some-name`) inside a Container
- `docker run -v /path/on/your/host/machine:path/in/container IMAGE` : Create a **Bind Mount** and connect a local path on your host machine to some path in the Container
- `docker volume ls` : **List all currently active / stored Volumes** (by all Containers)
- `docker volume create VOL_NAME` : **Create a new (Named) Volume** named `VOL_NAME`. You typically don't need to do that, since Docker creates them automatically for you if they don't exist when running a container
- `docker volume rm VOL_NAME` : **Remove a Volume** by it's name (or ID)
- `docker volume prune` : **Remove all unused Volumes** (i.e. not connected to a currently running or stopped container)