Containers lifetime and Persistent Data.

1. Containers are meant to be immutable and ephemeral.
2. This is an ideal scenario , but what about databases , or unique Data .?
3. Docker gives us the feature to update an application by re-creating a new container with an updated application and preserves the data for us.
4. Containers we have been using so far are persistent, i.e if we stop the container and restart the container, the data is not lost. Data is only lost when we remove the container and create a new container for the same.
5. The solution docker provides us two solutions to this problem
   1. Data Volumes
   2. Bind Mounts

**Volumes** : This is a concept of creating a file location outside the container.

**Bind Mounts** : This is a concept of binding a specific file location to the local file system to a container file system.

**Volumes:**  In order to be able to save (persist) data and also to share data between containers, Docker came up with the concept of volumes. Quite simply, volumes are directories (or files) that are outside of the default Union File System and exist as normal directories and files on the host filesystem.

Volumes is not deleted if containers is removed, it requires and extra step as we are able to achieve a concept os persistent Data

**Method 1:**

**Volume Command in dockerfIle:**

Consider the docker file in current Folder named “Dockerfile”.

We are actually specifying the volume to be created in out local FIle system for container Data to be stored.

For instance once the image of mysql is pulled from dockerhub with command

* **Docker image pull mysql**

And id we inspect the image under config the volume is actually created to store our databases created in container.

Execute the below command and we can see volume created

* **docker image inspect mysql --format "{{.Config.Volumes}}"**

If we run the container of pulled image and inspect the container , under Mounts tab , we can see the Volume and a new Volume id is created which is binded to local file system mentioned in volume tab.

* **Docker container inspect mysql:**

Mount tab of output JSON as below:

"Mounts": [

{

"Type": "volume",

"Name":"a7eb58e83399980fb5e8c3b47f72ff32e2c0e07b69ea31a05092f6b22ef9c3cc"

,”**Source**":"/var/lib/docker/volumes/a7eb58e83399980fb5e8c3b47f72ff32e2c0e07b69ea31a05092f6b22ef9c3cc/\_data",

"**Destination**": "/var/lib/mysql",

"Driver": "local",

"Mode": "",

"RW": true,

"Propagation": ""

}

],

**Here the docker has created a Volume for itself and stores data actually to the location mentioned in Source of local host. And in back ground binds that location to location of container which is mentioned in destination.**

* **Docker volume ls:**

This will return all volumes created. In below format:

**local a7eb58e83399980fb5e8c3b47f72ff32e2c0e07b69ea31a05092f6b22ef9c3cc**

**Method 2:**

We can also name the volume , as to be a more user friendly with -v command while running a container:

docker container run --name mysql -d **-v volume\_name:volume\_location** image\_name

This concept is called named\_vaolume , we are explicitly providing volume name as it more easy to manage.

**Bind Mounting :**

* BInd mapping is just mapping a host file or directory to container file or directory.
* It’s like two pointers ie file system of local host and file system of container pointing to same file location on hard drive.
* Here priority is given to Host i.e if we add a file or do any changes to a file which is already present in local host , that changes are directly reflected on the file system of the container.

**Syntax:**

**docker container run --name mysql -d -v localhost\_path: container\_path image\_name**

If we add any file to our to local host at mentioned location, the same changes are reflected on path of the container.