Docker Volumes:

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container.
- Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

Start a container with a volume

If you start a container with a volume that doesn't yet exist, Docker creates the volume for you. The following example mounts the volume myvol2 into /app/ in the container.

The -v and --mount examples below produce the same result.

- -v volname:/container_dirpath
- --mount source=volname,target=container_dirpath

Docker volumes commands:

mainly used for database (DB) servers(ex:mysql)

1) **Docker volumes create:**

This command used to create volume

[] docker volume create volumename

2) **Docker volume ls:**

It used to list the volumes

[] docker volume ls

3) **docker volume inspect:**

Display detailed information on volume

[] docker volume inspect volumename

4) **docker volume rm:**

Remove one or more volumes

[] docker volume rm volumename

5) **docker volume prune:**

Remove unused local volumes

[] docker volume prune

Docker network:

Docker networking enables a user to link a Docker container to as many networks as requires. Docker Networks are used to provide complete isolation for Docker containers. A user can add containers to more than one network.

Some of the major benefits of using Docker Networking are:

- They share a single operating system and maintain containers in an isolated environment.
- It requires fewer OS instances to run the workload.
- It helps in the fast delivery of software.
- It helps in application portability.

Docker networking commands:

1) Docker network create:

This command used to create network

[] docker network create networkname

[] docker network create –d drivername NWname

[]

2) Docker network ls:

This command used to display all the networks

[] docker network ls

3) Docker network connect:

This command used to connect network and container

[] docker network connect networkname containerid

4) Docker network disconnect:

This command used to disconnect network and container

[] docker network disconnect networkname containerid

5) Docker network rm:

This command used to rm the network

[] docker network rm network

6) Docker network prune:

This command is used to remove unused network

[] docker network prune

7) Docker network inspect:

This command is used to display the detailed information of network

[] docker network inspect networkname

Docker zero network: (no name resolution)

docker0 is a virtual bridge interface created by Docker. It randomly chooses an address and subnet from a private defined range. All the Docker containers are connected to this bridge.

User-defined bridges provide automatic DNS resolution between containers.

Containers on the default bridge network can only access each other by IP addresses, unless you use the `--link` option, which is considered legacy. **On a user-defined bridge network, containers can resolve each other by name or alias.**

- Imagine an application with a web front-end and a database back-end. If you call your containers web and db, the web container can connect to the db container at db., no matter which Docker host the application stack is running on.
- If you run the same application stack on the default bridge network, you need to manually create links between the containers (using the legacy --link flag). These links need to be created in both directions, so you can see this gets complex with more than two containers which need to communicate. Alternatively, you can manipulate the /etc/hosts files within the containers, but this creates problems that are difficult to debug.
- **User-defined bridges provide better isolation**.
- All containers without a --network specified, are attached to the default bridge network. This can be a risk, as unrelated stacks/services/containers are then able to communicate.
- Using a user-defined network provides a scoped network in which only containers attached to that network are able to communicate.

Host:

Id we give host network to container it don’t have I/p , it’ll use our ip or host

[]--network host (to the container)

It will use our host

Null:

No ip to the container

In real-time we don’t use null and host networks.