DEVOPS
Mr. R N RAJU

MANAGE DOCKER NETWORK

MAIL: Rnraju4u@gmail.com NUMBER: +91 9848363431 YOUTUBE: SYSGEEKS

DOCKER NETWORKING

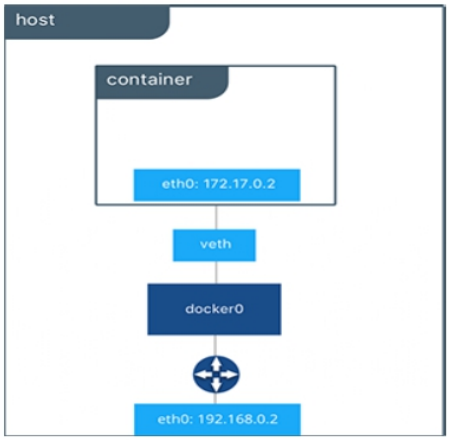
- Docker includes support for networking containers through the use of **network drivers.**
- It is used to establish communication between Containers and outside world via the Docker host machine.
- Docker supports different types of network drivers, each fit for certain use cases.

NETWORK DRIVERS:

- Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:
  - Bridge
  - Host
  - Overlay
  - Ipvlan
  - Macvlan
  - None
  - Network plugins

BRIDGE NETWORK:

- A **Default Bridge Network (Bridge)** is created automatically, when you start docker.
- A bridge network is a Link Layer device which forwards traffic between network segments. A bridge can be a hardware device or a software device running within a host machine's kernel.



NOTE: The relationship between a host and containers is 1:N

To Check default bridge driver:

#ifconfig

#docker network ls

#docker network inspect bridge

Create a New Container:

#docker run -d -it --name Sample1 centos

#docker exec -it Sample1 ip a

#docker network inspect bridge

Create Another New Container

#docker run -d -it --name Sample2 centos

#docker exec -it Sample2 ip a

Testing Network connection:

#docker exec -it Sample2 bash

#ping 172.17.0.2

NOTE: It will communicate each one because both are running on same bridge

To remove container after exit use --rm flag:

#docker run --rm -it --name Sample3 centos bash

To Change Container Hostname:

#docker run -it --name Sample4 --hostname Sample4.example.com centos bash

#hostname

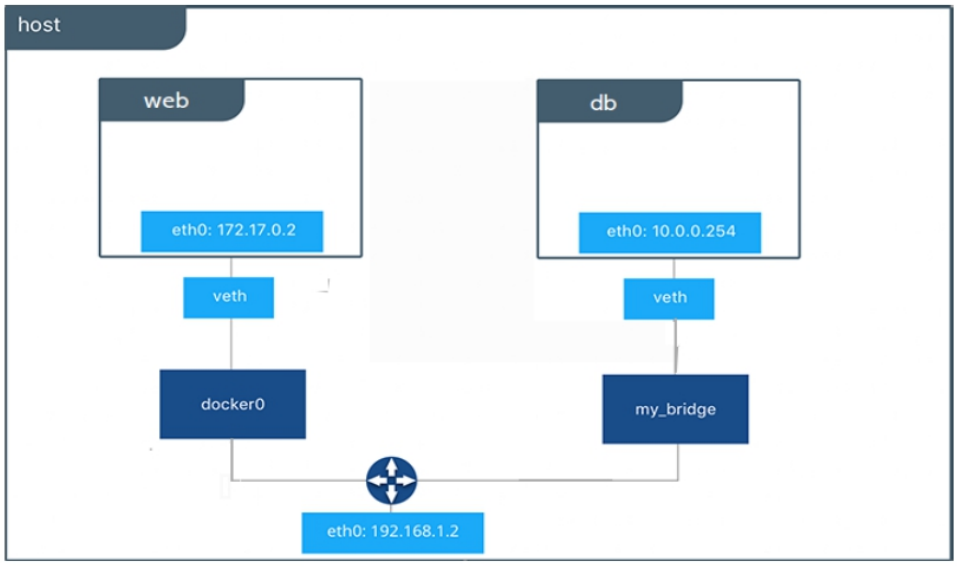
#vim /etc/hosts

172.x.0.x Sample4.example.com

#exit

USER-DEFINED BRIDGE NETWORK

- These networks are superior to the **default bridge network**.
- It is usually used when your applications run in standalone containers that need to communicate.
- These are best when you need multiple containers to communicate on the same Docker host.



DIFFERENCE BETWEEN USER-DEFINED BRIDGES AND THE DEFAULT BRIDGE:

- User-defined bridges provide automatic DNS resolution between containers.
- User-defined bridges provide better isolation.
- Containers can be attached and detached from user-defined networks on the fly.
- Each user-defined network creates a configurable bridge.
- Linked containers on the default bridge network share environment variables.

Manage a user-defined bridge:

#docker network ls

#docker network create my-bridge

#docker network ls

Create a container on my-bridge:

#docker run -d -it --name Test1 --network my-bridge centos

#docker ps

#docker exec -it Test1 ip a

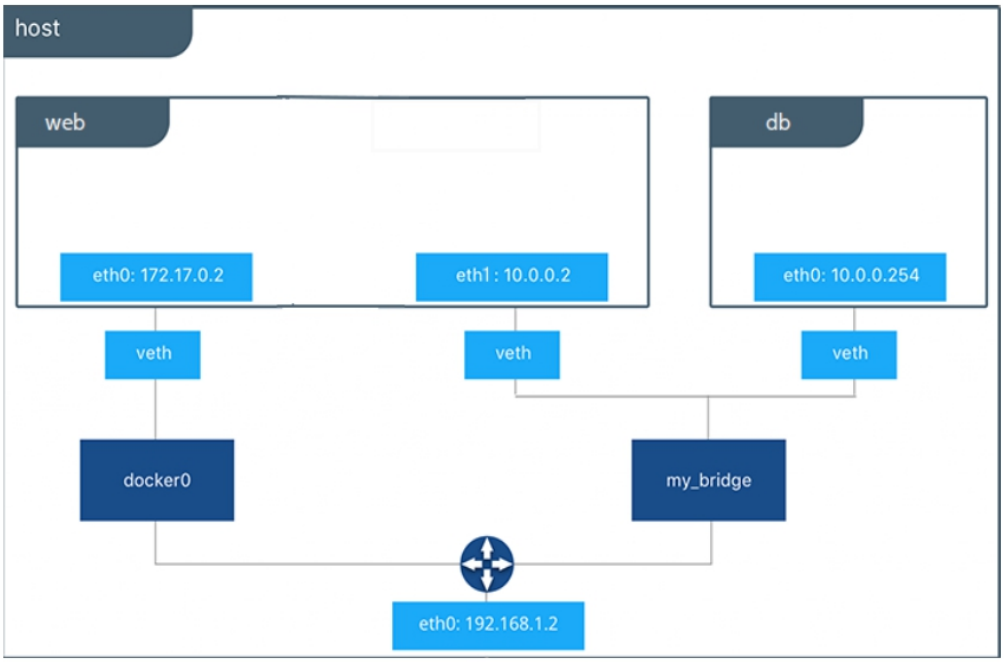
Connect a container to a user-defined bridge:

#docker run -it -d --name web-nginx --network my-bridge -p 8080:80 nginx

Disconnect a container from a user-defined bridge:

#docker network disconnect my-net my-nginx

COMMUNICATING DEFAULT & CUSTOM NETWORK CONTAINERS:



Create a new container on default bridge:

#docker run -d -it –name web centos

#docker ps

#docker exec -it web ip a

Creating custom subnet:

#docker network create my-bridge --subnet 10.0.0.0/16 --gateway 10.0.0.1

#docker network inspect my-bridge

Create a Container on My-Bridge:

#docker run -d -it --name db --net my-bridge centos

#docker exec -it db ip a

#docker exec -it db bash

#ping 172.17.0.2

Connect a Default network from My-Bridge network:

docker network connect bridge db

NOTE: Now Network Test is working

Disconnect a container:

#docker network disconnect bridge db

Removing User-Defined Bridge network:

Before Removing, if containers are currently connected to the network, disconnect them first.

#docker network ls

#docker network rm my-bridge

HOST NETWORK:

- If you use the **host network** mode for a container, that container's network stack is not isolated from the Docker host (the container shares the host's networking namespace), and the container does not get its own IP-address allocated.
- These are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.

  NOTE: The host networking driver only works on Linux hosts.

Create a new container:

#docker run -d -it--name Web1 host --network host nginx

#docker ps

#docker exec -it web1 ip addr show

Create another container from Centos image:

#docker run -it --name Web2 –network host centos bash

#ifconfig

NOTE: Host and Container networks are same

OVERLAY NETWORK:

- It connects multiple Docker daemons together and enable **swarm services** to communicate with each other.
- It provides to facilitate communication between a **swarm service** and a **standalone container.**
- These are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.

IPVLAN NETWORK:

- IPvlan networks give users total control over both IPv4 and IPv6 addressing.

MACVLAN NETWORK:

- It allows you to assign a **MAC address** to a container, making it appear as a physical device on your n/w.
- **macvlan driver** is sometimes the best choice when dealing with **legacy applications.**
- These are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.

NONE NETWORK:

- None network disables the complete networking stack on a container.
- Usually used in **conjunction** with a custom network driver.

NOTE: Not available for **swarm services**.

Create a New Container (Disabling network for a container):

docker run --rm -dit --network none --name myalpine alpine sh

Check the container's network stack, by executing some common networking commands within the container.

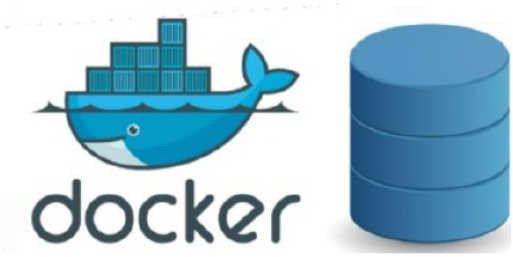
docker exec myalpine ip link show

NOTE: No ethernet was created

Only one instance of "host" and "null" networks are allowed.

#exit

MANAGE DOCKER STORAGE

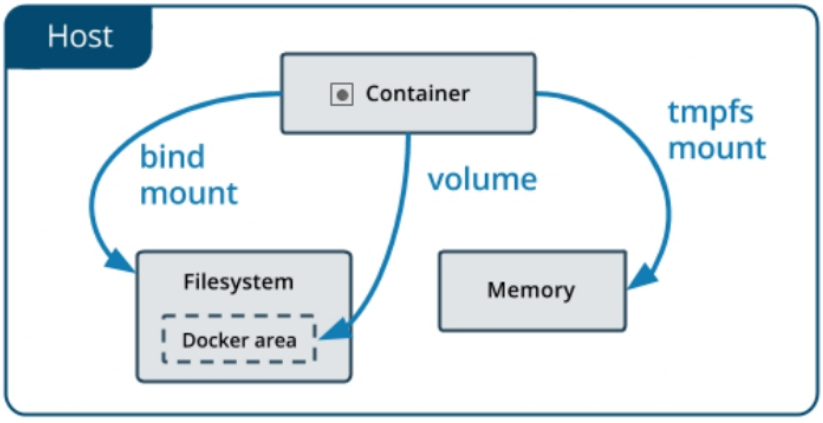


MAIL: Rnraju4u@gmail.com NUMBER: +91 9848363431 YOUTUBE: SYSGEEKS

MANAGING DATA IN DOCKER

- By default, all files created inside a container on a **writable container layer.**
- That means:
  - The data doesn't persist when that container no longer exists.
  - You can't easily move the data somewhere else.
- Docker has two options for containers to store files in the **host machine persistently**.
  - **volumes**
  - **bind mounts**
- Running Docker on **Linux** use a **tmpfs mount** and on **Windows** use a **named pipe.**

MOUNTING TYPES:



- **volumes** are often a better choice than persisting data in a container's writable layer, because a volume doesn't increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.
- If your container generates non-persistent state data, consider using a **tmpfs** mount to avoid storing the data anywhere permanently, and to increase the container's performance by avoiding writing into the container's writable layer.

TMPFS MOUNT

- When you create a container with a tmpfs mount, the container can create files outside the container's writable layer. As opposed to volumes and bind mounts, a tmpfs mount is temporary, and only persisted in the host memory.

LIMITATIONS OF TMPFS MOUNTS:

- Unlike volumes and bind mounts, you can't share tmpfs mounts between containers.
- This functionality is only available if you're running Docker on Linux.

Create a New Container with tmpfs mount:

#docker run -d -it --name app1 --tmpfs /data centos

#docker ps

Connect a Container:

#docker exec -it app1 bash

#cd /data

#touch abc

#exit

Now Stop & Start a Container:

#docker stop app1

#docker start app1

Verify the data in a /data mount point:

#docker exec -it app1 ls /data [No files in a mount point /data]

VOLUMES

- **Volumes** are the best way to **persist data** in Docker.
- These are stored in a **host filesystem** which is managed by Docker **(/var/lib/docker/volumes/** on Linux).
- Volumes support volume drivers, which allows you to store your data on **remote hosts / cloud providers.**

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.

MANAGING VOLUMES:

Create a new container:

#docker run -d -it –name -v /my-data Test1 centos

#docker exec -it Test1 bash

#cd /my-data

#touch file1 file2

By default docker volumes are: /var/lib/docker/volumes/

#cd /var/lib/docker/volumes/

#ls CID/_dada/

CREATE AND MANAGE VOLUMES:

Creates a new volume that containers can consume and store data in. If a name is not specified, Docker generates a random name.

Syntax: #docker volume create [options] [VOLUME]

#docker volume create my-vol

#docker volume ls

#cd /var/lib/docker/volumes

#ls

To inspect a Volume:

#docker volume inspect my-vol

Start a Container with a Volume:

#docker run -it --name Test2 -**v** my-vol:/my-data centos bash

#cd /my-data

#touch aws azure gcp

#ls

(or)

#docker run -it --name Test2 --**mount** source=my-vol,target=/my-data centos sh

##cd /my-data

#touch aws azure gcp

#ls

Creating volume from an existing directory with data:

#docker run -it -v volume:/var centos bash

#cd /var

#ls

#cd /var/lib/docker/volumes/volume/_data/

#ls

DATA VOLUME CONTAINERS:

Data volume containers come in handy(easy) when you have data that you want to share between containers.

Create a container with one volume:

#docker run -it -v /data --name datavolume centos bash

#cd /data

#touch abc

#exit

Now, we need to connect some containers to this /data directory in the container.

#docker run -it **--volumes-from** datavolume ubuntu bash

#cd /data

#ls

DOCKER VOLUME BACKUPS:

- while your containers are immutable, the data inside your volumes is mutable.
- It changes, while the items inside your Docker containers do not. For this reason, you need to make sure that you are backing up your volumes.
- Volumes are stored on the system at **/var/lib/docker/volumes/**

BIND MOUNT

- **Bind mounts** may be stored anywhere on the **host system.**
- 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 absolute path on the host machine.

NOTE: Bind mounts have **limited functionality** compared to **volumes.**

Start a Container with Bind mount:

#docker run -it -v /cloud:/aws centos bash

Here /cloud (on the Docker host) to the /aws directory inside the now running Docker container.

#cd /aws

#touch aws azure gcp

On the Host Machine verify the path:

#cd /cloud

#ls

USE A READ-ONLY VOLUME:

It mounts the directory as a read-only volume, by adding **ro.**

mount it in the read-only mode:

#docker run -it -v /data:/app:ro centos bash

#cd /app

#touch abc [

NOTE: Read-only file system error

Mount into a non-empty directory on the container:

- If you bind-mount a directory into a non-empty directory on the container, the directory's existing contents are obscured by the bind mount. This can be beneficial, such as when you want to test a new version of your application without building a new image.

#docker run -d -it --name broken-container -v /tmp:/usr nginx

The container is created but does not start. Remove it:

#docker container rm broken-container

REMOVING ONE OR MORE VOLUMES:

You cannot remove a volume that is in use by a container.

#docker volume ls

#docker volume rm my-vol

To Remove volume forcefully:

#docker volume rm -f my-vol

Remove all unused local volumes

#docker volume prune

#docker volume ls

NAMED PIPES:

- A **npipe** mount can be used for communication between the **Docker host** and a **container.**
- Common use case is to run a third-party tool inside of a container and connect to the Docker Engine API.

VOLUMES VS BIND MOUNT

- Compared to Bind Mounts, Volumes are more flexible and have more features, making them the most recommended option.
- In your container, Bind Mount provides you access to local file/directory storage on your local machine.

Volumes	Bind Mount
Easy backups and recoveries	There is a bit of complexity involved in backup and recovery. You don't have to worry about it if you know what folders to backup
To mount it, you only need the volume name. Paths are not required.	It is necessary to provide a path to the host machine when mounting with bind mounts.
Containers can have volumes created while they are being created.	The mount folder will be created when the host machine doesn't contain the folder.
There are APIs and CLIs for interacting with Docker volumes.	Using CLI commands, you cannot access bind mounts. The host machine still allows you to work with them instantly.
The volumes are stored in `/var/lib/docker/` volumes.	A bind mount can reside anywhere on a host computer.

MAIL: Rnraju4u@gmail.com **NUMBER:** +91 9848363431 **YOUTUBE:** SYSGEEKS

DOCKER COMPOSE

- Docker Compose is a tool for defining and running multiple containers as a single service.
- with a single command, you create and start all the services from your configuration.
- Compose works in all environments: production, staging, development, testing, as well as CI workflows. Commands for application:
  - Start, stop, and rebuild services
  - View the status of running services
  - Stream the log output of running services
  - Run a one-off command on a service
- The key features of Compose that make it effective are:
  - Have multiple isolated environments on a single host
  - Preserves volume data when containers are created
  - Only recreate containers that have changed
  - Supports variables and moving a composition between environments

SERVICE:

- A **service** can be run by one or multiple containers.
- Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.

DOCKER COMPOSE FILE STRUCTURE:

```
services:
  foo:
    image: foo
  bar:
    image: bar
    profiles:
      - test
  baz:
    image: baz
    depends_on:
      - bar
    profiles:
      - test
  zot:
    image: zot
    depends_on:
      - bar
    profiles:
      - debug
```

USING COMPOSE IS BASICALLY A THREE-STEP PROCESS:

STEP1: Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.

STEP2: Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.

STEP3: Run **docker compose up** and the **Docker compose command** starts and runs your entire app.

INSTALL DOCKER COMPOSE:

STEP 1: To download and install Compose standalone, run:

#curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compos

STEP 2: Apply executable permissions to the standalone binary in the target path for the installation.

#chmod +x /usr/local/bin/docker-compos

STEP 3: Create a Soft link for binary:

#ln -s /usr/local/bin/docker-compos /usr/bin/docker-compos

STEP 4: Test and execute compose commands using docker-compose.

#docker-compos version

SAMPLE APPLICATION WITH COMPOSE:

Create docker compose file at any location on your system.

#mkdir /dockercompos

#vim docker-compose.yml

```
version: '3'   ###https://docs.docker.com/compose/compose-file/     [versions]
services:
  web:
    image: nginx
    ports:
       - 9090:80
  database:
    image: redis
```

Check the validity of file:

#docker-compos config

Run the compose file:

#docker-compos up -d

#docker-compos ps or #docker ps

Bring down application

#docker-compos down

SCALE A SERVICES:

#docker-compos up -d --scale database=4

#docker-compos ps

#docker-compos down