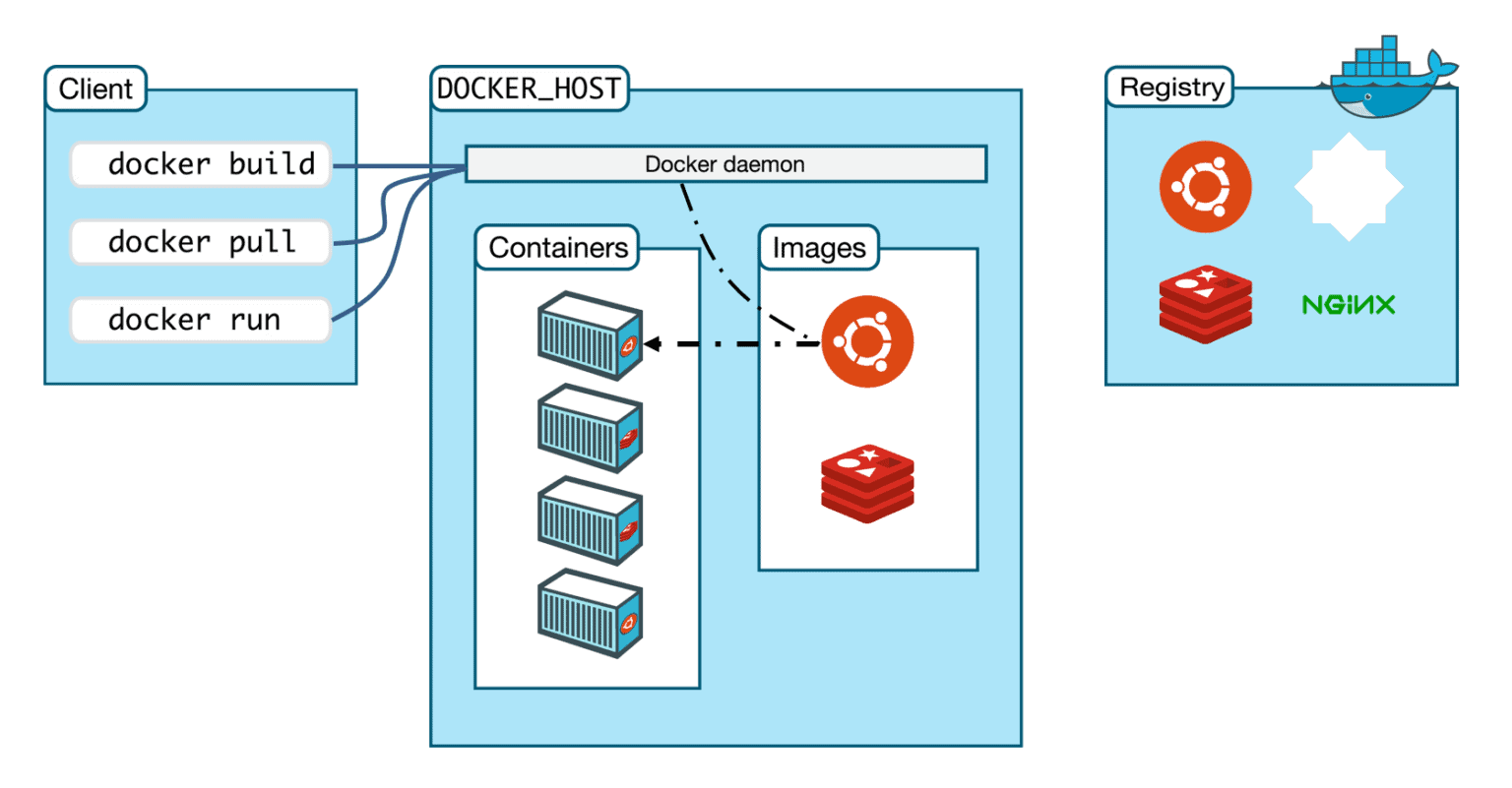
Docker

**1.What is Docker?**

Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment, be it development, test or production.

**2. Docker Architecture.**



a) **Docker Daemon** - listens to Docker API requests and manages Docker objects such as images, containers, networks and volumes.

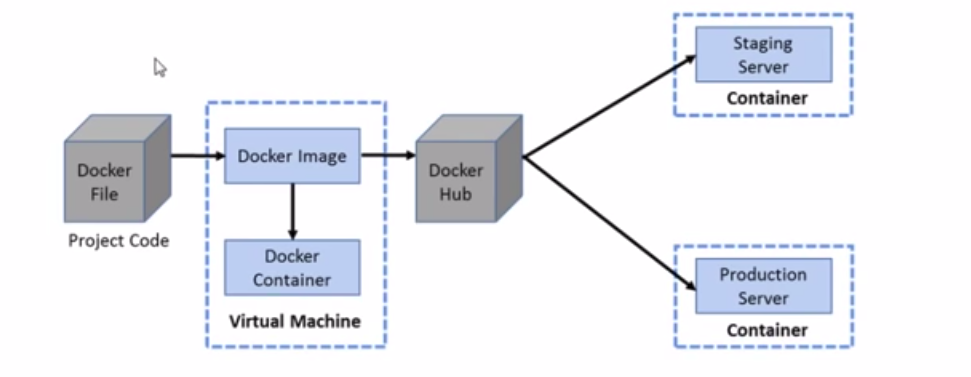
b) **Docker Clients** - With the help of Docker Clients, users can interact with Docker.

c) **Docker Host**- provides a complete environment to execute and run applications. It comprises of the Docker daemon, Images, Containers, Networks, and Storage.

d) **Docker Registry** -Docker Registry manages and stores the Docker images. Public Registry is also called as Docker hub.Private Registry - It is used to share images within the enterprise.

e) **Docker Image**s- are read-only templates that you build from a set of instructions written in Dockerfile.

Workflow of Docker container



**Installation of docker in centos machine.**

1. Set up the repository

sudo yum install -y yum-utils

2. sudo yum-config-manager \  
 --add-repo \  
 <https://download.docker.com/linux/centos/docker-ce.repo>

3. sudo yum install docker-ce docker-ce-cli containerd.io docker-compose-plugin

4. sudo systemctl start docker

5. sudo docker run hello-world

**What is container?**

Containers are a software package into a logical box with everything that the application needs to run. The software package includes an operating system application code, runtime,system tools, system libraries, and binaries

**Docker commands:**

**1.To start docker**

systemctl start docker

**2.To pull image from repository**

docker pull centos

docker pull ubuntu

**3.To show all images**

docker images

**4. creates the container named mycontainer from centos image**

docker create --name mycontainer centos

creates the container named mycontainer from centos image

**5. To delete docker container**

docker rm <containername>

**6.To remove image**

docker rmi <imagename>

**7.To run a container**

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

docker run <image-name>

note: if you use an image that is not in your local else it pull the image from docker registry.

docker create --name mycontainer <imagename>

docker create --name mycontainer <imageid>

**8. To run a container in attached mode(foreground) and in detached mode (background)**

By default, Docker runs the container in attached mode. Meaning it’s attached to the terminal session, where it displays output and messages.

if you want to keep the container and current terminal session separate, you can run the container in the background using the -d attribute

docker container run -d [docker\_image]

**9.To run a container interactively**

This allows you to execute the commands inside the container

docker container run -it [docker\_image] /bin/bash

**10.Run a container and publisher container port**

When you run a container, the only way to access the process is from inside of it. To allow external connections to the container, you have to open (publish) specific ports.

-p [host\_ip]:[host\_port]:[container\_port]

docker container run -p 8080:80 [docker\_image]

**11. To start the container**

docker start [OPTIONS] CONTAINER [CONTAINER...]

Docker start –a <container-name>

**12.To stop the container**

docker stop <container-name>

**13. To restart a container- stops and restart a container**

. docker stop <container-name>

**14. Shows only the running container.**

docker ps

15.Docker ps –a ->shows all stopped and running containers.

**16.container creation**

Docker container create –it –name <container-name> <image-name>

**17. The exec command runs a new command in a running container.**

Docker exec <container-name> <command name>

Docker exec <container-name> ls

**18. Docker exec command which is used to run the command inside a container or to go inside a container.**

docker exec –it test /bin/bash

To effectively interact with the container this command is used.

**19.docker commit**

Instead of launching a new container from zero, we can commit the old Docker container to create a new Docker image and use that to start a new container with the right ports open.

* docker stop <container-name>
* docker commit <container-name> <new-container-name>
* docker rm <container-name>
* Docker container rm –force <containername>
* docker run -d --name httpd-container httpd

**20. Attach command connects our terminal to running container**

Docker container attach [options] container

21.Default mode - The above command links the standard output (*stdout*), and the standard error (*stderr*) streams with our terminal. So, **we can see the console output of the container in our terminal.**

docker run --name test\_redis

**22. Interactive mode** – Let us to interact with the container from out terminal. -I option attaches a standard input of the bash shell to the container.

**23.Detached mode** - -d option which lets the container to run in background.

So whenever required we can connect to container using its id and name.

**24.Docker Build**

This command which is used to build a docker image from a docker file.The build context is set of files specified in the path or url.

Url parameter – which can be git repositories,tar ball contexts,text files.

docker build dockerfile

25. **docker container rm --force mycontainer1**

To delete the running container forcefully.

**26.docker container cp**

This command copies file from container to host machine or from host machine to container .

docker container cp <containerid>:/src path /destination path

docker container cp /srcpath containerid:/destination path.

**27.** **docker container create image-name.**

This command which creates the container and but does not start it.

Docker containr create nginx

**28. Docker logs <cont-d>**

To show the logs of the container.

**29.Docker inspect <cont-id>**

To check the state of the container.

**30 .Docker checkpoint**

checkpoint and Restore is an experimental feature that allows you to freeze a running container by checkpointing it, which turns its state into a collection of files on disk. Later, the container can be restored from the point it was frozen.

Usage: docker checkpoint create [OPTIONS] CONTAINER CHECKPOINT

Create a checkpoint from a running container

--leave-running=false - Leave the container running after checkpoint

--checkpoint-dir - Use a custom checkpoint storage directory.

Inspect changes to files or directories on a container’s filesystem

**31.docker container export –o <gzfilename> <containername>**

This command is used to export the container in gz file.

**32. docker container inspect <container name>**

Docker inspect command which provides detailed info about container

**33.Docker container logs [options] container-name**

Fetches the container logs

* Docker container ls [options]

Docker container ls -a

Lists the running container by default and lists all container

**34.Docker container pause <container-name>**

Pause all processes or container with one or more processes.

**35. Docker container port <container-name> private\_port [/proto]**

List port mappings for a container.

**36. docker container prune**

Removes stopped container

**37.docker container rename <container-name> <new-name>.**

Renames a container name

**38.Docker container restart <container –name>**

Restart one or more container.

**39. docker container rm [options] container-name**

Removes one or more container .

**40.docker container rm --force <container –name >**

To remove the running container forcefully.

.41. docker diff

Inspect changes to files or directories on a container’s fileeystem.

42. Docker info

This command displays system wide information regarding the Docker installation.

43. Docker kill <container1> <con2> …

Kills one or more container.

44. docker load --input fedora.tar

Load an image from a tar archive or STDIN

45. Docker login

Login to a registry . Need to enter docker id and password in order to pull and push from the registry.

46. Docker top <contid>

List the top processes in container

47. Docker pause <contid>

To pause the container and unpause the container.

48. Systemctl docker stop

To stop the docker

**---------------------------------------------------------------------------------------------------------------**

**DOCKERFILE**

Docker file is a file which consists set of instruction to create a docker image.

Docker file instruction command

**FROM** – specifies an image to start from

**WORKDIR** – sets the current working directory inside the container equivalent to running cd.

**RUN** – executes the given command inside the container.

**COPY**- copies the file or directory from local host machine to container file system

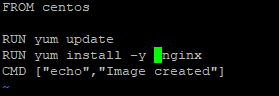
**CMD**- specifies a command to execute when the image is run.CMD is the command the container executes by default when you launch the built image.

Once the dockerfile is ready we can execute docker build command to build an image from docker file.

MAINTAINER**-** command is the person who is going to maintain this image. Here you specify the keyword and just mention the email ID.

**docker build –t image-name:tagname <dir> or docker build –t dockerfile .**

* **-t** − is to mention a tag to the image
* **ImageName** − This is the name you want to give to your image.
* **TagName** − This is the tag you want to give to your image.
* **Dir** − The directory where the Docker File is present.





FROM alpine

RUN apk update

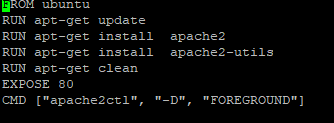
RUN apk add wget

RUN rm -rf /var/cache/apk/\*

WORKDIR /root/

ENTRYPOINT [ "wget"]

CMD ["--help"]



From the built image docker container can be created .

**Docker volumes.**

* Docker volumes which is used to persist data backup.
* To define Docker Volumes, they are file systems that can be mounted on Docker containers.
* They help in preserving the data and are independent of the container life cycle.
* One of the major advantages of Docker Volumes is that it allows the developers to backup their data and also allows easy sharing of file systems among Docker containers.
* We can easily mount a volume when we launch a Docker container.
* It is also possible to mount the same volume to different containers and this allows easy sharing of data between them and this can be easily achieved with the use of simple commands and flags.

**Creation of docker Volume**

Sudo docker volume create <volume-name>

Docker creates a particular directory for volume on the local machine

**List all the docker volumes.**

Sudo docker volume list

**Inspecting a docker volume**

Sudo docker volume inspect <volume-name>

**Mounting docker volumes.**

sudo docker run −−mount source=<name of volume>,destination=<path of a directory in container>/volumename <image\_name>

For example- sudo docker run −it −−mount source=myVolume,destination=/usr/src/app/myvolume ubuntu

In order check the volume for stopped container. You need to start the container and need to execute docker exec –it <container-name> /bin/bash command.

You can mount a docker volume to a docker container using the mount –flag. when you are running the Docker run command. You can also mount the same volume to multiple Docker containers and all the containers would have a shared access to the volume.

**Deleting a docker volume.**

In order to delete a docker volume, you need to ensure that the volume is not in use at that moment. If a container is running with the volume mounted in it, you would have to stop the container first before removing the mounted volume. After you have stopped the container, you can use the following command to remove the volume.

sudo docker rm <name of volume>

In order to delete all the volumes at once, make sure that none of the volumes is currently in use

Sudo docker volume prune.

## Sharing a Docker Volume with multiple Docker Containers

If you want to share multiple files with multiple docker container ,you can put your files in a docker volume mount that volume with multiple containers and get shared access to that volume.

Sudo docker volume create myvolume

Sudo docker run –it –name=container1 –mount source=myvolume,destination=/app ubuntu

This will create a volume myvolume and mount this volume to a container called container1 of ubuntu image at a destination /app

**How to mount a volume of host directory to docker container**

docker run -it --name container1 –v /home/centos:/datavolume ubuntu

docker run –it –name <container name> -v <path of the host directory>:/datavolume ubuntu

**How to share the voume from one container to other container.**

**Docker run –it –volumes-from container1 –name container2**

**How to push the images to your account in docker hub.**

docker login

Which will ask for username and password.

Once login is succeeded proceed with other steps.

Docker tag <imageid> username/<imagename>:tagname

docker tag ab736043b5ac 76625/dockerfile:firsttry

docker push <dusername>/imagename

**Docker Network**

Docker network which is used to connect two containers in a network. For Docker containers to communicate with each other and the outside world via the host machine, there has to be a layer of networking involved. Docker supports different types of networks, each fit for certain use cases.

### **The Bridge Driver**

This is the default. Whenever you start Docker, a bridge network gets created and all newly started containers will connect automatically to the default bridge network.

### **The Overlay Driver**

The Overlay driver is for multi-host network communication, as with [Docker Swarm](https://docs.docker.com/engine/swarm/) or [Kubernetes](https://kubernetes.io/). It allows containers across the host to communicate with each other without worrying about the setup. Think of an overlay network as a distributed virtualized network that’s built on top of an existing computer network.

**Host Networking**

f you use the host network driver for a container, that container’s network stack is not isolated from the Docker host. For instance, if you run a container which binds to port 80 and you use host networking, the container’s application will be available on port 80 on the host’s IP address.

Macvlan Network

Some applications, especially legacy applications or applications which monitor network traffic, expect to be directly connected to the physical network. In this type of situation, you can use the macvlan network driver to assign a MAC address to each container’s virtual network interface, making it appear to be a physical network interface directly connected to the physical network. In this case, you need to designate a physical interface on your Docker host to use for the Macvlan, as well as the subnet and gateway of the Macvlan

**Network commands**

1. **Docker network ls -** This command can be used to list all the networks associated with Docker on the host.

2. **docker network inspect networkname** - If you want to see more details on the network associated with Docker, you can use the Docker **network inspect** command.

sudo docker network inspect bridge

3.**Creating Your Own New Network**- One can create a network in Docker before launching containers.

docker network create –-driver drivername name

4.**Attach the new network** when launching the container.

sudo docker run –it –network=new\_nw ubuntu:latest /bin/bash

Docker network create network-name

Docker run –net mynet –name server1 –dit ubuntu

Docker run –net mynet –name server2 –dit ubuntu

Docker inspect server1 | grep IPADRESS

Docker inspect server1 | grep IPADRESS

Go inside server1 container docker exec –it server1 bash

Then ping ip address of another container

**Container linking**

* Container Linking allows multiple containers to link with each other. It is a better option than exposing ports. Let’s go step by step and learn how it works.
* Sudo docker jenkins pull
* Sudo docker run --name=jenkinsa –d jenkins
* Docker run –name=reca –link=jenkinsa:alias-src –it ubuntu
* Now attach to the receiving container docker attach reca
* Then run **env** command
* **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
* **Host networks** 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.
* **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
* **Macvlan networks** 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.
* **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

**Docker-compose**

* **Docker Compose** is used to run multiple containers as a single service.
* For example, suppose you had an application which required NGNIX and MySQL,
* you could create one file which would start both the containers as a service without the need to start each one separately.

**Docker-compose Installation**

Step 1: curl -L "https://github.com/docker/compose/releases/download/1.10.0-rc2/dockercompose  
 -$(uname -s) -$(uname -m)" -o /home/demo/docker-compose

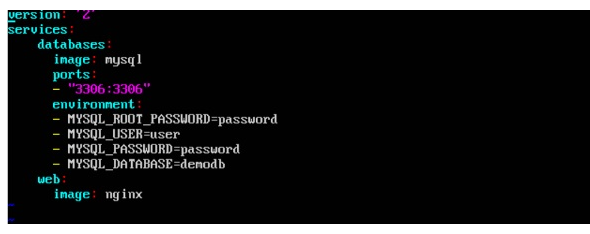
Step 2: Next, we need to provide **execute privileges** to the downloaded Docker Compose file, using the following command

chmod +x /home/demo/docker-compose

Step 3: docker-compose version

**Creating the docker-compose file**

sudo vim docker-compose.yml



* The **database** and **web** keyword are used to define two separate services. One will be running our **mysql** database and the other will be our **nginx** web server.
* The **image** keyword is used to specify the image from **dockerhub** for our **mysql** and **nginx** containers
* For the database, we are using the ports keyword to mention the ports that need to be exposed for **mysql**.
* And then, we also specify the environment variables for **mysql** which are required to run **mysql**.