DOCKER

Installation :

→ sudo yum update

→ yum install docker-io

→ systemctl start docker

→ systemctl status docker

→ systemctl enable docker

→ systemctl stop docker

→ docker --version

→ docker search (image name)

Ex: docker search tomcat

→ docker pull image-name (to pull the image from docker hub to your local registry)

→ docker images (to list the images)

→ docker rmi image-name (to remove or delete image)

--> curl ifconfig.co (to know the public ip)

--> vi Dockerfile

--> docker pull image:version

--> docker inspect imageid (to inspect that image, such as home directory and other important concepts)

--> docker images --- to list the images

--> dcoker rmi imagesid -- to delete the images

--> docker build -t jpets:${BUILD\_NUMBER} Dockerfileloation

--> docker run -i -t imageid /bin/bash -- to make a container

-- > docker run -- name <NAME> -it image

--> docker rmi ID --force --- to delete forcefully

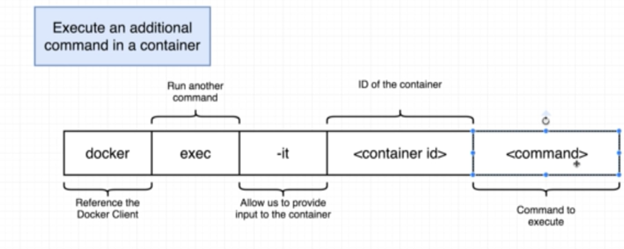
--> ip addr | grep global --- to get the ips

--> ctrl pq --- to exit safely from running container

--> docker ps --- to see running container

--> docker ps -a --- to see all the the exited running containers

--> docker exec -it container\_id /bin/bash -- to run a command inside the docker container in this case we are passing command to have ‘/bin/bash’ shell, so that you can perform linux commands.



----> docker exec -it -u root deb88bd52bf7 /bin/bash (we can loign directly as a root user using bash shell command)

--> docker run --name newtom -it --rm -p 8080:8080 newtom:2.0 (here 'rm' is used to delete the container once exit from the container)

--> docker run -i -t imagename:version (if you dont want to map any port)

--> docker run --rm --name newtown -p 8080:8080 -d tomcatimage:1.0

--> docker run –name <NAME> -it -p 8889:8080 -d imageid:tag /bin/bash ---> to run a image

--- here lefthand side ip i.e, 8889 is the machine ip and right hand side ip 8080 is the container ip(i.e, tomcat ip). In this case we are mapping tomcat container ip with machine avaiable ips.

--> docker start containerid --- to start the stopped container

--> docker start -ai containerID --> to start the stopped container in attached and interavtive mode

--> docker attach conatinerID --- to enter into the conatiner

--> docker container rm -f 6abe7d7e87a5 (to delete docker containers)

--> docker stop containerid (NOTE: If docker file has not met the guidelines, we can't start the container, again we need to create the container with right content)

--> docker commit #ContainerID Newimagename:version (This is used for creating the new image from the existing container id)

we can create a new image from the existing image with the required changes as we wish. so that we can map to the new port numbers....

---> docker ps -a --filter "status=exited"

---> docker rmi -f $(docker images)

--> docker ps -aq --filter "status=exited"

--> docker ps -a --filter "status=running"

--> docker rm $(docker ps -aq --filter "status=exited") (to remove all the containers which are in 'exited state')

--> docker system prune (to delete all the exited containers in , can use -f (force)) option too

* Docker container prune

--> docker start $(docker ps -aq) (To start all the containers)

* docker start $(docker ps -a -q -f status=exited)

--> docker volume create volume-name

--> docker volume ls

--> docker inspect 'volume-name' (default location: /var/lib/docker/volumes)

--> docker -d -v 'volume-name':/containers/volume-direcotry:ro imageid .........

Note: ro --> means read only , this is optional

docker run -d -u 0 -p 8080:8080 -p 50000:50000 -v /opt/jenkins:/var/jenkins\_home --privileged --name Jenkins:2.46.3 (this is too map containers volume with the docker host volume)

--> docker inspect containerID | grep -A10 -B10 'Source' (we can inspect which local volume has been mapped to this container id)

--> docker volume rm 'volume-name' (Note: we can remove 'volume' only after removing the container)

How to push the docker images to docker hub:-

-----> docker login and it will ask for username and password (or) "docker login -u DockerhubUsername -p password"

-----> docker tag IMAGEid docker.io/Username/imagename:tag(ex: 2.0)

-----> docker push docker.io/Username/imagename

How to push the docker images to nexus registry:-

----> docker login -u usrname -p password <nexus-hostname>:<repository-port>

----> docker tag <imageId or imageName> <nexus-hostname>:<repository-port>/<image>:<tag>

----> docker push <nexus-hostname>:<repository-port>/<image>:<tag>

----> docker pull saiboby/timage:2.0 (to pull images from private docker registry)

----> docker network ls

----> docker network inspect netowrk-name(for example bridge,host,none)

----> docker network create --driver bridge new-network-name

----> docker run -itd --name alpine4 --network new-network-name alpine ash(bash shell)

--> docker network rm netowrkID

Note: The default network driver is "bridge"

--> docker network create --subnet=172.18.0.0/16 mynet123 (creating network using our own subnet)

--> docker run --net mynet123 --ip 172.18.0.22 -it ubuntu bash (creting contianer using our won ip address)

---> docker rename old-name new-name (using this we can rename the existing/running containers)

--> docker logs containerid/name

---> docker-compose up

--> docker-compose down

---> docker-compose ps

---> docker-compose config (to validate the syntax)

link : https://www.linode.com/docs/applications/containers/how-to-use-docker-compose/

WORKDIR --> The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.

To mention the absolute direcotry of your workdirectory. Note: This location is inside your container

VOLUME --> Instruction should be used to expose any database storage area, configuration storage, or files/folders created by your docker container

ENTRYPOINT --> The best use of ENTRYPOINT is to set the image’s main command, allowing that image to be run

COPY --> it is used to copy local files to the container(Note: the files should be in the direcotry where Dockerfile presents or in the sub directories)

ex: COPY myfile.txt /tmp/

ADD --> To download the files remotely and place in container

ex: ADD http://example.com/big.tar.xz /usr/src/things/

EXPOSE --> It is used to mention the port which the services are running

ex: EXPOSE 8080 (Default for tomcat)

RUN ---> RUN lets you execute commands inside of your Docker image.

These commands get executed once at build time and get written into your Docker image as a new layer.

For example if you wanted to install a package or create a directory inside of your Docker image then RUN will be what you’ll want to use.

For example, RUN mkdir -p /path/to/folder

LABEL 🡪 The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing

CMD --> CMD lets you define a default command to run when your container starts.

For example start the tomcat service on container starting time...

Referecfen link: https://nickjanetakis.com/blog/docker-tip-7-the-difference-between-run-and-cmd

Note: ENTRYPOINT overwrite the CMD commands. And CMD can be overwritten during the runtime of docker container

he ENTRYPOINT specifies a command that will always be executed when the container starts. The CMD specifies arguments that will be fed to the ENTRYPOINT

ARG 🡪 instruction defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag.

ONBUILD 🡪 The ONBUILD instruction adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build. The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile (https://vsupalov.com/docker-build-time-env-values/)

NOTE: docker run -d -u 0 -p 8989:8080 -p 50000:50000 -v /opt/jenkins:/var/jenkins\_home --privileged --name jenkins (this is too map containers volume with the docker host volume)

docker run -u 0 -p 8988:8080 -p 50000:50000 -v /your/home:/var/jenkins\_home Image:version --name jenkins

Note-1: in the above we are maping container volume to the local docker host (and whatever we do changes in local volume will appear automatically in container and vice versa will get chnages made in container will see in local volume)

and Left hand side ip is the machine ip and right hand side ip is contianer ip. Here we are mapping container with the machine or host's ip. So that we can access in the browser.

suppose the above created on the name of jenkins , that we can stop/start like docker stop/start jenkins

---> 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.

If we go with 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.

---> 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.

Installation of Docker CE: https://www.vultr.com/docs/installing-docker-ce-on-centos-7

Dockerfile : https://linuxtechlab.com/learn-create-dockerfile-example/ for centos creation

ENV DIR\_NAME /opt/ {Inside dockerfile}

docker run -it -e DIR\_NAME='babu' cen:5 {ex of passing the environment value duing run command execution}

CMD ["sh", "-c", "echo $DIR\_NAME >> /opt/sai.txt"]

CMD cp -rf /home/sai.txt /opt/sai1.txt

docker exec -it -u root deb88bd52bf7 /bin/bash ()

DOCKER COMPOSE:

Docker compose is a utility which connects multiple containers together and runs accordingly. It is like a POD in kubernetes.

--> docker-compose up

--> docker-compose down

---> docker-compose ps

---> docker-compose config (to validate the syntax)

link : <https://www.linode.com/docs/applications/containers/how-to-use-docker-compose/>

We have to write docker-compose yaml file to build the container structure.

Below is the sample structure of docker compose.

**version: "3"**

**services:**

**web:**

**build: web**

**command: python app.py**

**ports:**

**- "5000:5000"**

**volumes:**

**- ./web:/code # modified here to take into account the new app path**

**links:**

**- redis**

**environment:**

**- DATADOG\_HOST=datadog # used by the web app to initialize the Datadog library**

**redis:**

**image: redis**

**# agent section**

**datadog:**

**build: datadog**

**links:**

**- redis # ensures that redis is a host that the container can find**

**- web # ensures that the web app can send metrics**

**environment:**

**- DD\_API\_KEY=\_\_your\_datadog\_api\_key\_here\_\_**

**- DD\_DOGSTATSD\_NON\_LOCAL\_TRAFFIC=true**

**volumes:**

**- /var/run/docker.sock:/var/run/docker.sock**

**- /proc/:/host/proc/:ro**

**- /sys/fs/cgroup:/host/sys/fs/cgroup:ro**