Docker -2024

Monday, April 25, 2022

4:08 PM

To list the containers and images ID's and info

#Docker ps

To list the images existed in your local machine

#docker images

To pull images from the distribution

#docker run ubuntu:latest

Or

#docker pull httpd

To create a docker container out of image

#docker run -it -d httpd

To connect to the docker container

#docker exec -it <containerID> /bin/bash

To build an docker image:  once the docker file is create we will use below command to build.

#docker build  -t apache-server:v1 .          ( . Is pwd)

====================================================>

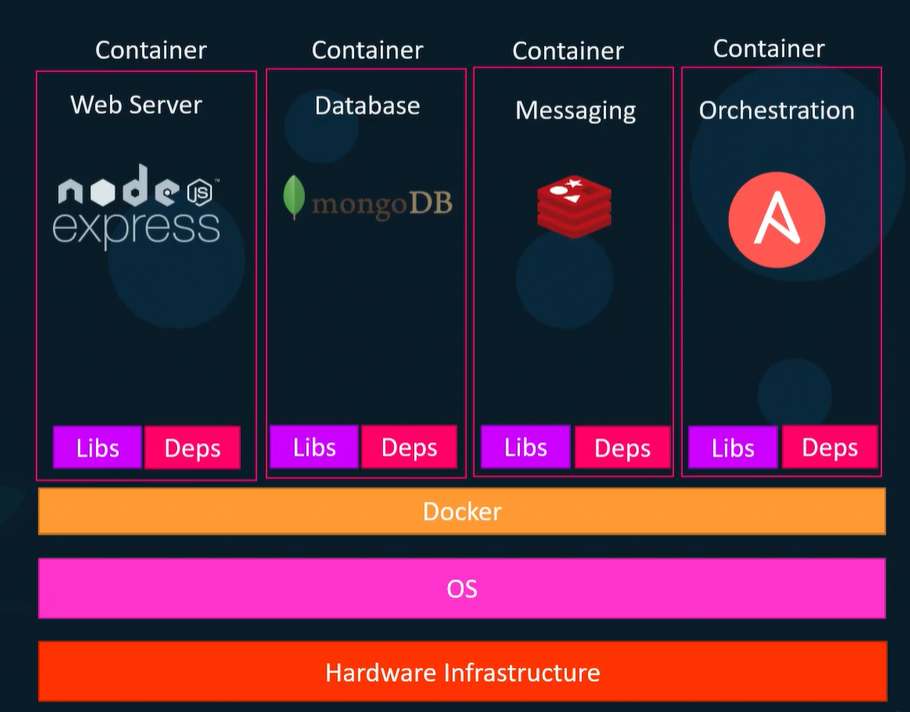
2024 :

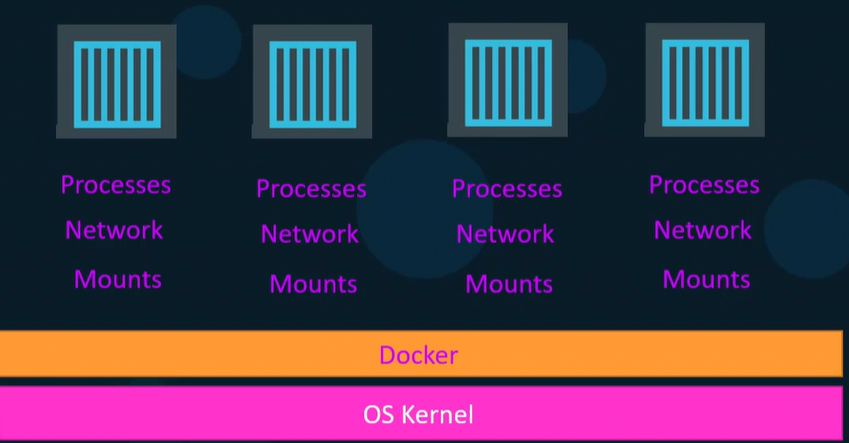
**Links**:

<https://docs.docker.com/compose/>

<https://docs.docker.com/reference/cli/docker/compose/>

<https://github.com/dockersamples/example-voting-app>





====> Containers are completely an isolated Env. They have own processes,network,mounts.

=====> Docker uses LXC Containers

=====> Docker allow us to run diff OS containers ,until we they are matching with underlying OS-Kernel.

====> if underlying OS Kernel is Linux ,Docker will not allow us to run Windows OS.

=====> Docker Image is template or package which is used to create a one or more Container.

=====> Docker Container is an isolated instance which runs the docker image and have own process, mounts ,network.

==>Docker Host = Where we install the docker.

**Basic Commands :**

#docker run ngnix ---> *this will run instance of ngnix application, first it will check in docker host , if not present it will check the Docker Hub.*

#docker ps -------->  *list containers with information*

#docker ps -a ---------> *List containers along with previously stopped or exited containers*

#docker stop "Container Name or Container ID" -------- > *this will stop the mentioned container.*

#docker remove "container name or ID" ---------> *this will remove the container , once done verify using "#docker ps -a " command.*

*#*docker rm "containername or ID "

#docker images ----> *this will list the images available in our host.*

#docker rmi "image name" ----> *this will remove the image from docker host, make sure no containers are using this image before removing.*

#docker pull "image name" -------> *this will pull the image and store it our docker host, so that it can be used in future without downloading again*.

Append a command :

#docker run ubuntu sleep 5 -----> *this will run the ubuntu image and goes to sleep for 5 seconds. Check status in ps -a.*

Execute a command:

#docker exec ngnix cat /etc/hosts ----> *this command will allow us to see certain outputs of a running container.*

To run in the background use detach option:

#docker run -d ngnix -----> *this will allow us to run the desired container in the be background. We can view the status from #docker ps -a*

To Attach back:

#docker attach <container ID> ------*>first four digits of container is enough.*

**Practice** :

#docker run centos -------> *this will download and runs the centos container, however as this is base image , it will get excited, as process ends because we are not running any service inside it.*

#docker run -it centos bash -----> *this will run the centos container and allow us to enter into the centos container. Once you done with it and press exit, then this centos container will get exited, as we are not running any service in it.*

#docker rm container name or ID ---------> *remove the container*

**Note** :

* *multiple containers can be removed at once*
* *starting 3 numeric of container ID is enough to mention*

#docker rm container id container id ------> *this will remove two containers at a time.*

#docker rmi image name -----------> *remove the Image*

#docker run --name webapp "image name" --------> to run a container with particular name

**More Deeper Commands :**

#docker run redis:4.0 ---> "4.0" is considered as tag , where we are launching a redis container with particular version.

#docker run ubuntu:17.10 ------> downloading particular version

#docker run ubuntu cat /etc/hosts ----> this will run the ubuntu container and provide us the /etc/hosts output.

#docker run -it ngnix

-it = interactive and -t for terminal

**Port mapping of docker host and docker container :**

* By Default every container is assigned with Port & internal IP
* In order to map docker host Port to container port , so that it can be accessed , we can use below command.

Eg :

#docker run -p 80:5000 nginx

#docker run -p 8080:8000 nginx

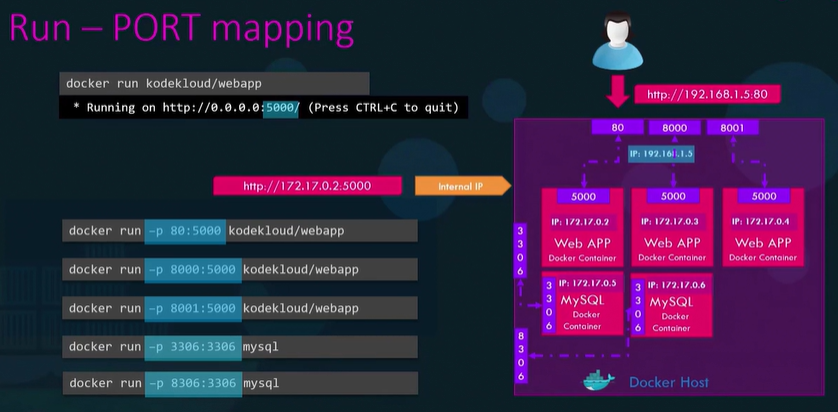
**Note**: Port cannot be done while docker container is running , we need to first stop container and do port mapping.

**Eg:**

**Steps**:

Incase if port mapping need to be done on running container:

* #docker stop container ID or name
* #docker ps
* #docker run -p 8080:8080 jenkins
* Docker ps
* Now container is mapped with docker host port 8080 and now the jenkins container can be accessed from outside using url <http://ip:8080>



**Volume Mapping :**

By default when container is created it will store the data in locations such as " /var/lib/mysql ", however this is not persistent.

If the container is deleted or stopped , the data will be wiped off or no longer available.

To have a persistent Volume , we can map from docker host using below command.

**#docker run -v /opt/datadir:/var/lib/mysql MySQL**

**Inspect Command:**

Below command will provide all details about the container ( like IP address and etc )

**#docker inspect nginx**

**Container logs:**

**#docker logs ngnix**

**Eg:**

**Benefit of having persistent volume :**

* Suppose we are running a jenkins container without mapping the persistent volume.
* If we stop the container and start container , the data is lost as it is saved under /var/lib folder within the container.
* Now create a container with mapping
* #docker run -p 8080:8080 -v /root/myjenkins-dir:/var/jenkins-home jenkins
  + This will make sure that the volume is persistent and data is stored under /root/myjenkins-dir.
* Now if we stop and start container again, the data is present on dir and it will not ask you to do everything from scratch
  + ( #docker run -p 8080:8080 -v /root.myjenkins-dir:/var/jenkins-home jenkins )

**Tags:**

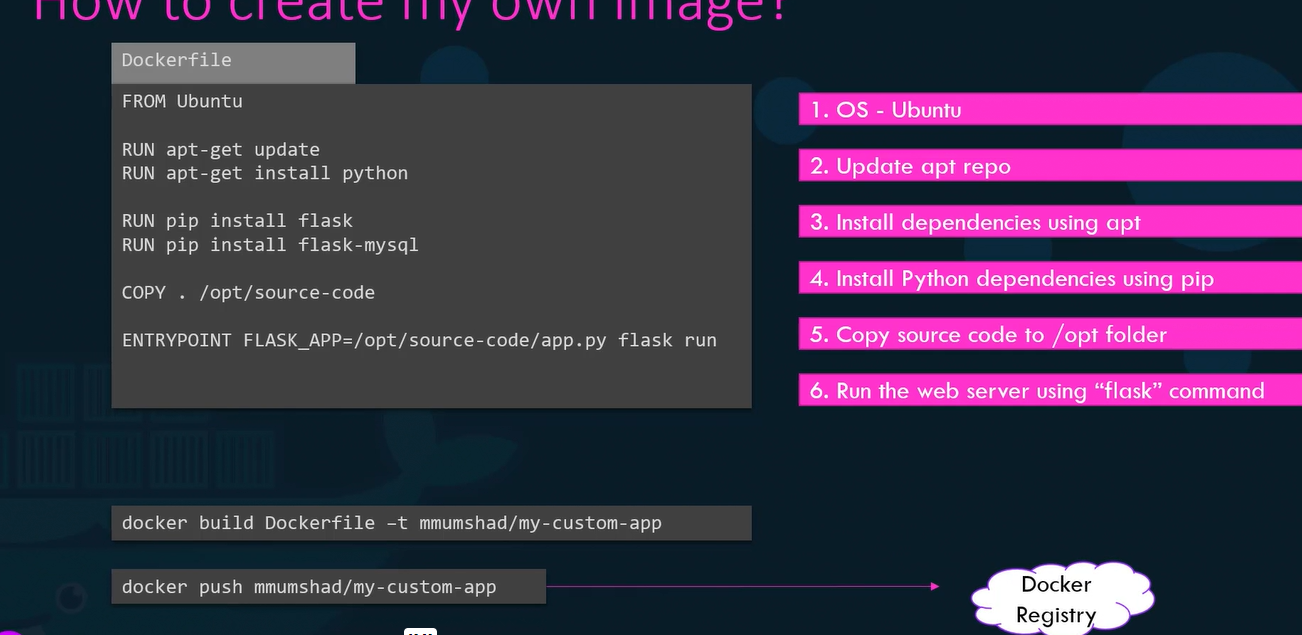
By default when we run "#docker run ubuntu" , it will install the latest version ubuntu container, however if we want particular version, we can use tags

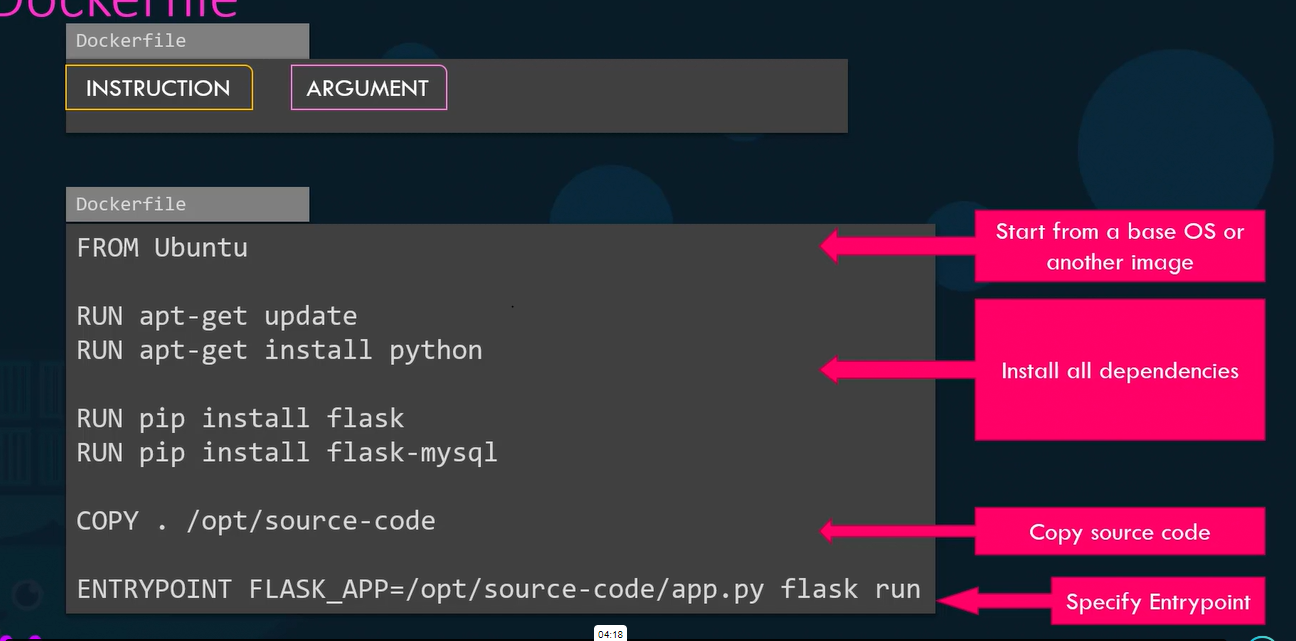
#docker run ubuntu:17.10

#docker run ubuntu:17.10 cat /etc/hosts

**Creating New Docker Image :**

Docker is a layered architecture , where when we are building an image it is built in layered format.





Once the docker file is created, we can run command

#docker build DockerFile -t samanth/my-app ------ this command will build the docker image.

To see the sizes of each layer ,how much it consumed while creating an docker image , we need to run below command

#docker history samanth/my-app

**Failure While building Docker Image:**

**Note** : Docker will store the cache content, which means.

* After running docker build command #docker build DockerFile -t samanth/my-app , While building docker image , if there is a failure or errors at installing dependencies point, we need to open docker file and fix the error,

And then when we re-run the same command #docker build DockerFile -t samanth/my-app , docker will start from the failure point as cache is saved.

* Similarly when we are planning to add few additional things in docker image we can just open the file and edit it, as docker saves the cache, it will try to install the whatever is newly added,
  + Instead of building everything from scratch.

Building Docker Image from existing container:

* Gather all information required like os , dependencies , entry point etc.
* Create a file "docker file"
* Provide all details as required
* Run #docker build -t web-app --------> images will be built without tag
* Run #docker images
* Run created image as container and check if all good ( #docker run webapp )
* Push it to repository (#docker push web-app)
  + When we do this we get permission error , so we need to rebuild using tag
* #docker build dockerFILE -t samanth/webapp

**Environmental Variables**:

To see any Env field of a running docker container provide command

#docker exec -it blue-app env

To run a docker container with particular env field provide below command

#**docker run -e APP\_COLOUR=blue --name=blue-app -d simple-web**

To run a docker container with mysql image and name as mysql-db and setting password of mysql to db\_pass123

#**docker run --name mysql-db -e MYSQL\_ROOT\_PASSWORD=db\_pass123 -d mysql**

**Docker Compose:**

This will allow us to run application stack , where if we want to deploy multiple applications at once ,which works together.

**Eg:**

We need to deploy an voting application which need

* Python ( front end )
* Redis ( in memory )
* Worker Node
* PostgreSQL ( stores DB)
* Result ( Node js )

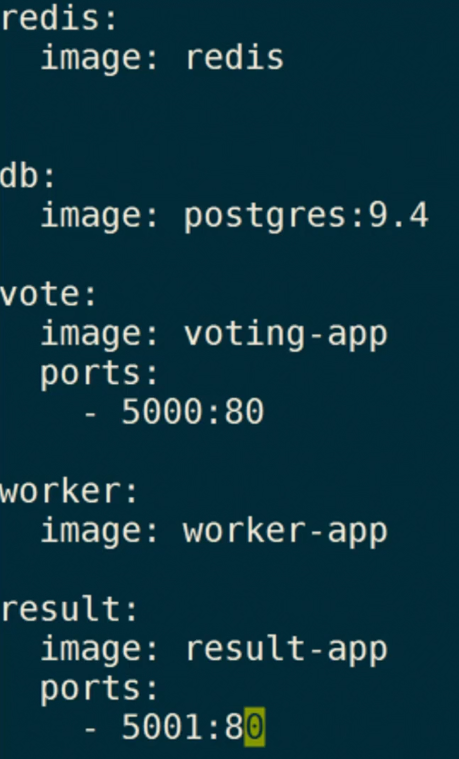
All these applications can be deployed at once using docker compose. We need to link them and ports need to be enabled for python and Result container as they are front end customer.

**Steps**:

Install Docker Compose

Create Dockercompose.yml

Write yaml file ------ However these are not the latest version of docker compose yml.



Once the yaml file is ready

* Go the location
* Trigger the command # docker compose up

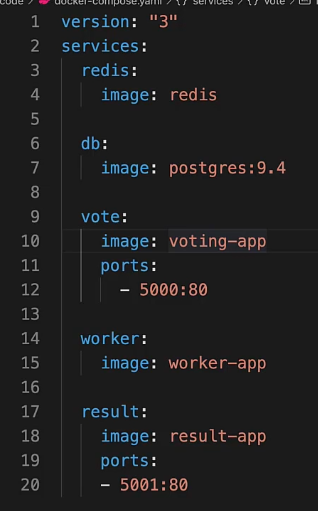
**Version 3 yml file:**

In the YML file we need to mention as version 3

Version 3 have feature of advance networking, where mentioning links in yaml ( like previous version ) is not necessary.

As this will automatically connect all to an network & dns Resolution.

Eg:



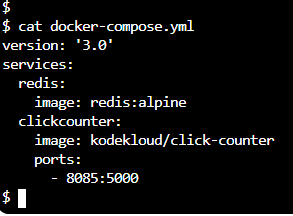
**Eg:**

This command will run container with name clickcounter and port host=8085,container=5000 and linked to diff container called redis

Link generally usefull if one container need to talk with another container

#**docker run --name clickcounter -p 8085:5000 --link redis -d kodekloud/click-counter**

**Eg:**



This Is how we can write docker yml file , once done we can run command

#docker-compose up -d

**Docker Volume:**

Docker store all files under the location

**/var/lib/docker**

If container is deleted/removed the data will be lost ,

so if we want to have a persistent volume , we need to create a volume and run it along with docker run command while creating container.

* create a volume in docker

**#docker volume create data\_volume**

* Docker will create volume under

**/var/lib/docker**

**/volumes**

**/data\_volume**

* We can mount it on container while creating a container.
  + **#docker run -v data\_volume:/var/lib/mysql mysql.**

* We can also have diff locations such has /opt/data or /efs/data, and these can be mounted similarly while creating a container
  + **#docker run -v /opt/data:/var/lib/mysql mysql**
    - **Or**
  + **#docker run -v /efs/data:/var/lib/mysql mysql**

**Note :** From now on all the data will be stored under /efs/data or /opt/data and the data is persistent.

**Or**

**#docker run \ --mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql**

**Docker Networking:**

We have 3 types In docker networking

* **Bridge ---- This is commonly used when we are creating multiple containers , containers will be having their own IP and PORT ,**

**and we need to MAP with HOST to make it publicly accessible.**

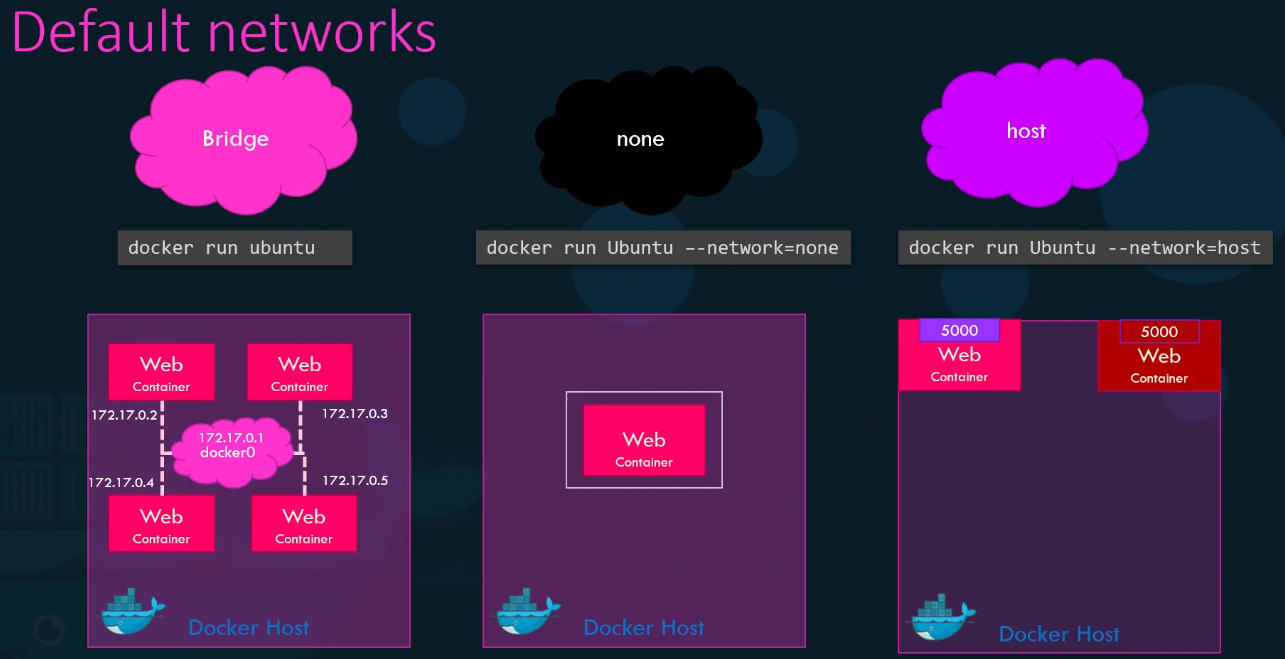
**#docker run ubuntu**

* **None --- this completely isolated container, it will be having either host or container network.**

**#docker run ubuntu --network=none**

* **Host ---- In this the container will be provided with a port and that port is common to Docker host and container, where we cannot use same port again for diff container.**

**#docker run ubuntu --network=host**



**User Defined Network:**

when we are creating containers , docker uses a default networking and same IP range is used for multiple containers like

Nginx = 172.20.1.2

Python= 172.20.1.3

But we can also have a different CIDR ranges/networks which we need to define ( user defined networks )

Note:

* All Containers can resolve each other with the name of the container.
* Docker runs with built in DNS server that has address 127.0.0.11

**Commands:**

This command will lis t networks iu.

**#docker network ls**

To view subnet configured on bridge network or host or none network

**#docker network inspect bridge**

This command will run alpine image with none network

**#docker run --name alpine-2 --network=none alpine**

Command to create a new network with bridge driver and subnet and gateway

**#docker network create wp-mysql-network --driver=bridge --subnet=182.18.0.1/24 --gateway=182.18.0.1**

Run Docker container with specific network driver and variable

**#docker network ls**

**#docker run --name mysql-db -e MYSQL\_ROOT\_PASSWORD=db\_pass123 --network=wp-mysql-network -d mysql:5.6**

Running a container with certain name, tagging to network , with diff variables, and allowing port and linking

#**docker run --network=wp-mysql-network --name webapp -p 38080:8080 -e DB\_Host=mysql-db -e DB\_Password=db\_pass123 --link mysql-db -d kodekloud/simple-webapp-mysql**

**Docker Registry:**

#docker login registryname

Running an registry as a container :

#docker run --name my-registry -p 5000:5000 --restart=always -d registry:2

docker pull nginx:latest

docker images

docker pull httpd:latest

docker image tag nginx:latest localhost:5000/nginx:latest

docker image tag httpd:latest localhost:5000/httpd:latest

docker push localhost:5000/httpd:latest

docker push localhost:5000/nginx:latest

To delete all Images locally

#docker image prune -a

