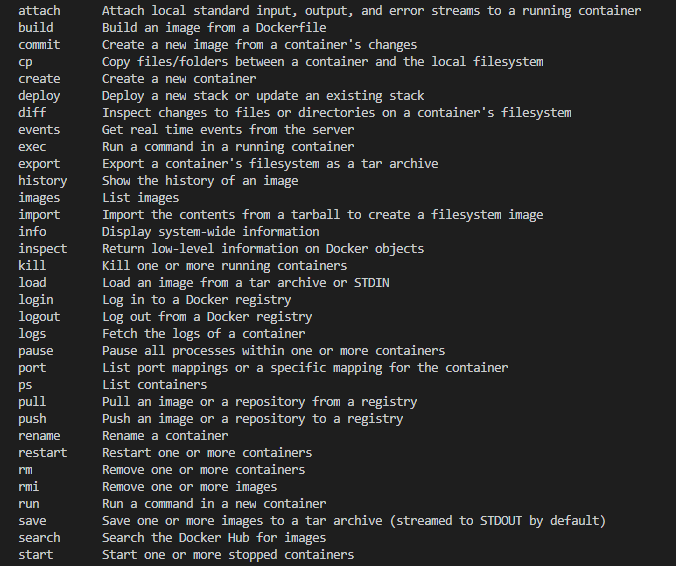
* Installation of Docker can be found here

<https://docs.docker.com/get-docker/>

* Docker has default image registry which is Docker Hub hub.docker.com
* image = binaries and lib, source code application which we want to run
* Container = is a running instance of that image running. we can have many containers running off the image
* Docker --version = verified cli can talk to engine
* docker info - info about images, containers and most config value of engine
* docker = this will list the commands



* Management cmd - docker cmd subcmd option = to organize cmd structure,



* docker cmd option = this is the old way of running command and it is still valid
* docker container run = this is the new way to run cmd
* docker run = initially this cmd was launch and still running

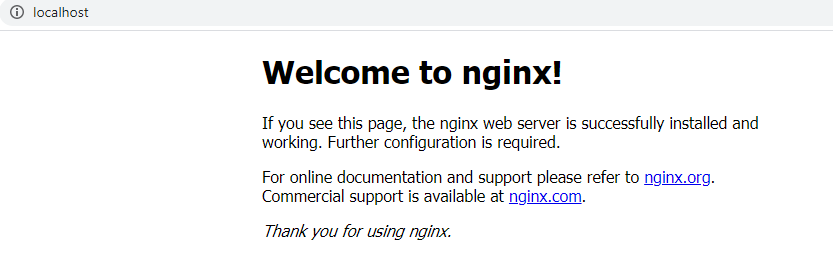
Example to running a container

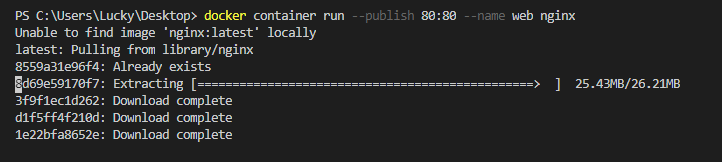
* **docker container run --publish 80:80 nginx** = this will launch a container image nginx for you at port 80 access. check it on local host

download image nginx from docker hub and started a new container.

--publish will expose your container to port 80. so, on left 80 is used to access on localhost.

if this 80 is used by some other prog in your machine use any other port 8080 or anything

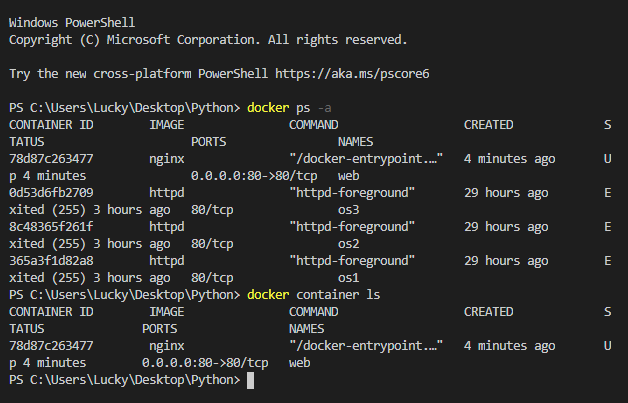




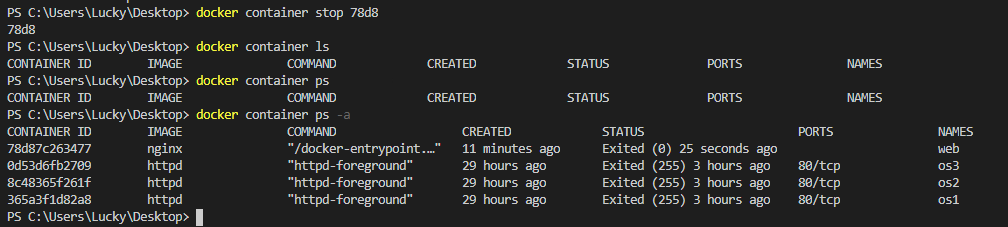
* docker container run --publish 80:80 --detach nginx = -d or(--detach )this is detach mode, will run in background.

**Docker container ls** = this will list running container

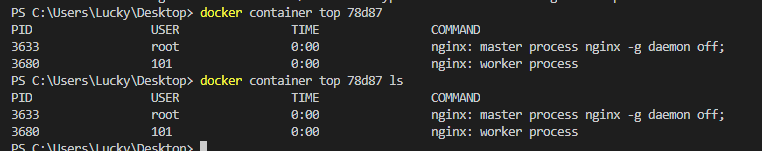
**Docker container ps -a** = This will show all your container running/stop



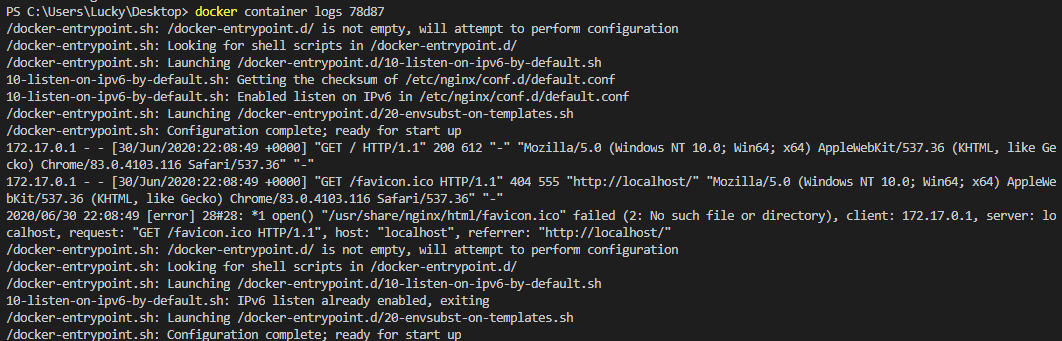
**docker container stop id** = this will stop your running container



**Docker container top id =** top to list the process ID running inside this



**Docker container logs id =** this will tell you about logs of your container



**docker container --help** = this will list help cmd associated with docker. This is the best resource for you

**docker rm -f id** = this will remove all container including a running one, forcefully

* what happens in docker container run

1. Looks for that image locally in image cache, doesnt find anything then look in remote image repo(dcokerhub)

2. Download the latest ver(nginx:latest)

3.Create new container based on that image and prepare to start

4. Gives it a Virtual IP on a private n/wk inside docker engine

5. Opens port 80 on host and frwds to port 80 in container

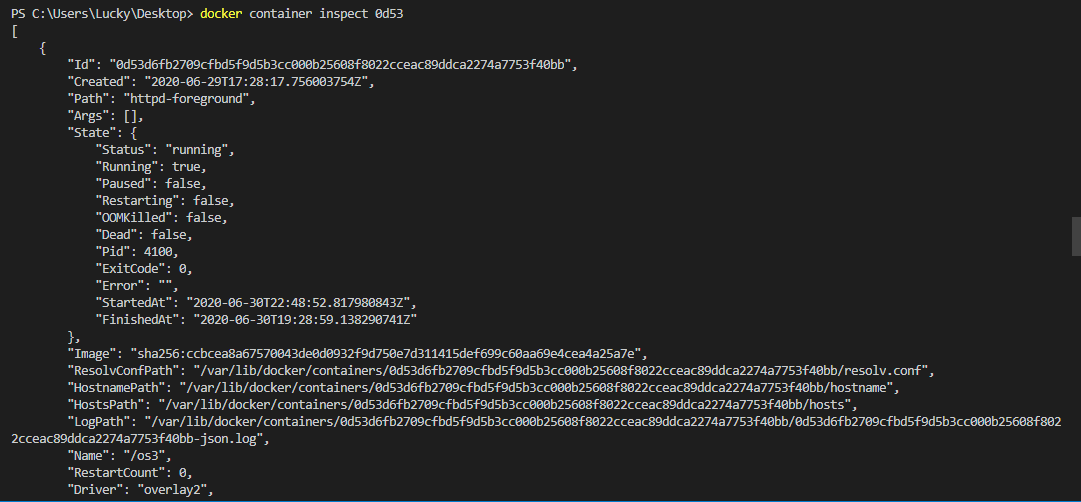
6. Starts container by using the CMD in the image Dockerfile

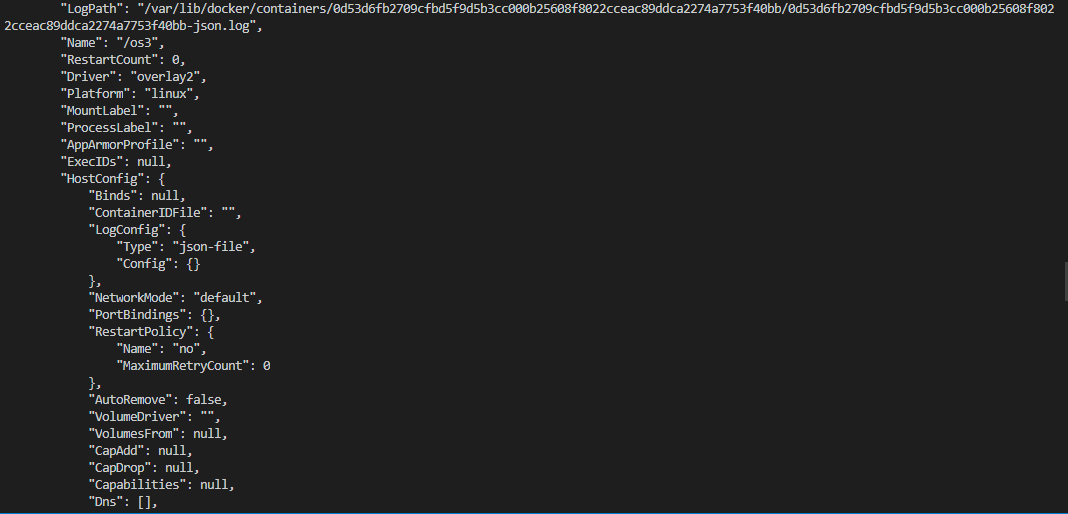
**Containers are not mini VM's,** They are process

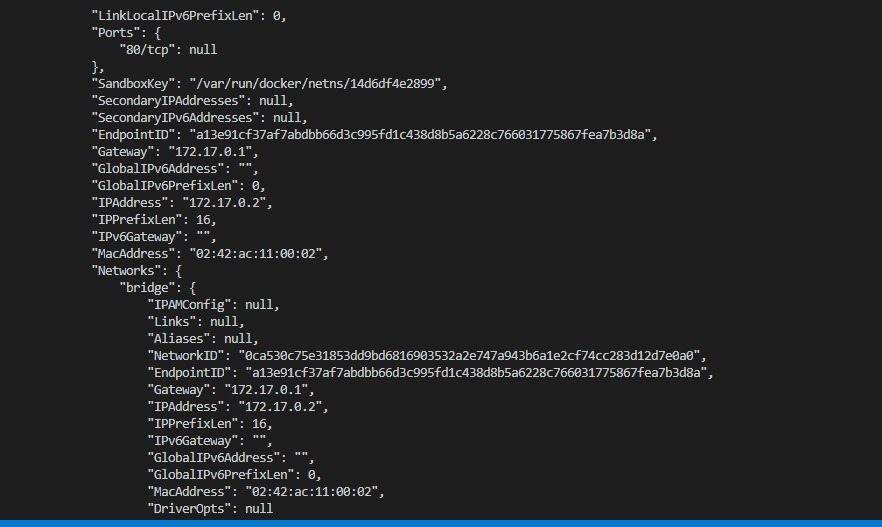
**docker run --name webhost -d httpd**

**docker top httpd** = One process of httpd will see here

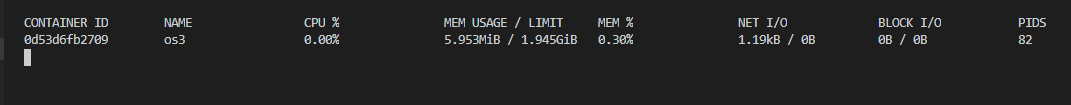
**docker container inspect id** = details of one container configuration,







docker container stats id - performance stat for container



**docker pull alpine** = alpine is small security-focused distro. this is very small image just in few 6mb. this doesn’t has bash shell in the image so they have sh cmd to enter

inside the image. So, we can use sh cmd to go inside this container. This has a package manager call apk so in case you need to install this you can do.

**Docker Networking**

Each container connected to a private virtual network "Bridge", each virtual network routes through NAT firewall on hostIP

All container on a virtual network can talk to each other

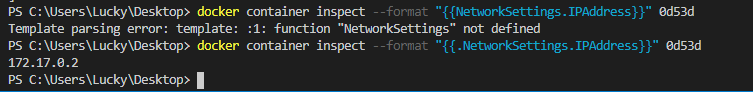
Best practice is to create new virtual network for each app:

**network "internal app'' for internal app**

**network "external app" for external app**

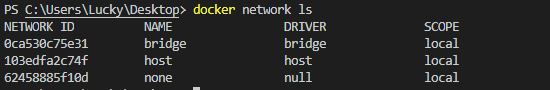
**docker container inspect --format"{{ .NetworkSettings.IPAddress}}" webhost =** format is use to format the output of cmd using Go Template. This will give you IP details. remember

inspect will give you an output in json format, and if you scroll down to check your ip would be mention there.



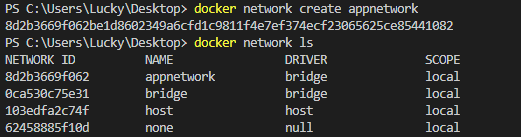
**Netwokrs - cli management**

**docker network ls** = show networks



**docker network inspect** - inspect a network, here I inspect a network name bridge

**docker network create** - create a network, here created a network appnetwork

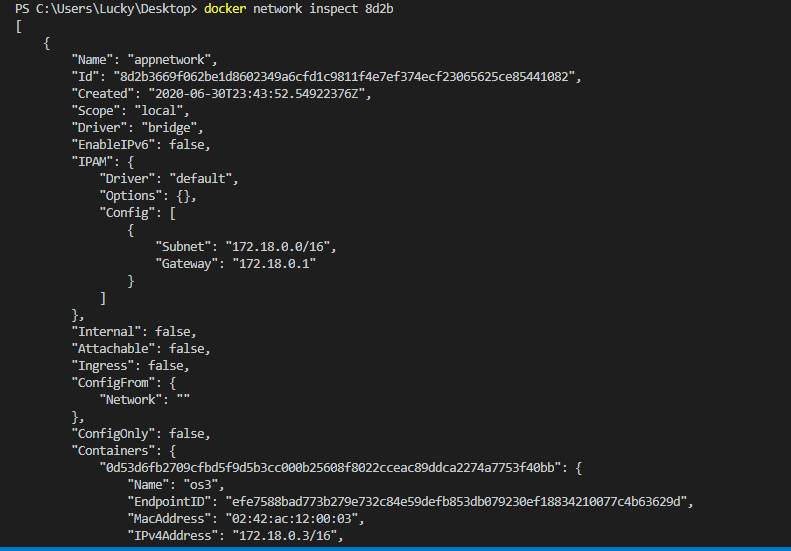


**docker network connect** - attach a n/w

**docker network disconnect** - Detach a n/w from container

**docker container run -d --name nginx\_new --network appnetwork nginx** = launching a new container in the appnetwork

**docker inspect my\_app\_net** = inspect this new network attach with new container with IP



docker network connect - Dynamically creates aNIC in a container on an existing virtual network

**Docker Networks - DNS**

docker has built-in DNS server that containers use by default.

<https://dyn.com/blog/dns-why-its-important-how-it-works>

<https://docs.docker.com/config/formatting>

docker run -dit --name centos centos:7 = install centos 7 image and run container

docker container exec -it centos bash = go inside bash and work inside os

# yum update curl -y = installed curl

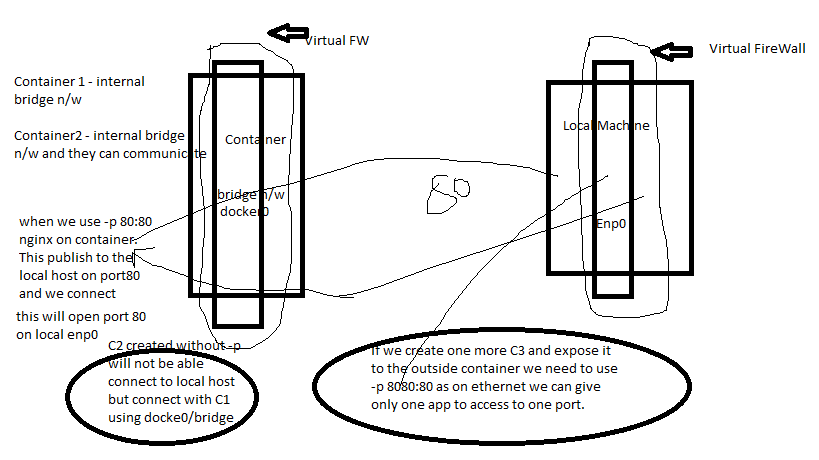
# curl --version = check version

**docker container exec -it ubuntu bash** = go bash in ubuntu

apt-get update & apt-get install curl = installed curl and update repo

curl --version = check version

<https://www.digitalocean.com/community/tutorials/package-management-basics-apt-yum-dnf-pkg>



**Images -**

* This contain app binaries and dependencies. Metadata about the image data & how to run the image
* It is not a complete OS. No kernel, kernel modules (e.g drivers)
* Small as one file (your app binary) like a golang static binary
* Big as a ubuntu distro with apt, and Apache , PHP and more

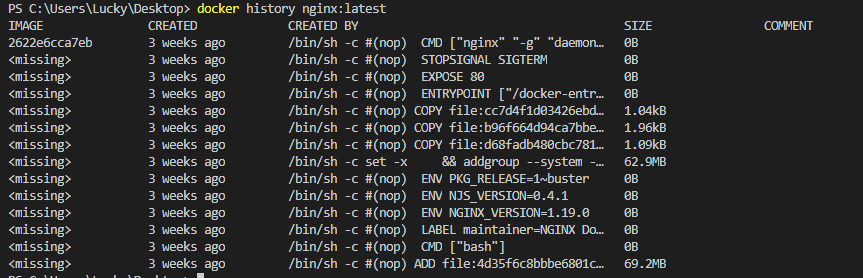
**Docker Hub**

* This has official image and another user created one
* They have complete documentation
* The official one has version updates with tag.
* **docker pull nginx** = this will get latest from hub

**Images and layers**

* It uses union file system. They are the prebuilt configuration files to run the container process.
* So, images are made up of file system changes and metadata.
* Each layer is uniquely identified and only stored once on a host
* This saves storage space on host and transfer time on push/pull
* A container is just a single read/write layer on top of images

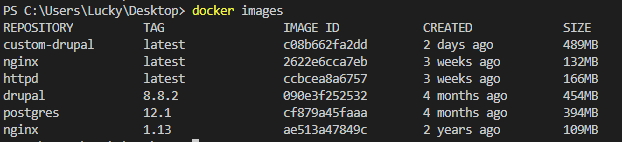
**docker history nginx:latest** = history command will tell you what happed with this Image, what changes done on Dockerfile.



**docker inspect image nginx** = this will give info about metdata of this image

ex architecture, default logon files, version.

**docker image or docker images ls**



**docker image tag nginx himashu/nginx =** This is to tag an image. Here I have tagged nginx to himashu/nginx

**docker image push himashu/nginx** = This is to push to your Docker hub account. Ensue you have Docker Hub account created already and you need to provide your id/pass

**docker image –help** = you can check what all option available with this command. Best Resource

**docker prune images** = this will clean up dangling images, do not use this in production

**docker system prune** = this will clean up everything, do not do this in production

**Volumes** – <https://docs.docker.com/storage/volumes/>

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

**Persistent Data - Bind Mounting**

* Maps a host file or directory to a container file or dir
* Basically 2 location pointing to same files
* can't use in Dockerfile must be at container run

**docker container run -d --name nginx1 -p 80:80 -v ${pwd}:/usr/share/nginx/html nginx**

this command will create a volume at /usr/share/nginx/html for the container and also at pwd of your host system. So whatever you write in the volume of container this will be update at host location

**Ex, Create one docker file(Dockerfile) and placed this in your current login dir**

**FROM nginx:latest** = # this shows how we can extend/change an existing official image from Docker Hub

**WORKDIR /usr/share/nginx/html** = # highly recommend you always pin versions for anything beyond dev/learn

# change working directory to root of nginx webhost

# using WORKDIR is preferred to using 'RUN cd /some/path'

**COPY** index.html index.html

**docker container run -d --name nginx -p 80:80 -v ${pwd}:/usr/share/nginx/html nginx**

**Docker compose Template format : -** [**https://docs.docker.com/compose/compose-file**](https://docs.docker.com/compose/compose-file)

version: '3.1'  # if no version is specified then v1 is assumed. Recommend v2 minimum

services:  # containers. same as docker run

  servicename: # a friendly name. this is also DNS name inside network

    image: # Optional if you use build:

    command: # Optional, replace the default CMD specified by the image

    environment: # Optional, same as -e in docker run

    volumes: # Optional, same as -v in docker run

  servicename2:

volumes: # Optional, same as docker volume create

networks: # Optional, same as docker network create

cat .\docker-compose.yml

version: '3'

services:

proxy:

image: nginx:1.13 # this will use the latest version of 1.13.x

ports:

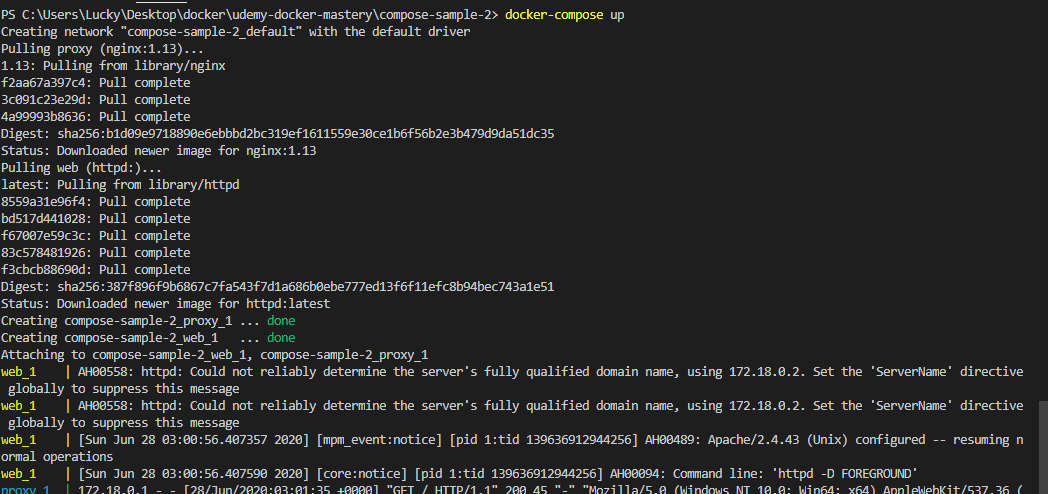
- '80:80' # expose 80 on host and sent to 80 in container

volumes:

- ./nginx.conf:/etc/nginx/conf.d/default.conf:ro

web:

image: httpd # this will use httpd:latest



**Docker Compose CLI -** <https://docs.docker.com/compose/reference/>

this tool is for dev/test env not a prod fav tool

docker-compose up = setup volumes/networks and start all containers

docker-compose down = stop all container and remove cont/vol/net

**Here is my simple docker compose yaml file to deploy httpd server using a proxy nginx server for the web server.**

cat .\docker-compose.yml

version: '3'

services:

proxy:

image: nginx:1.13 # this will use the latest version of 1.13.x

ports:

- '80:80' # expose 80 on host and sent to 80 in container

volumes:

- ./nginx.conf:/etc/nginx/conf.d/default.conf:ro

web:

image: httpd # this will use httpd:latest

**docker-compose up** # this will run the yaml file and deploy 2 container with networking and all configuration done using docker-compose

check your result using localhost:80