**cDocker Notes and Hands on**

**Types of Hypervisors:**

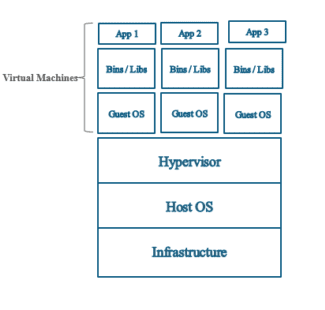
Type1: Runs on top of the hardware directly (BareMetal)

Example: Vmware esxi, HyperV

Type2: Runs on top of the OS installed

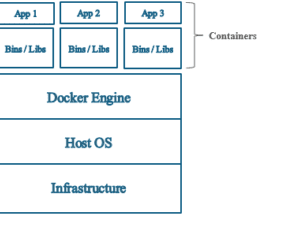
Example: Virtual box

As the below Image for a virtual machine we need an OS to be installed and on top of it we can run application. Where as in docker we require only the dependencies of the application for it to run refer image below.



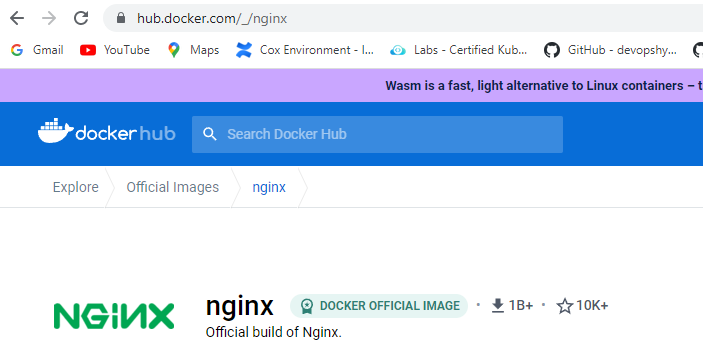
**Docker:**

Docker is a software which helps us to create container. Containers are the running instances of an application. Container runs in isolation whatever changes we make inside the container it will not impact the host OS. Container has its own IP address, MAC address etc. In a layman’s term container are miniature OS which has everything of itself, and all the resources of the container are borrowed from the HOST OS. Refer below Image to see the difference between Virtual machine and container.



**DockerHub:**

Whenever we try creating any container first docker will look for an image on our local computer if it not available it will search on docker hub to download the image and from that image it will create a container. By default, docker will looks for an image in docker hub if its not available locally on our computer. But we can also provide our local repository as well for docker to search for any images



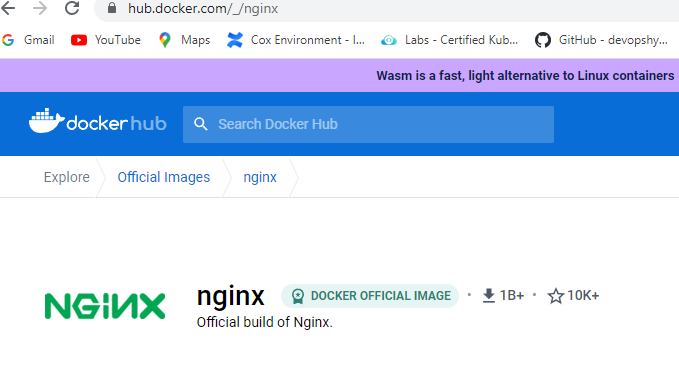
**Docker File:**

Docker file is a plain text file without any extensions it has all the dependencies to create a particular Image.

**Docker Image**: From the mentioned dependencies in the docker file we create a docker image. There are two kinds of docker images, first are those publicly available and we can download it from the Docker Hub.

Example of Public Docker Image:

Nginx, Jenkins, etc.



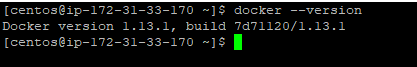
Second are the customized images which we can create using docker file

**Docker Container**: Containers are created from the docker images

**Docker Installation:**

**Docker was originally designed for Linux based server but from 2017 it is available to be downloaded to windows and MAC OS i.e., Docker Desktop.**

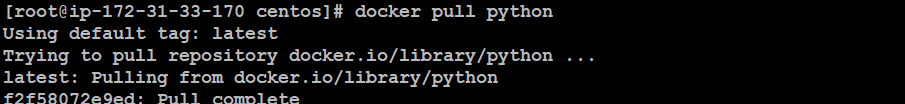
1. **sudo yum install -y docker (Use this command to install docker on Centos/Redhat**
2. **docker - - version ( To check the version of Docker)**



**Docker Commands:  
Note: Apart from docker - - version any operation command of docker requires elevated rights i.e. (Sudo, Root) privileges.**

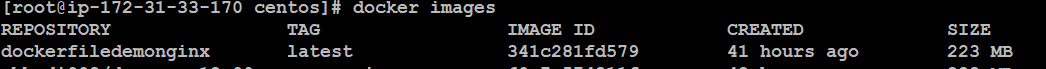
To pull any image from docker hub for the container use the below command:

Command: docker pull python



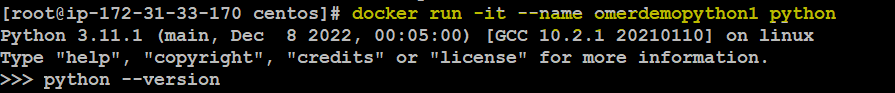
To check the images downloaded on our local computer use the below command:

Command: docker images



To create the container in interactive terminal (Once the container is created it will directly log in the container) we use option -it as the command below

Command: docker run -it --name omerdemopython1 python

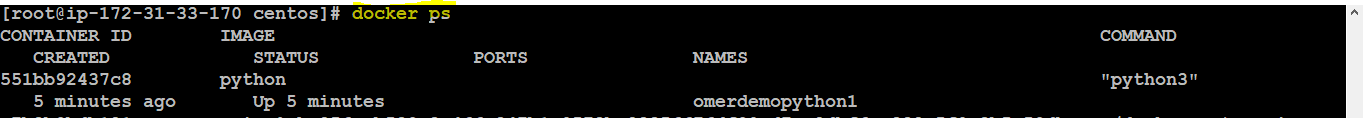


To exit out of the container without stopping it we press Ctrl+p+q Note: if you press exit, ctrl c or any other key it will stop the container after you exit out of it



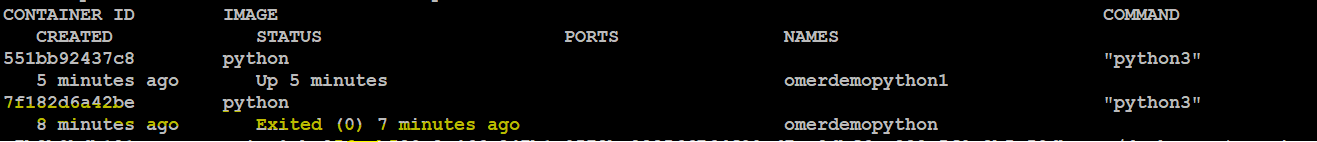
To check the running container in our system we use below command:

Command: docker ps



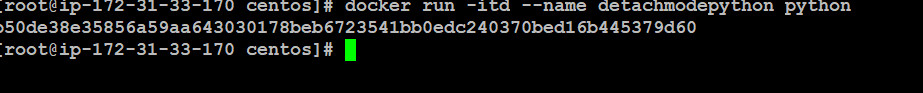
To list out all the container irrespective if its running or not use the below command:

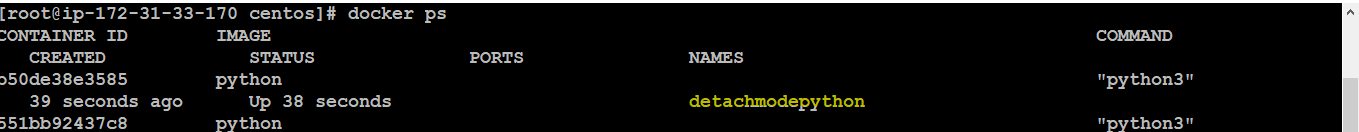
Command: docker ps -a



To create a container but not get into the container use the below command it is achieve by adding the flag -itd(Interactive termincal detach)

Command: docker run -itd –name detachmodepython python

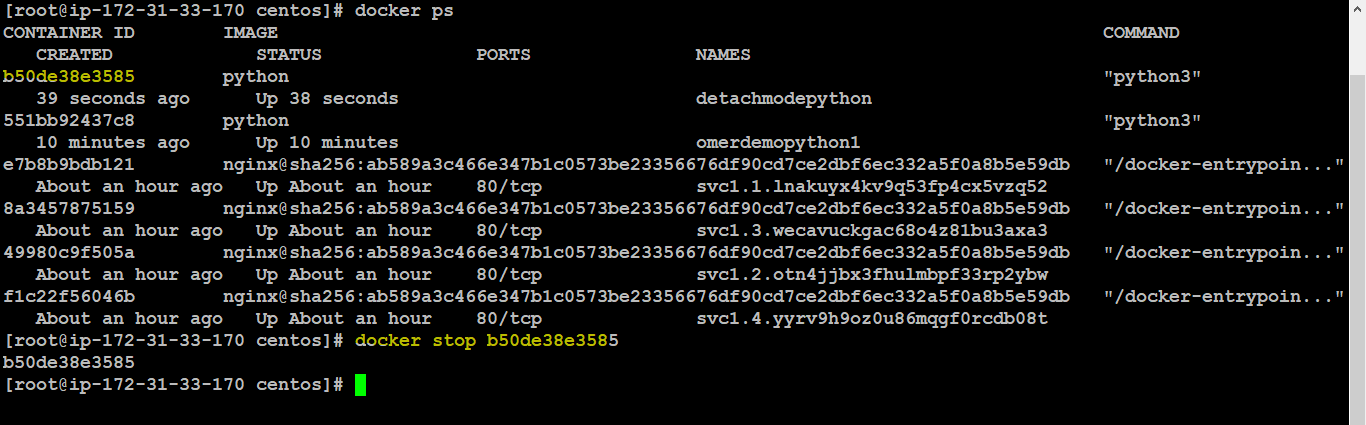




To stop any create use the below command

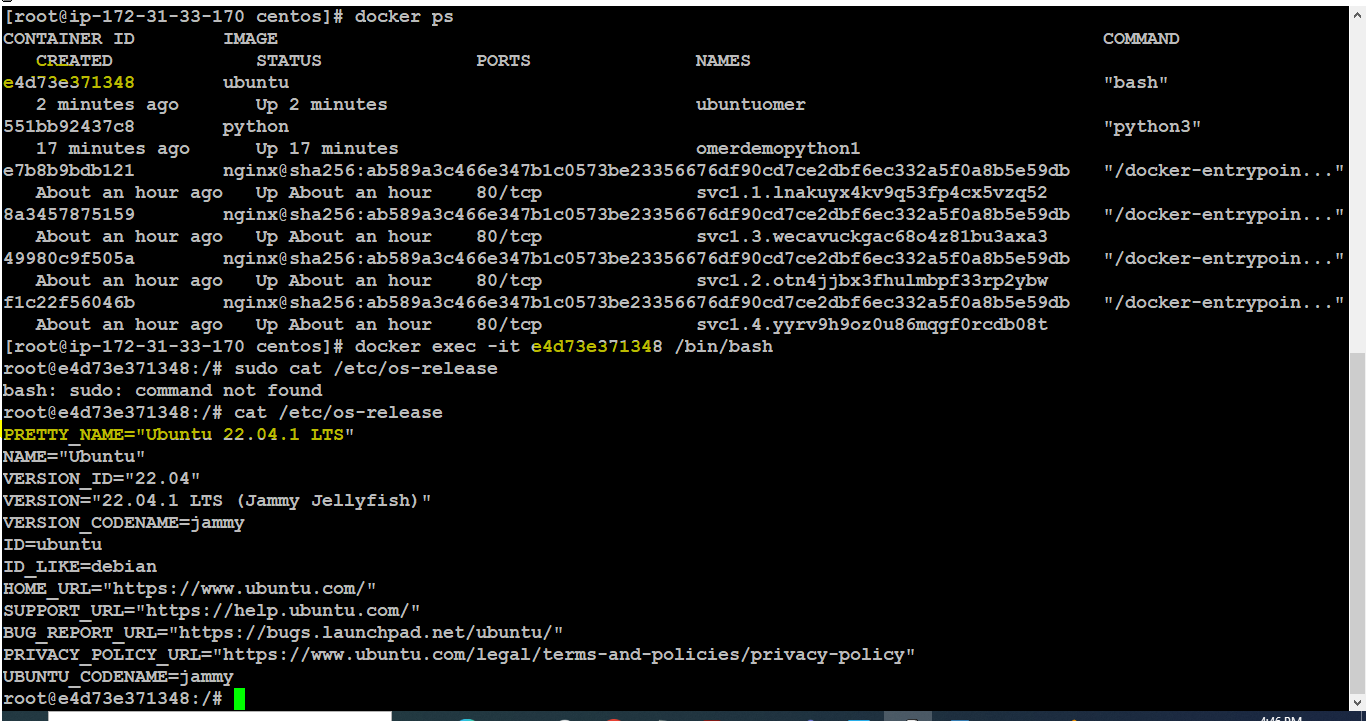
Command: docker stop containerID

Note: we can use start, restart option in the same way



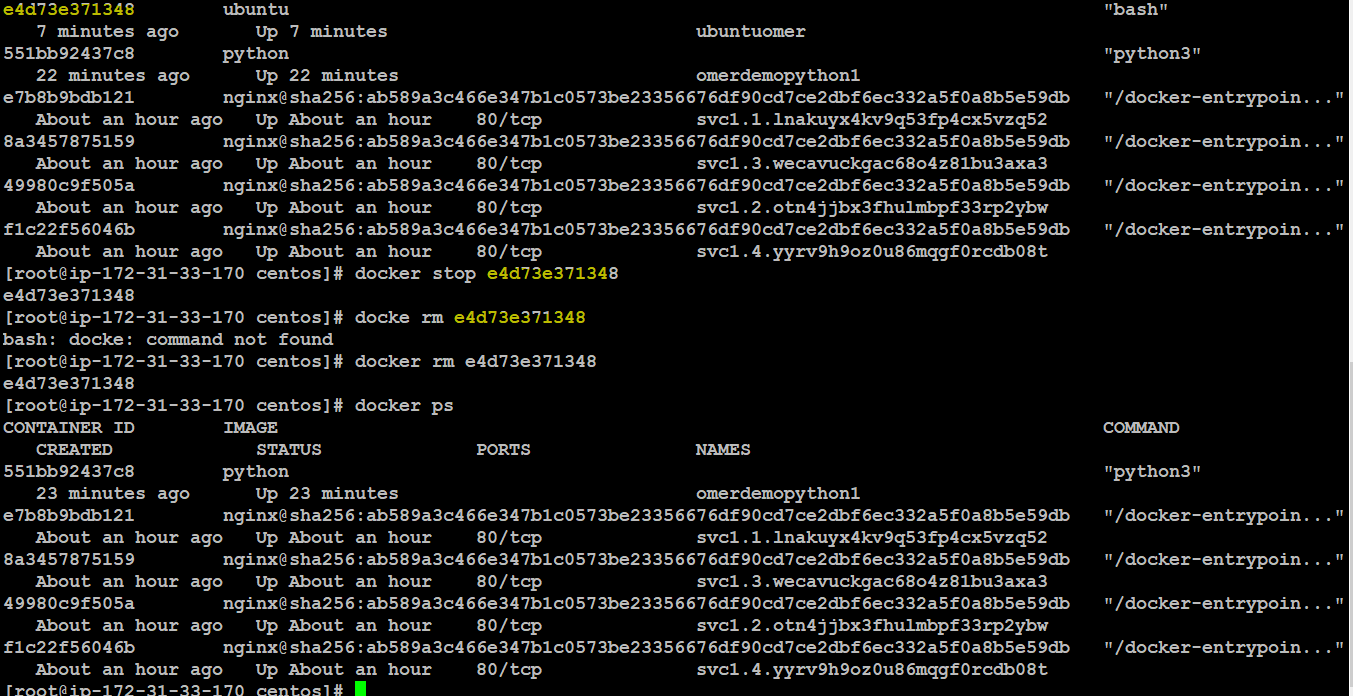
To get inside the container which is running already use the below command:

Command: docker exec -it container id \bin\bash



To delete the container, use the below command: Note: Never delete the container which is in running state always stop the container and then delete it

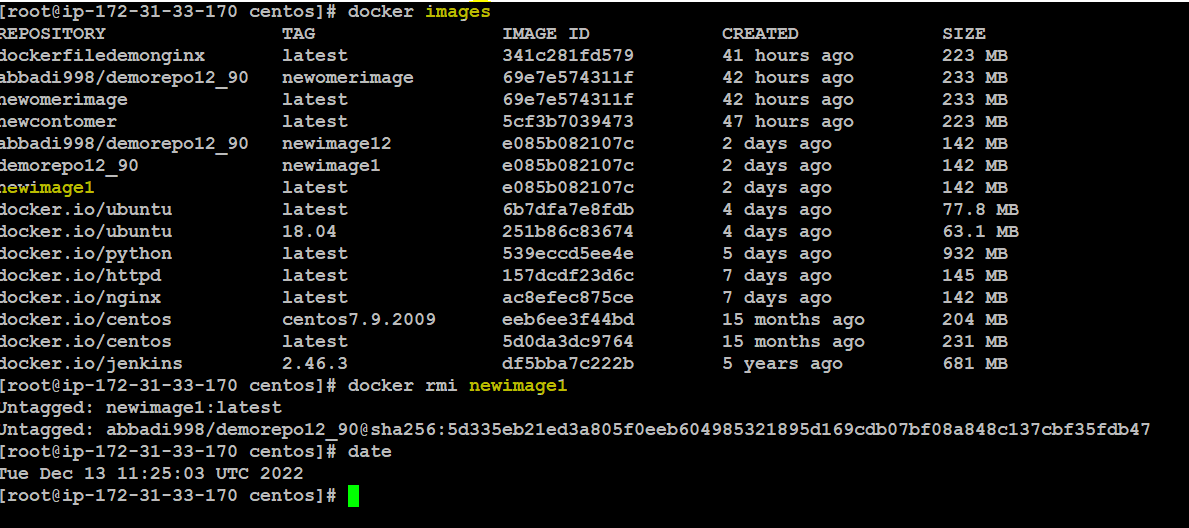
Command: docker rm containerID



To remove the image from local computer use the below command:

Note: before you delete the image make sure it is not used by any running container

Command: docker rmi imagename



**Port Forwarding:**

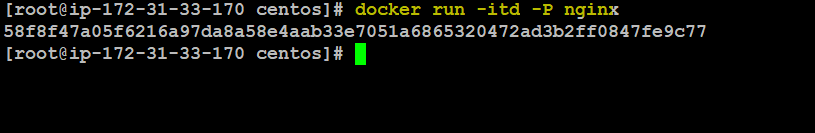
**As we know already now that the container will have its own Ip address for a container to expose to outside world or accessible outside our VM we can use either port forwarding or Binding port.**

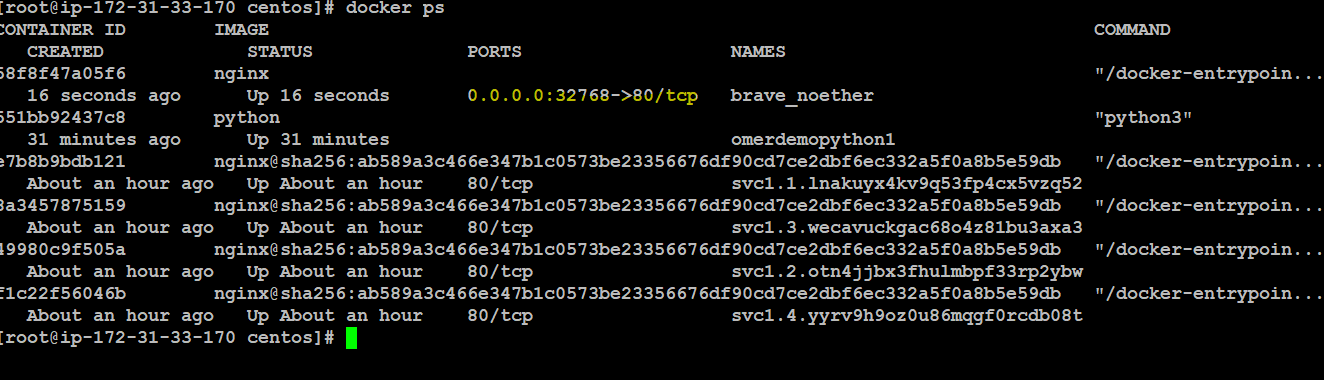
**Port forwarding will auto assign the port number to our container when created. It will be assigned by docker from the range 32768-65535(Any random port number in between)**

**Port Forwarding Example:**

**Note: we use flag P (uppercase for port forwarding)**

**Command: docker run -itd -P nginx**

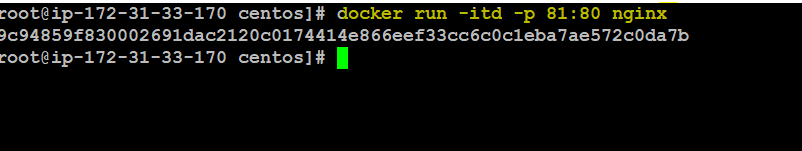


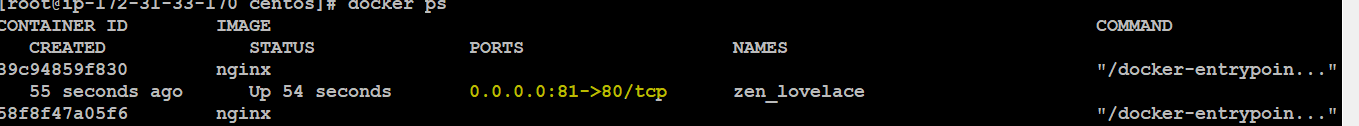


**BindingPort:**

**Example**

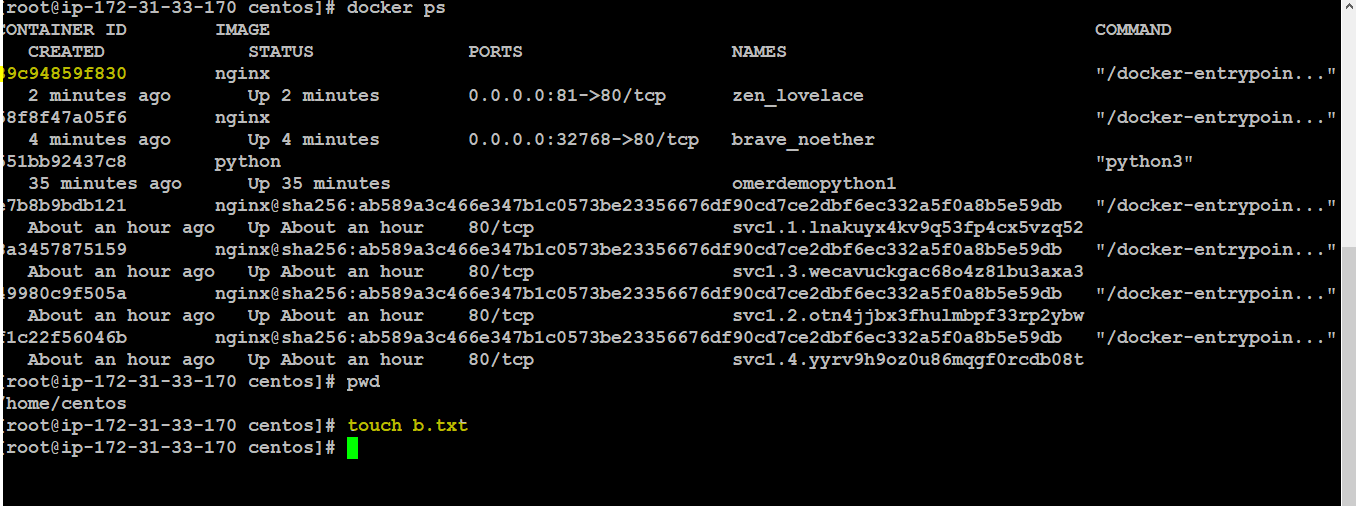
**Command: docker run -itd -p 81:80 nginx (81 is the outside port and 80 is external default port for nginx)**

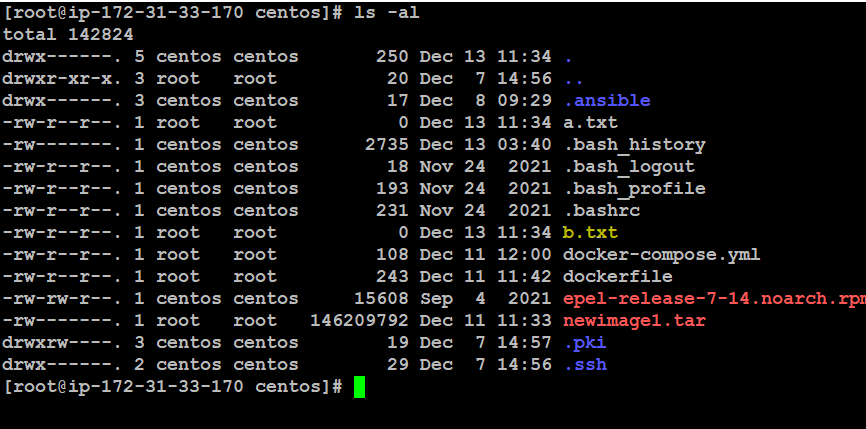


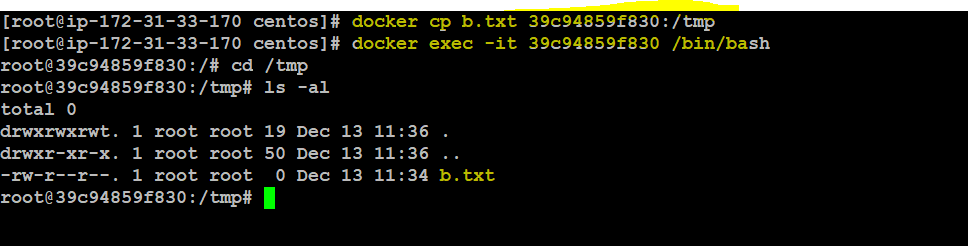


**DockerCopy Command to copy file from local computer to a container**

**Command: docker cp b.txt containerid:/tmp**

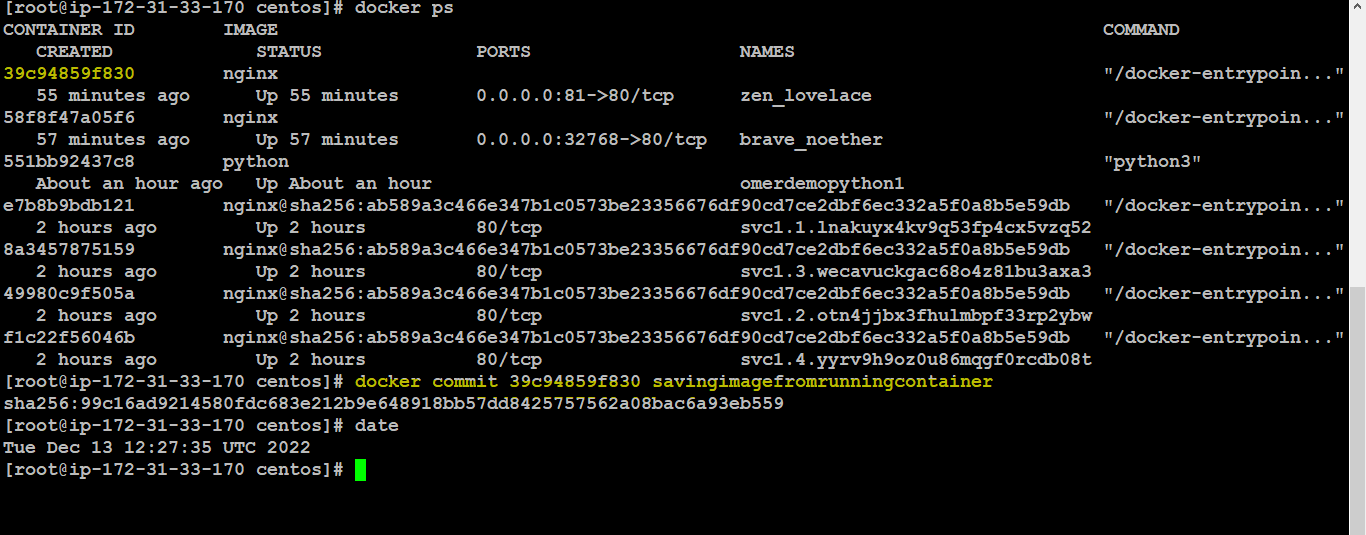


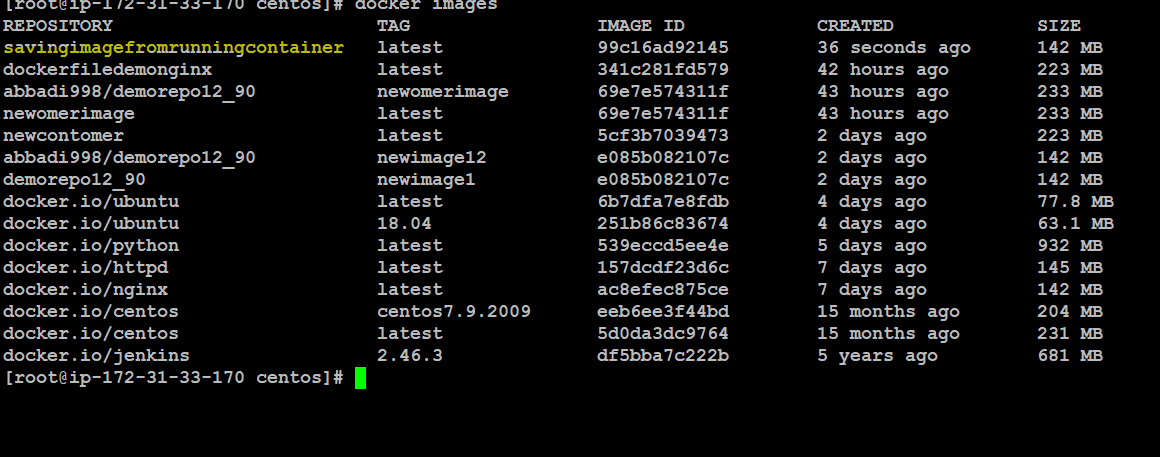




**To save an image from the running container use the below command**

**Command: docker commit ContainerID ImageName(OfYourChoice)**





**How to push(store) Images in DockerHub**

**In order to save any image in docker hub it must change to the format suggested by dockerhub**

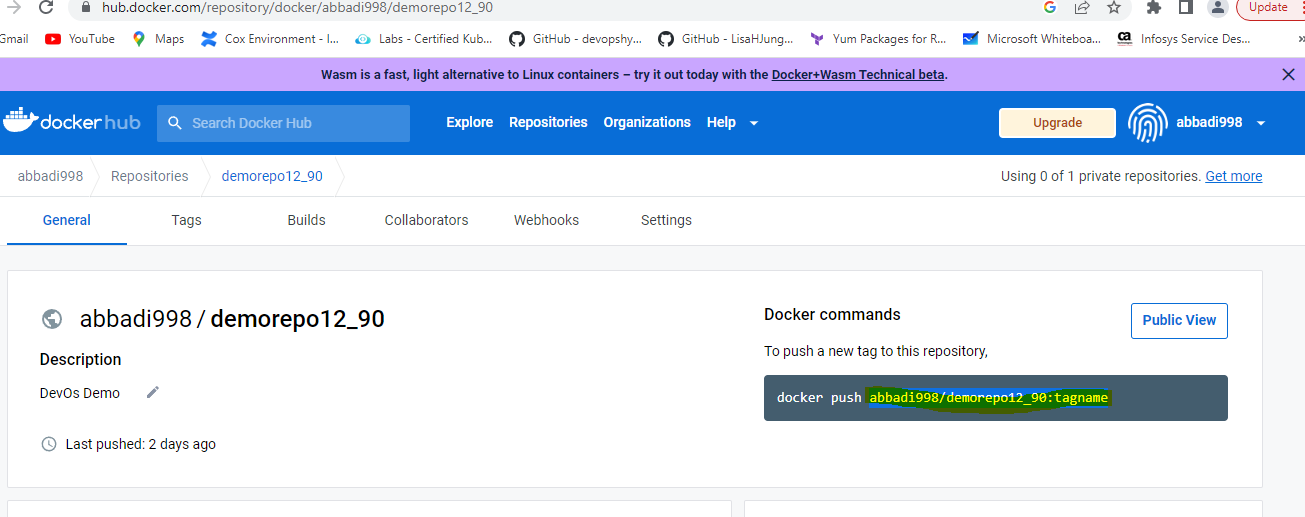
**For Example:**

**As highlighted in the image below the mentioned format is accepted by dockerhub for it to store any image from our local computer**

**Format:** abbadi998/demorepo12\_90:tagname

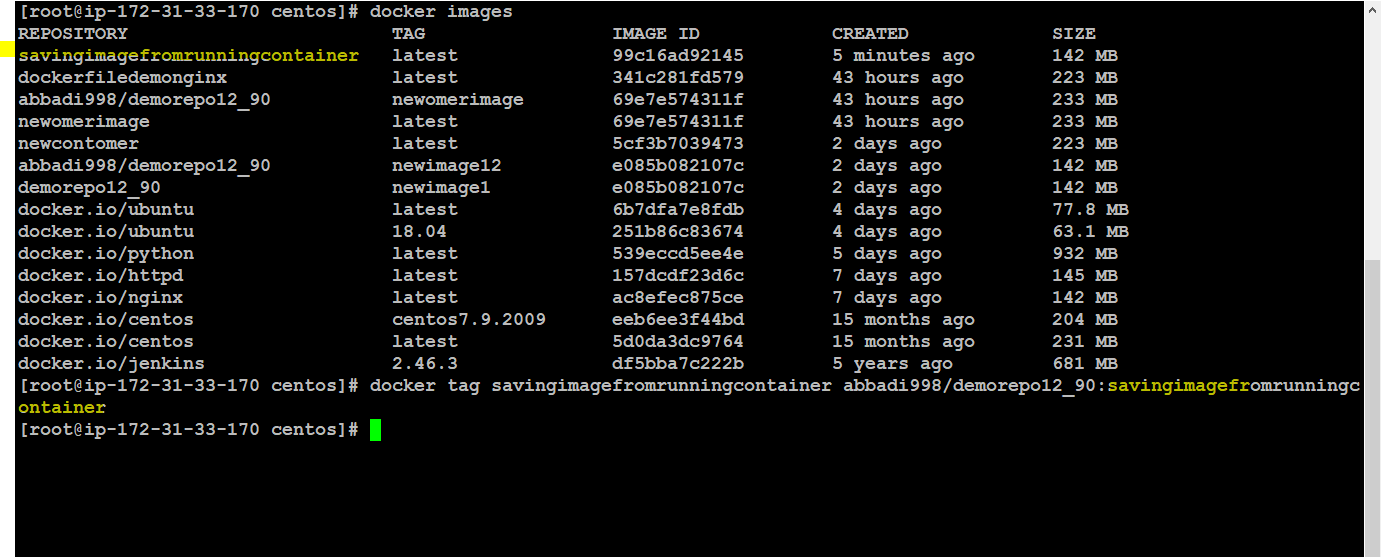
**So here the default format should be**

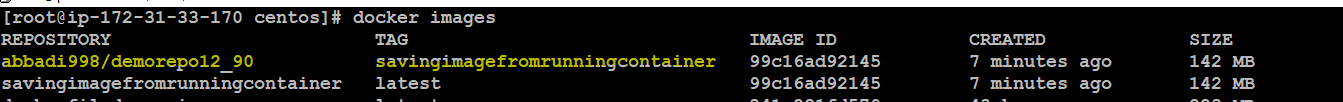
**abbadi998/demorepo12\_90:tagname(OurImageNameSavedOnLocalComputer)**



**In order to check the image name to the format mentioned above we use the below command:**

**Command: docker tag savingimagefromrunningcontainer abbadi998/demorepo12\_90:savingimagefromrunningcontainer**

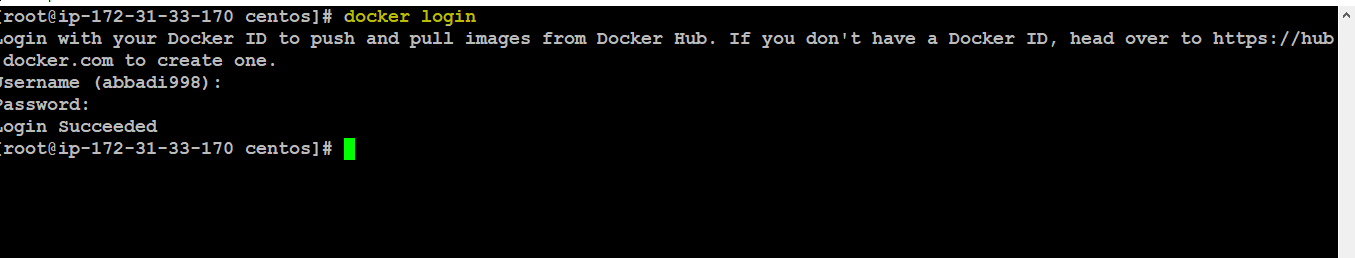




**In order to push our images to DockerHub we have to login to Docker first**

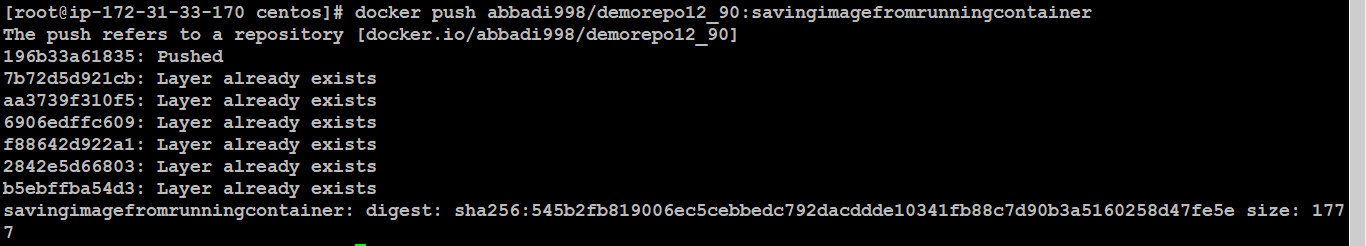
**Use the below command to login to DockerHub (Enter the username and password of your dockerhub login**

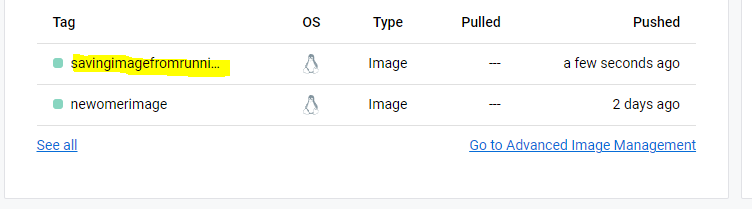
**Command: docker login**



**To push the image use the below command after you logged in to the DockerHub**

**Command: docker push ImageNameConvertedInTheMentionedFormat**



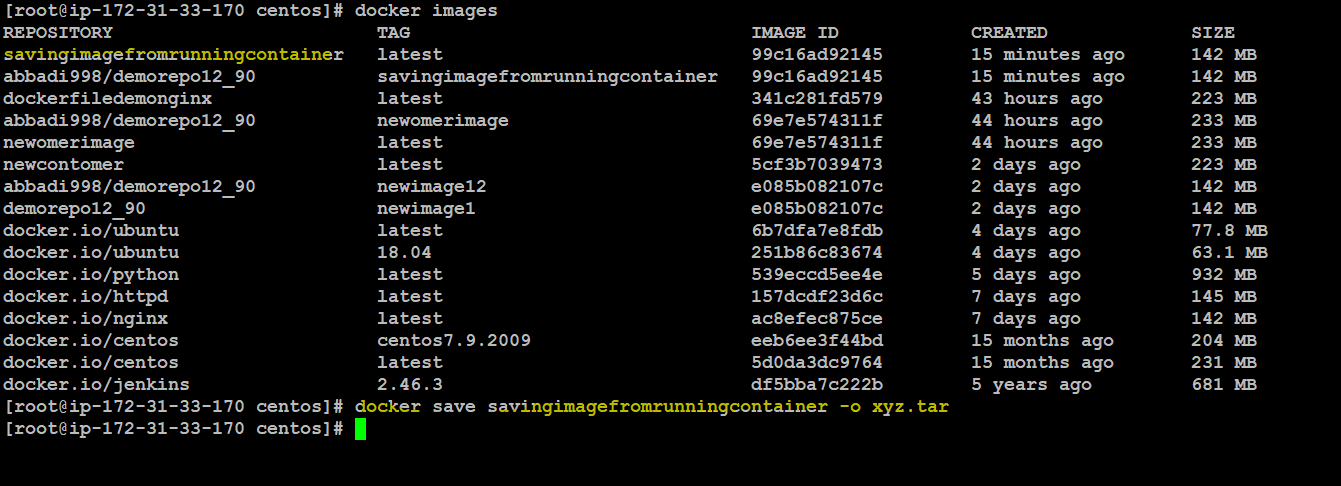


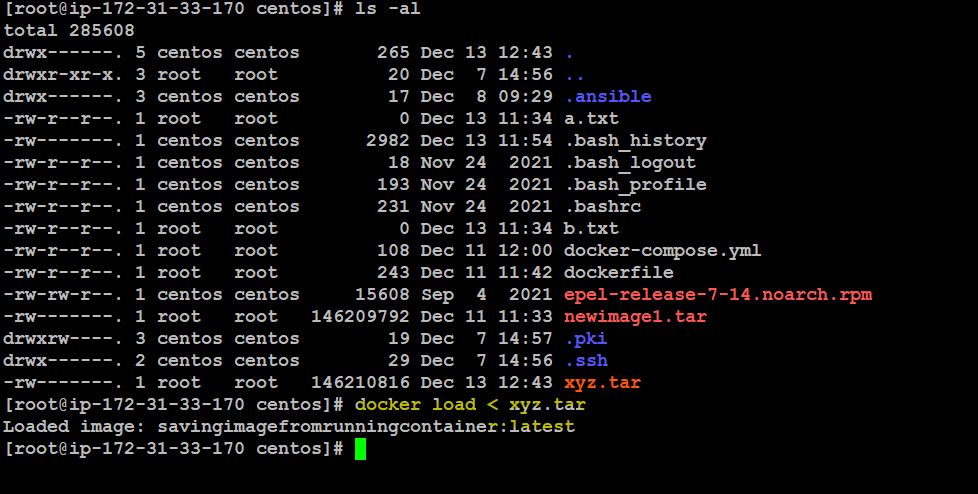
**TO convert the docker image into a Torball use the below command**

**Command: docker save ImageName -o xyz.tar(It can be any name).**

**To convert above tar file into a docker image use the below command**

**Command: dockerload < xyz.tar**





**To build an image from scratch use the dockerfile(default name for the file) without any extension**

**Command: vi dockerfile**

**FROM ubuntu:18.04**

**RUN apt-get update**

**RUN apt-get install -y vim**

**RUN apt-get install -y nginx**

**RUN rm /var/www/html/\***

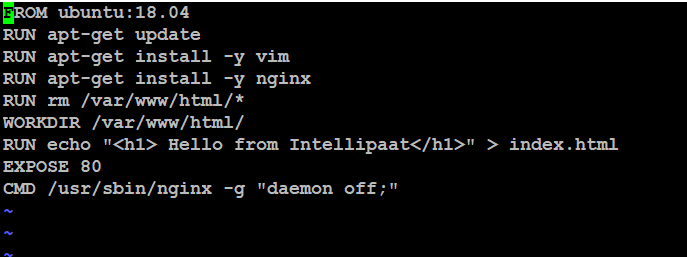
**WORKDIR /var/www/html/**

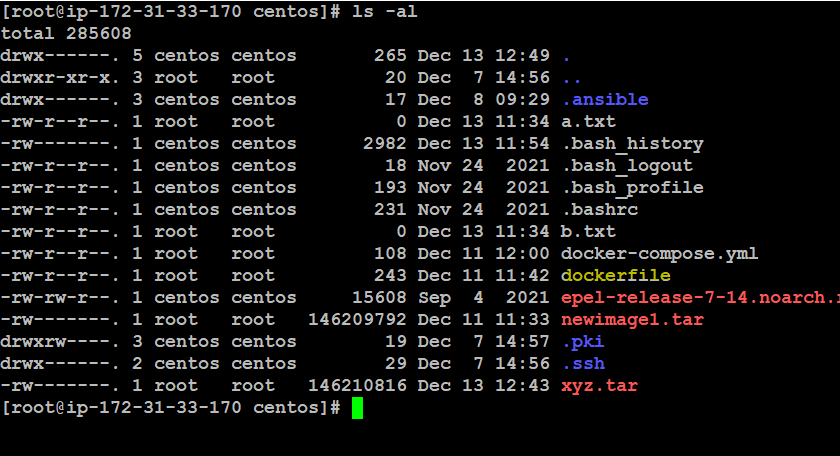
**RUN echo "<h1> Hello from Intellipaat</h1>" > index.html**

**EXPOSE 80**

**CMD /usr/sbin/nginx -g "daemon off;"**

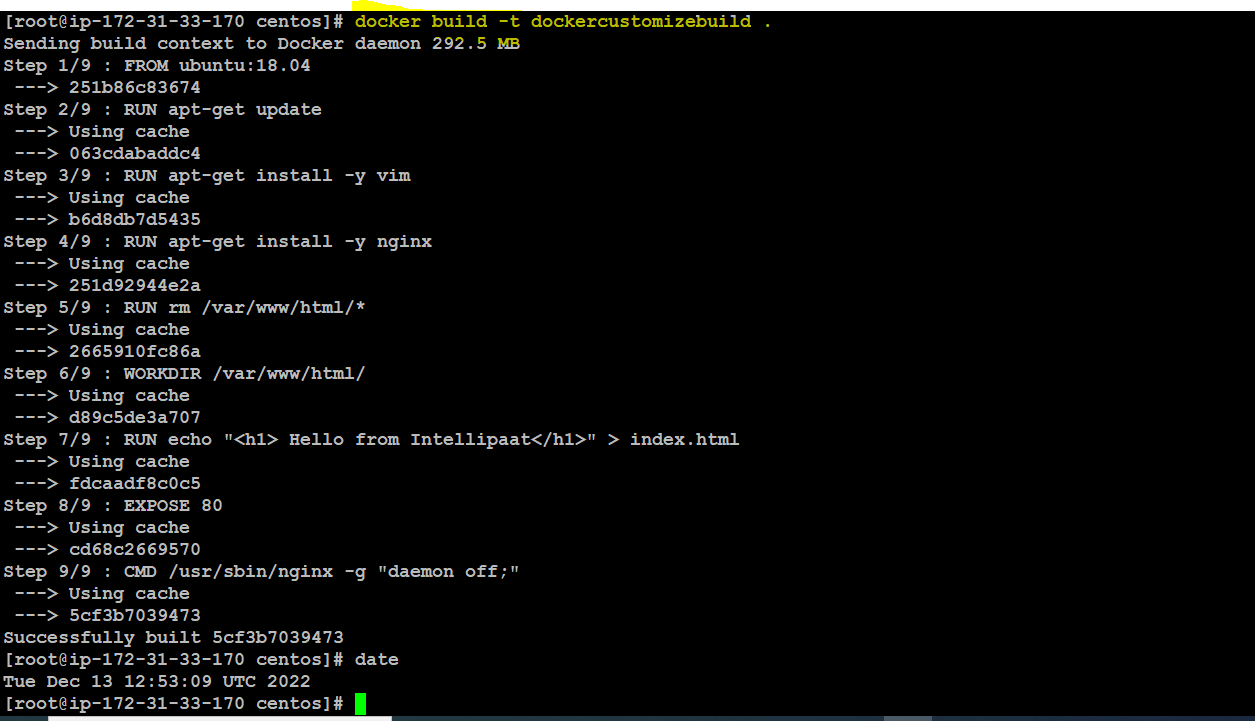
**Press Esc, Shift :, wq!(to save the file)**

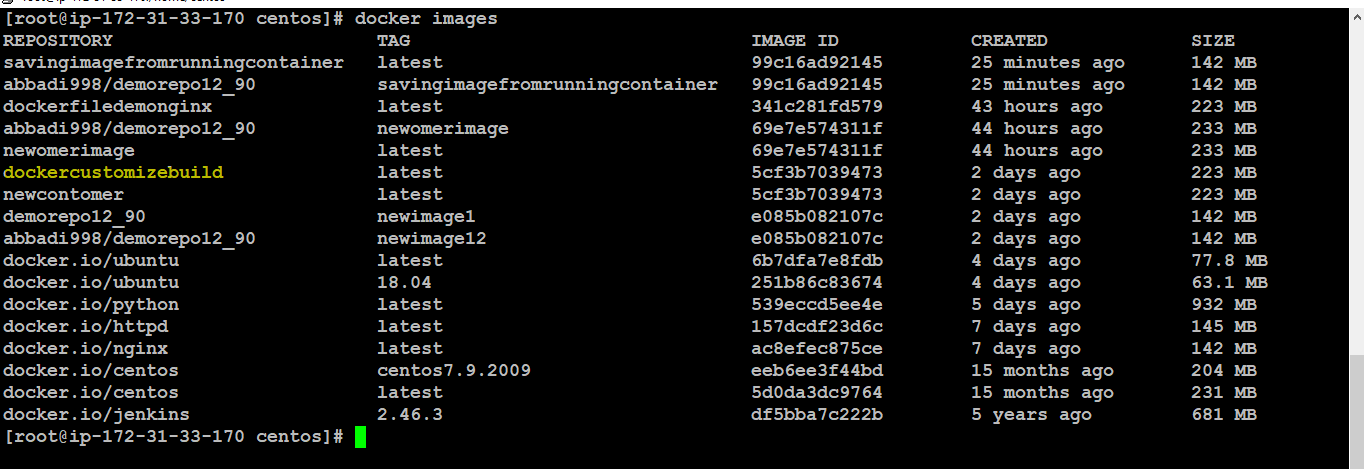




**To build the image from the docker file use the below command:**

**Command: docker build -t ImageNameOfUrChoice .(We use dot to tell docker to search for docker file in the current directory if you have the docker file saved in some other location please provide the exact path for the same)**





**DockerCompose:**

**We use docker compose as using the docker run command we can only build one container what if I have to build multiple container those scenarios we can use dockercompose plugin to do that**

**Docker Compose Installation:**

**So by default when you install docker docker compose will not be installed on our linux machine unlike docker desktop. We have to install docker compose plugin as standalone in our linux machine. Use the command below to install docker compose on Centos Machine**

**Command:**

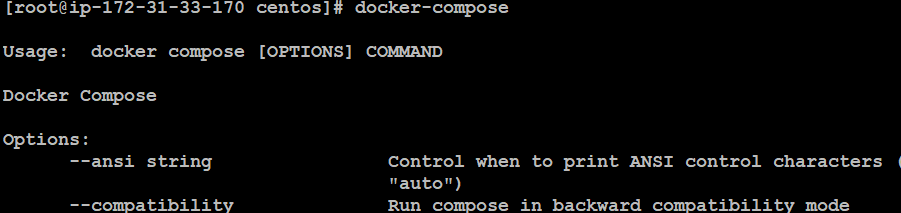
**curl -SL https://github.com/docker/compose/releases/download/v2.14.0/docker-compose-linux-x86\_64 -o /usr/local/bin/docker-compose**

1. Apply executable permissions to the standalone binary in the target path for the installation.

**Chmod +x /usr/local/bin/docker-compose**

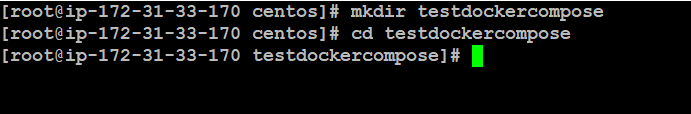
**TO check if its installed run the below command**

**Command: docker-compose**



**Docker Compose Example:**

**Create a directory by the testdockercompose and get into the directory by cd command like shown in the image below**



**Create a file app.py with the below code just copy and paste the code in the file and save the file**

**vi app.py**

**import time**

**import redis**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**cache = redis.Redis(host='redis', port=6379)**

**def get\_hit\_count():**

**retries = 5**

**while True:**

**try:**

**return cache.incr('hits')**

**except redis.exceptions.ConnectionError as exc:**

**if retries == 0:**

**raise exc**

**retries -= 1**

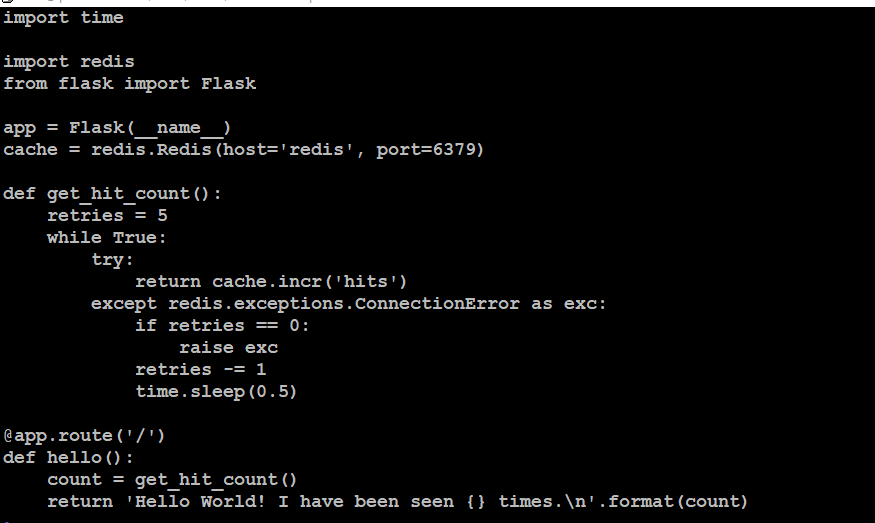
**time.sleep(0.5)**

**@app.route('/')**

**def hello():**

**count = get\_hit\_count()**

**return 'Hello World! I have been seen {} times.\n'.format(count)**



**Create requirements.txt with the below details and save the file**

**vi requirements.txt**

**flASK**

**redis**

**Create the dockerfile in the current directory without any extension**

**vi dockerfile**

**FROM pythOn:3.7-alpine**

**WORKDIR /code**

**ENV flask\_app=app.py**

**ENV FLASK\_RUN\_HOST=0.0.0.0**

**RUN apk add --no-cache gcc musl-dev linux-headers**

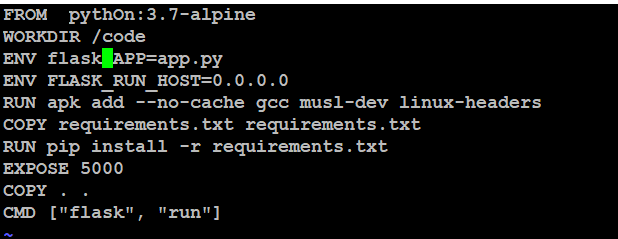
**COPY requirements.txt requirements.txt**

**RUN pip install -r requirements.txt**

**EXPOSE 5000**

**COPY . .**

**CMD ["flask", "run"]**



This tells Docker to:

* Build an image starting with the Python 3.7 image.
* Set the working directory to /code.
* Set environment variables used by the flask command.
* Install gcc and other dependencies
* Copy requirements.txt and install the Python dependencies.
* Add metadata to the image to describe that the container is listening on port 5000
* Copy the current directory . in the project to the workdir . in the image.
* Set the default command for the container to flask run.

**Create a docker-compose.yml file with the below content and save the file**

**vi docker-compose.yml**

**version: "3.9"**

**services:**

**web:**

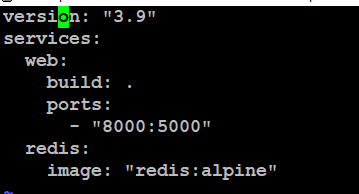
**build: .**

**ports:**

**- "8000:5000"**

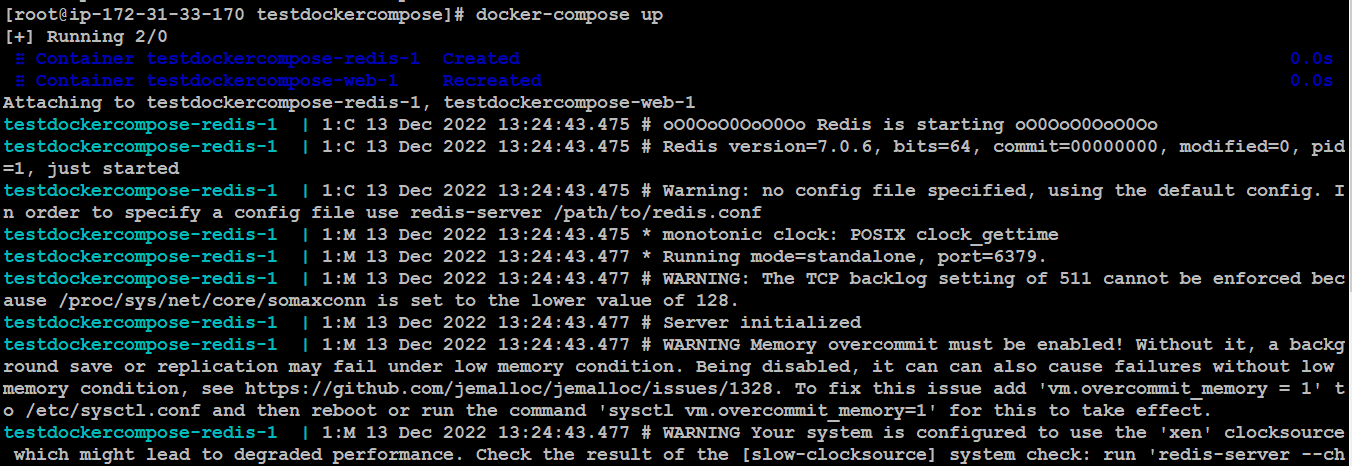
**redis:**

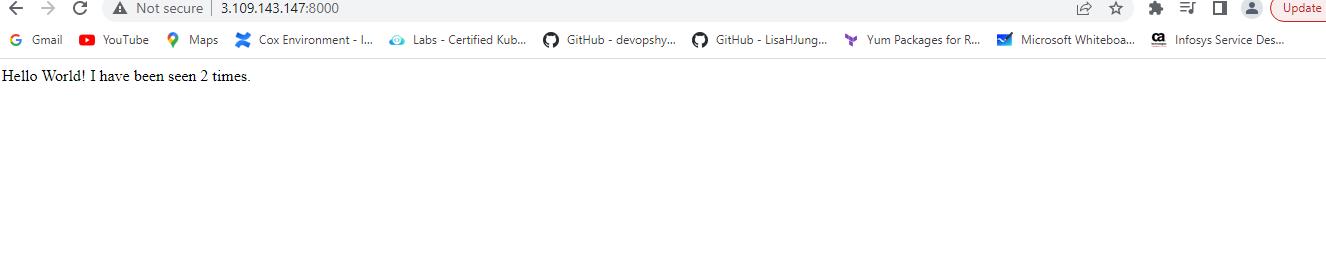
**image: "redis:alpine"**



**Use the below command to execute the docker compose**

**Command: docker-compose up**





**To check the output, enable the port 8000 on your ec2 instance and use your public ip along with 8000 port as shown in the image above**



