***Docker Container***

# Introduction

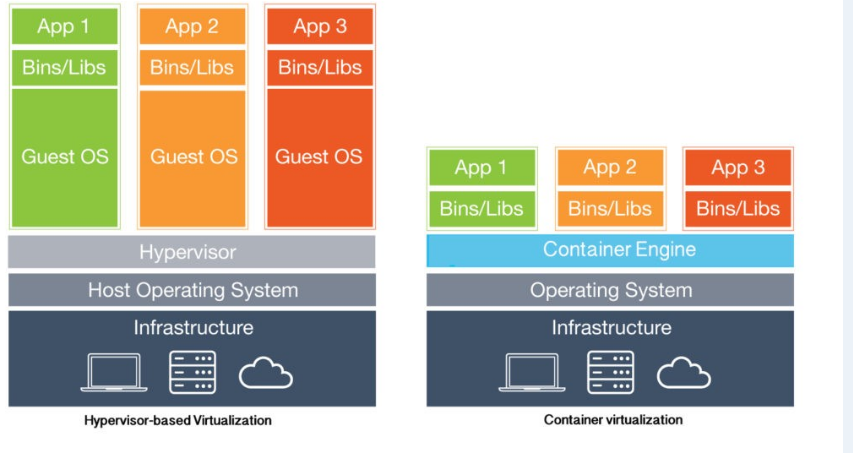
***What is Virtualisation?***

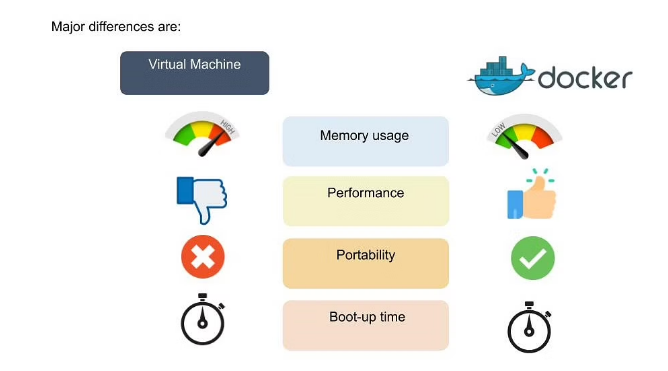
Virtualization is the fundamental technology that powers cloud computing. Virtualization is software that manipulates hardware, while cloud computing refers to a service that results from that manipulation. You can’t have cloud computing without virtualization.

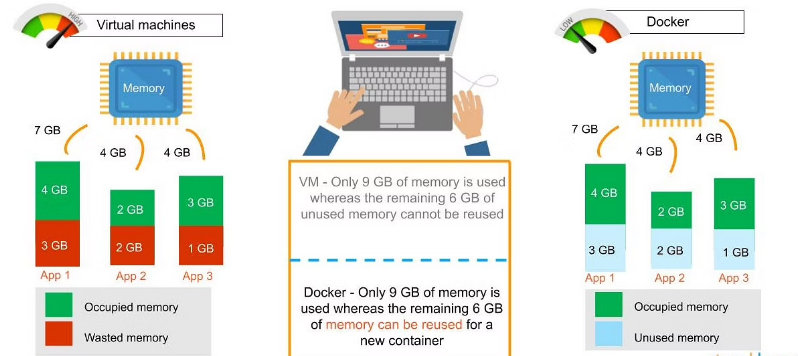
***What Is Container?***

Containers are lightweight software packages that contain all the dependencies required to execute the contained software application. These dependencies include things like system libraries, external third-party code packages, and other operating system level applications. The dependencies included in a container exist in stack levels that are higher than the operating system.

***Virtual Machine vs Containers?***

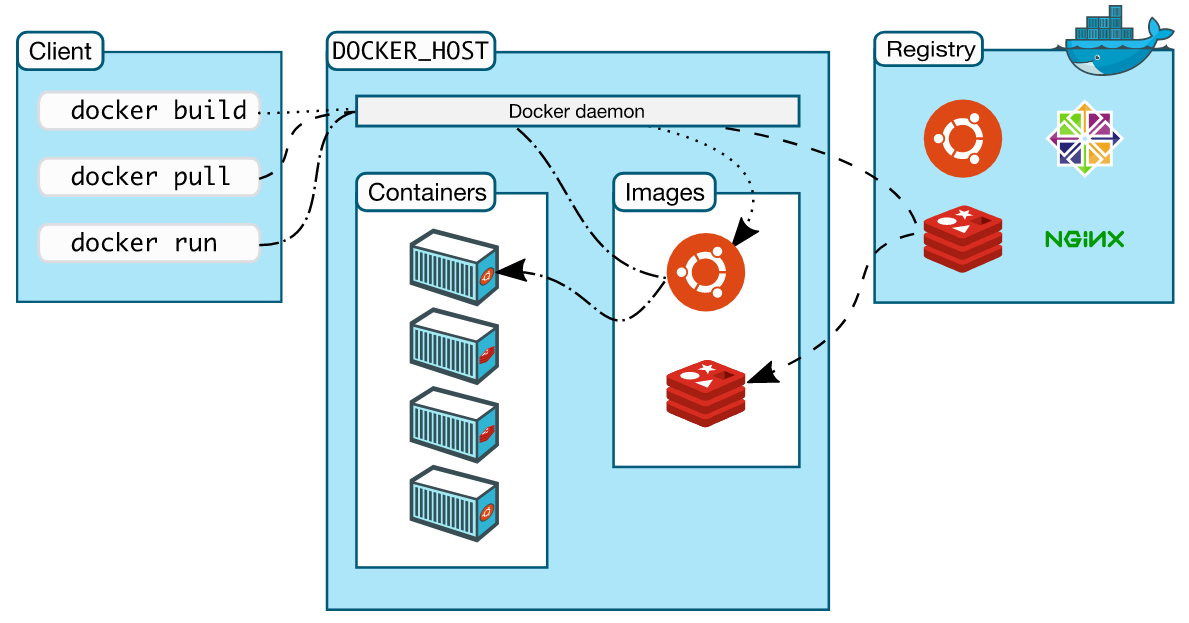






***Docker Components?***

Docker uses a ***client-server*** architecture. The Docker ***client*** talks to the Docker ***daemon***, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon



***The Docker daemon***

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

***The Docker client***

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

***Docker registries***

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

# Installation

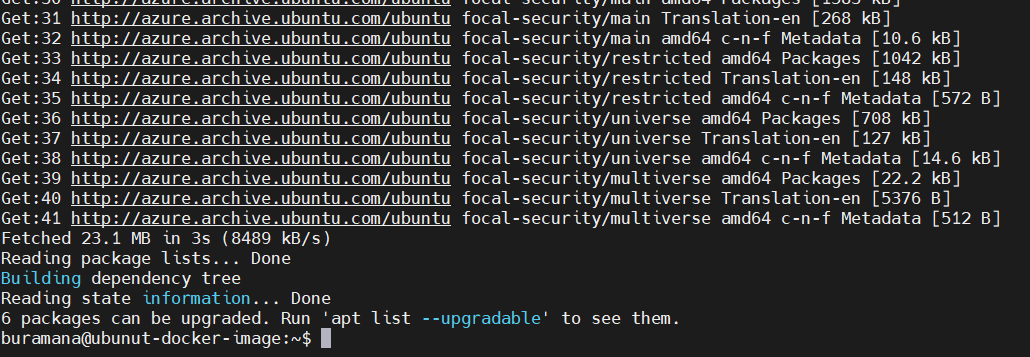
## On ubuntu:

If you want install latest version of docker run the below commands

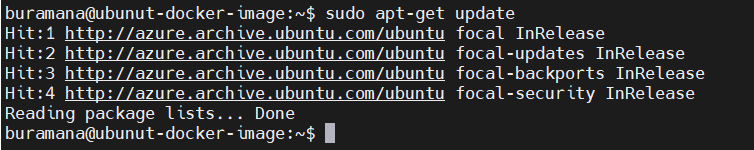
To avoid permission issues while executing commands run ***“sudo su”***

**apt update** - to update the repositories

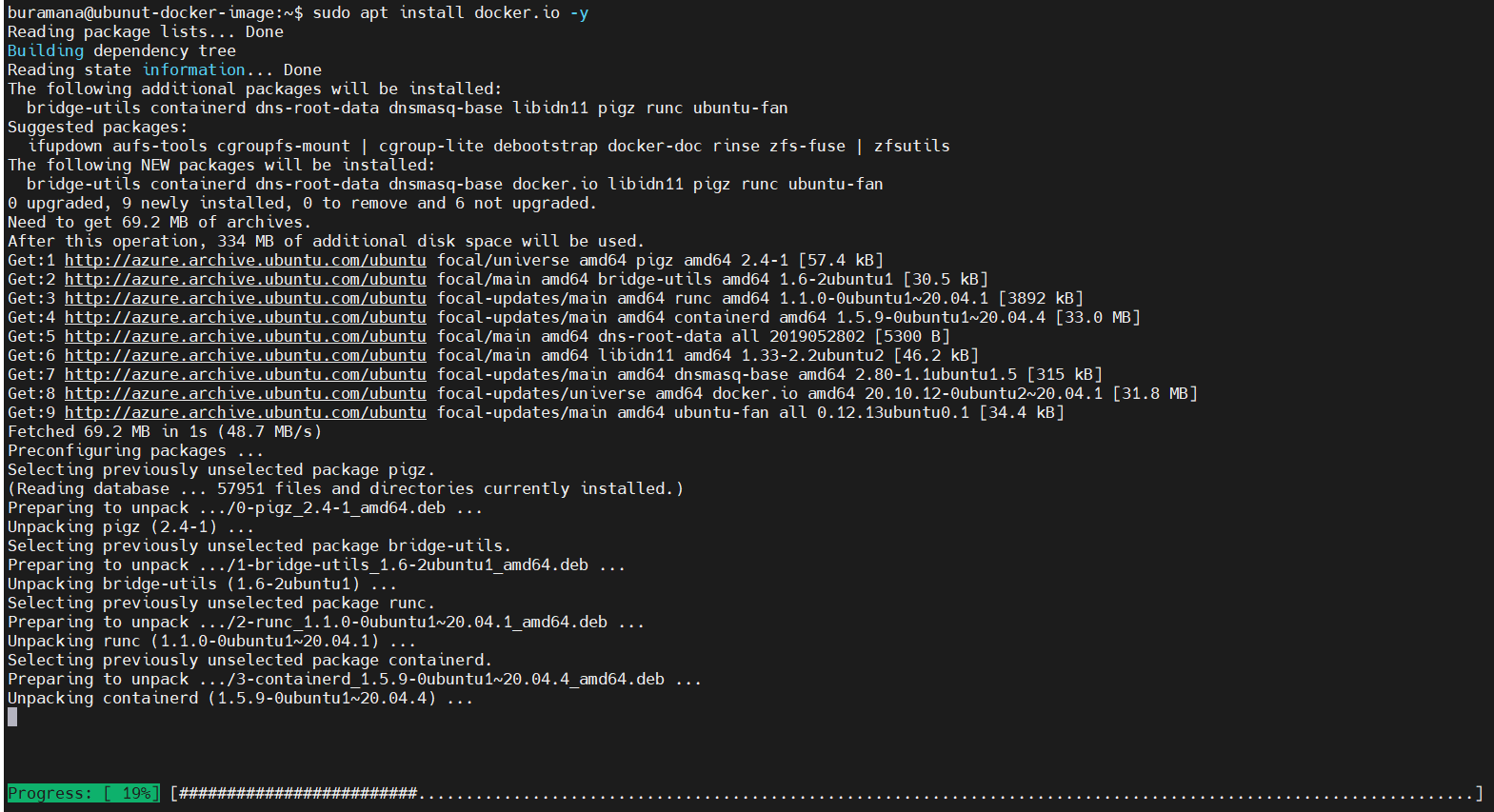


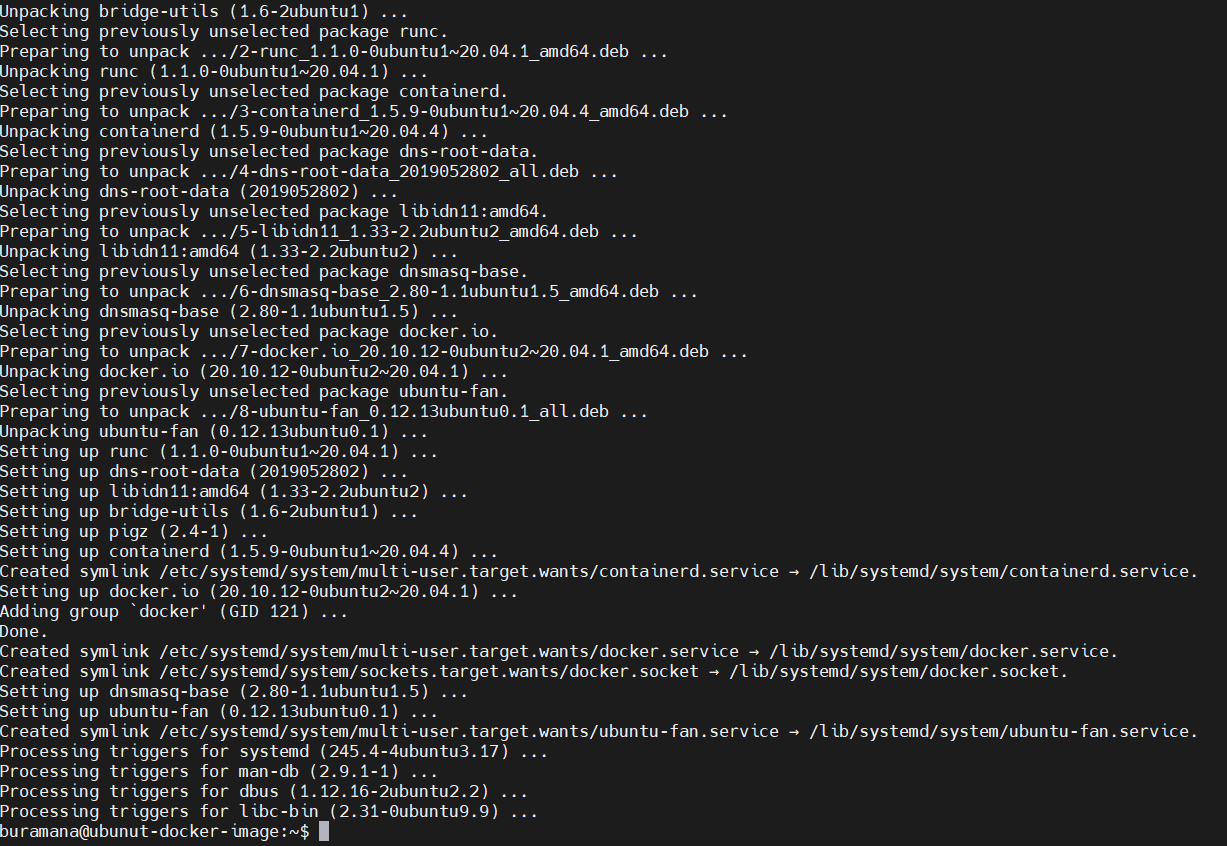


**apt-get update –** to update apt-get repo as well

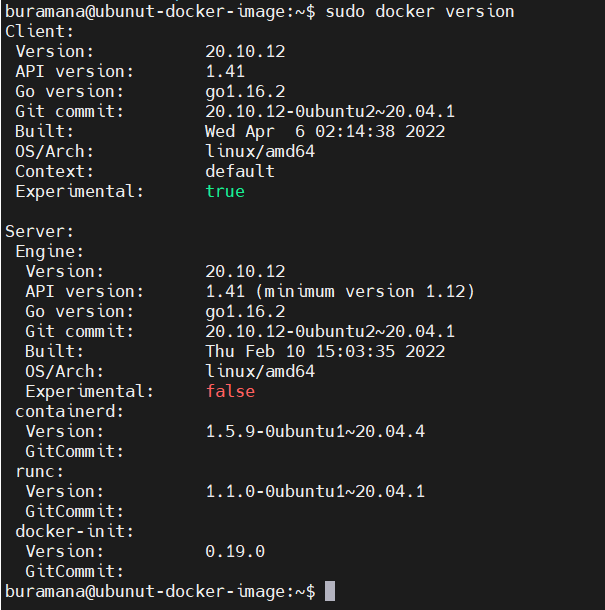


**apt install docker.io -y** – To install docker engine

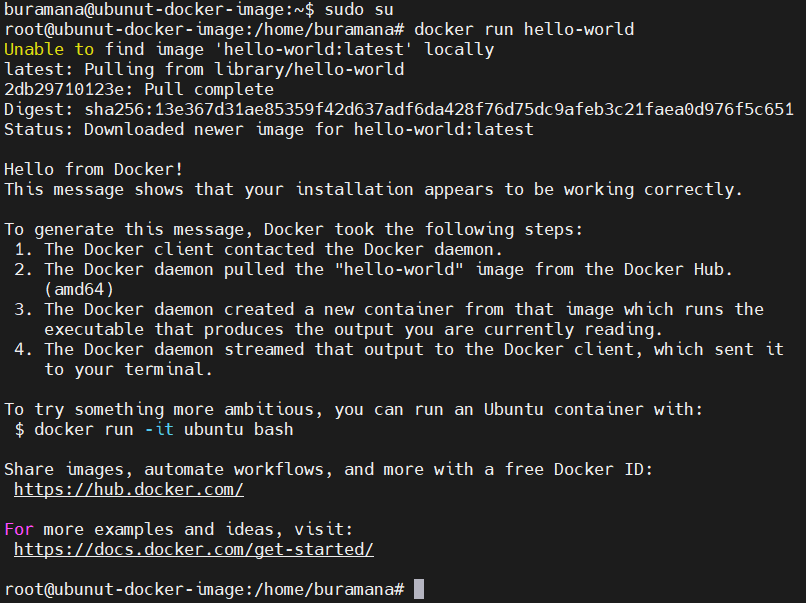




***docker version*** – to verify version of docker engine



***docker run hello-world*** – run simple hello-world container to make sure docker is working properly.



## On Windows

|  |  |
| --- | --- |
| **Steps** | **screenshot** |
| Login to the windows machine and download the Docker desktop here is the url [Docker Desktop](https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe?utm_source=docker&utm_medium=webreferral&utm_campaign=dd-smartbutton&utm_location=header)  Once download completed, file will be available in Downloads folder or default download location. |  |
| Double click on the installer to start the Docker installation. |  |
| It will asks to create icon on desktop. Click OK. |  |
| Wait till installation completed. Installation succeeded screen click on close. Installation is completed. |  |
| After installation to start it double click on Docker Desktop icon on your machine’s desktop. |  |
| It will open start page, check the accept the terms and then click on Accept button. Ignore any warning windows or messages. |  |
| Now it will restart the docker service |  |
| Once service got restarted. Getting started page will open. Either you start tutorial by click on start or click on skip tutorial to go main window. |  |
| Here I am click on Skip tutorial.  Here is the Home page. |  |
| Here we can see the running containers, if any. |  |
| In Images page , will display the local images and remote repository if logged to your private registry |  |
| In Volumes page,it will list if any volumes created. |  |
| Dev Environments will allow us to share our work with others |  |
| We can also use the power shell as command line to execute commands.  *docker version*  *docker --version* |  |
| Open services, to look at the docker service |  |

# Docker Commands

Graphical user interface, text, table

Description automatically generated

docker create <imagename>:<tag> - this command will just create container but not started.

docker create --name first-container httpd:latest



It will create the container with the name “first-container” & image name is httpd with latest tag and it will return the Container ID.

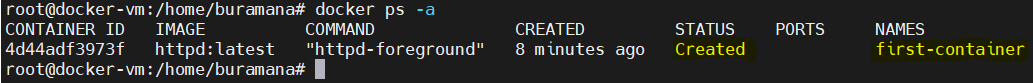
docker ps - To list the running containers

docker ps



docker ps -a – To list all containers

docker ps -a



docker start <container name or id> - to start the container and it will container id as output.

docker start first-container



Now, let check the running containers. Execute “docker ps” and observer the output.



docker stop <container name or Id> - to stop the running container

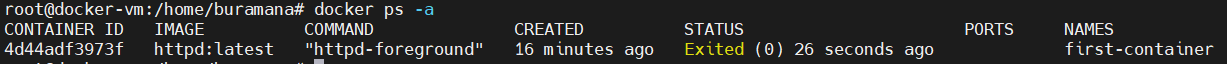
docker stop first-container



After stopping, the running container list will show empty



And all container list will have exited an running. But in this example we have only one that exited.

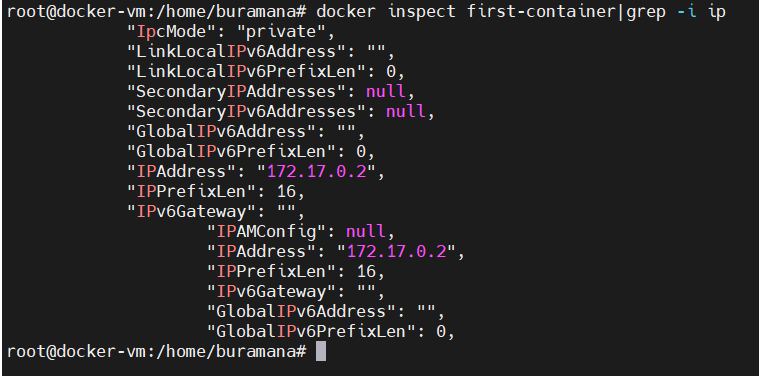


Till this we have seen creating, start and stop the containers.

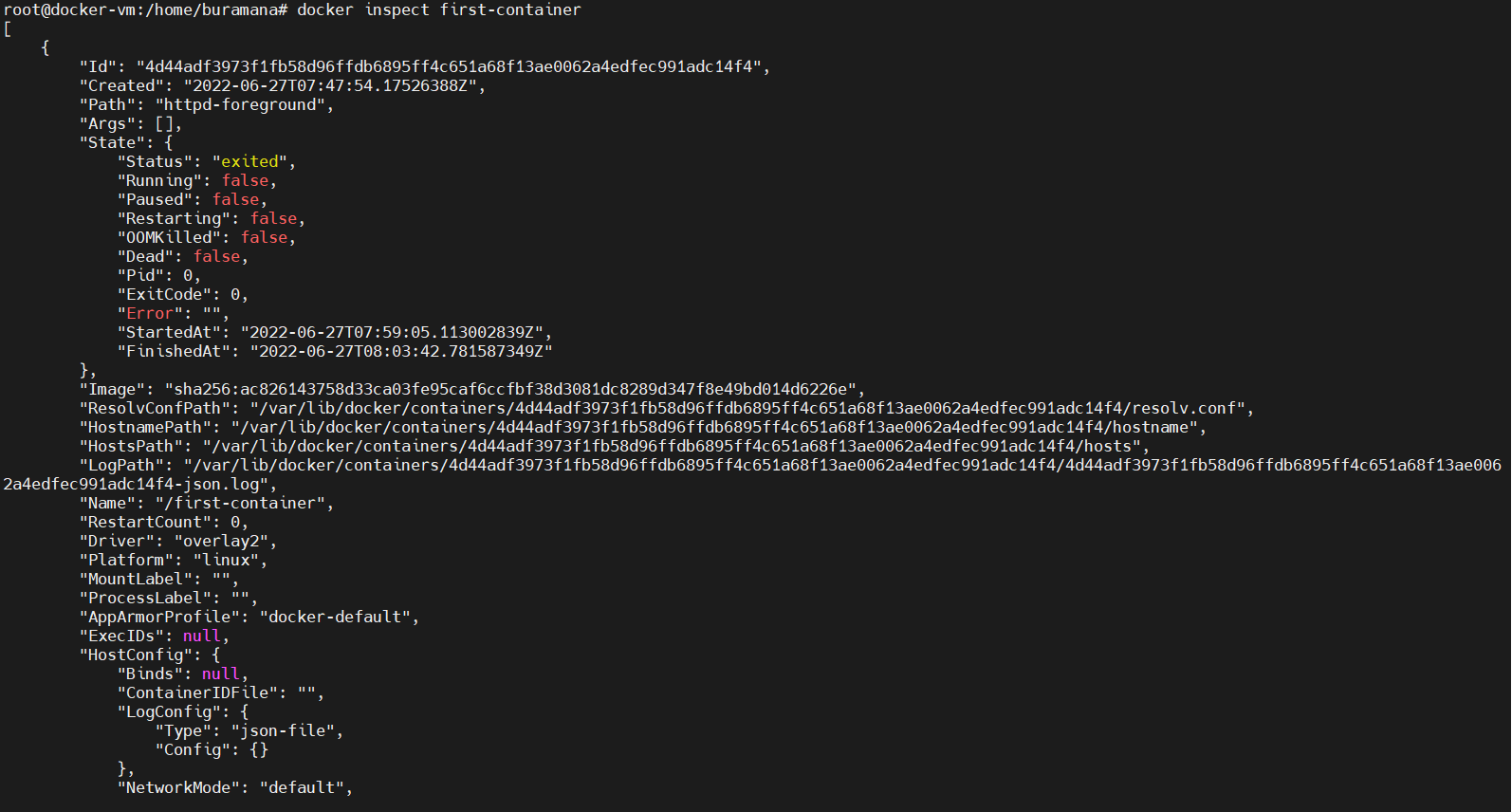
docker inspect <container-name or ID> - to get the container information

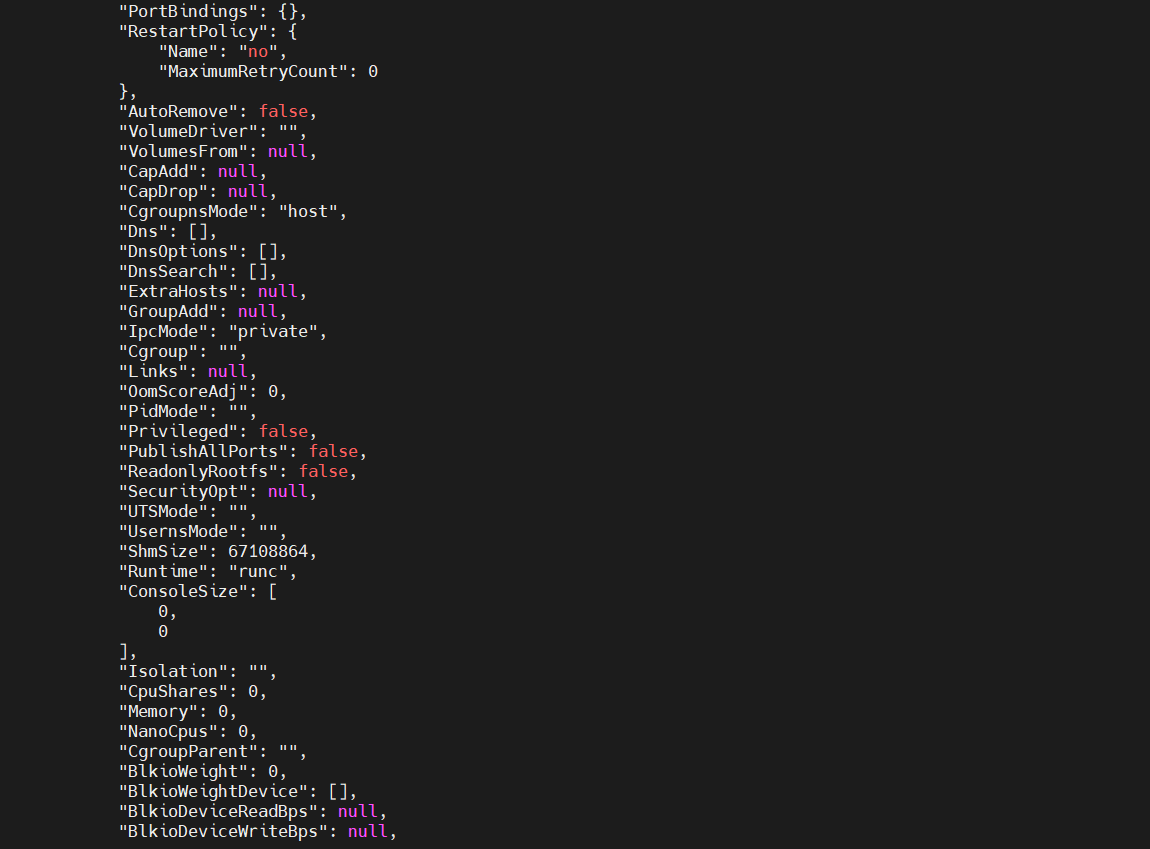
docker inspect first-container

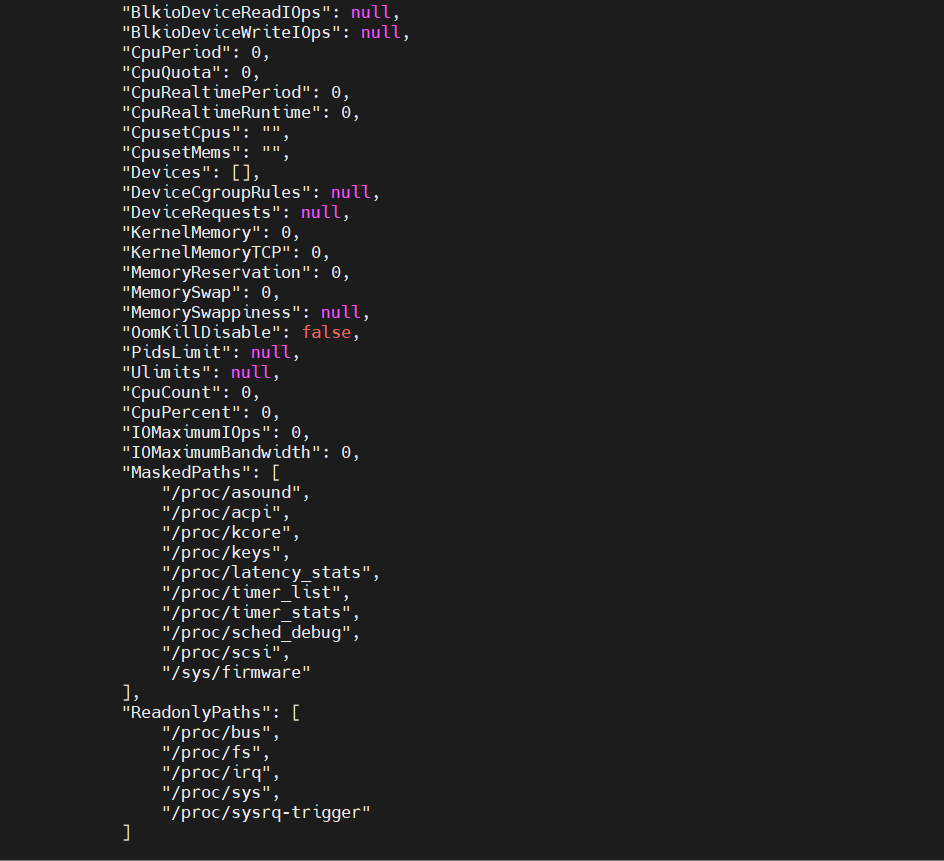
docker inspect first-container|grep -I ip

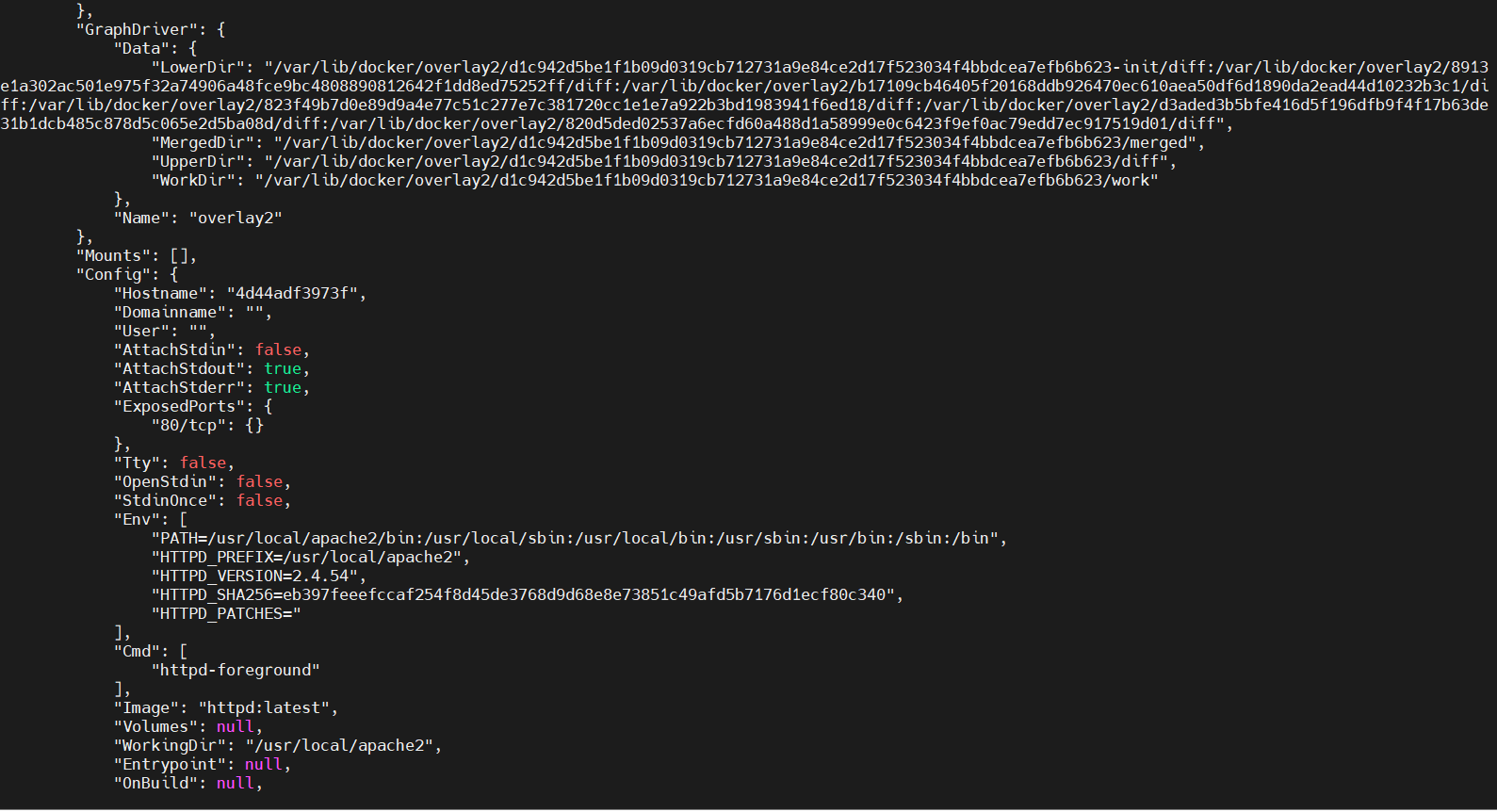


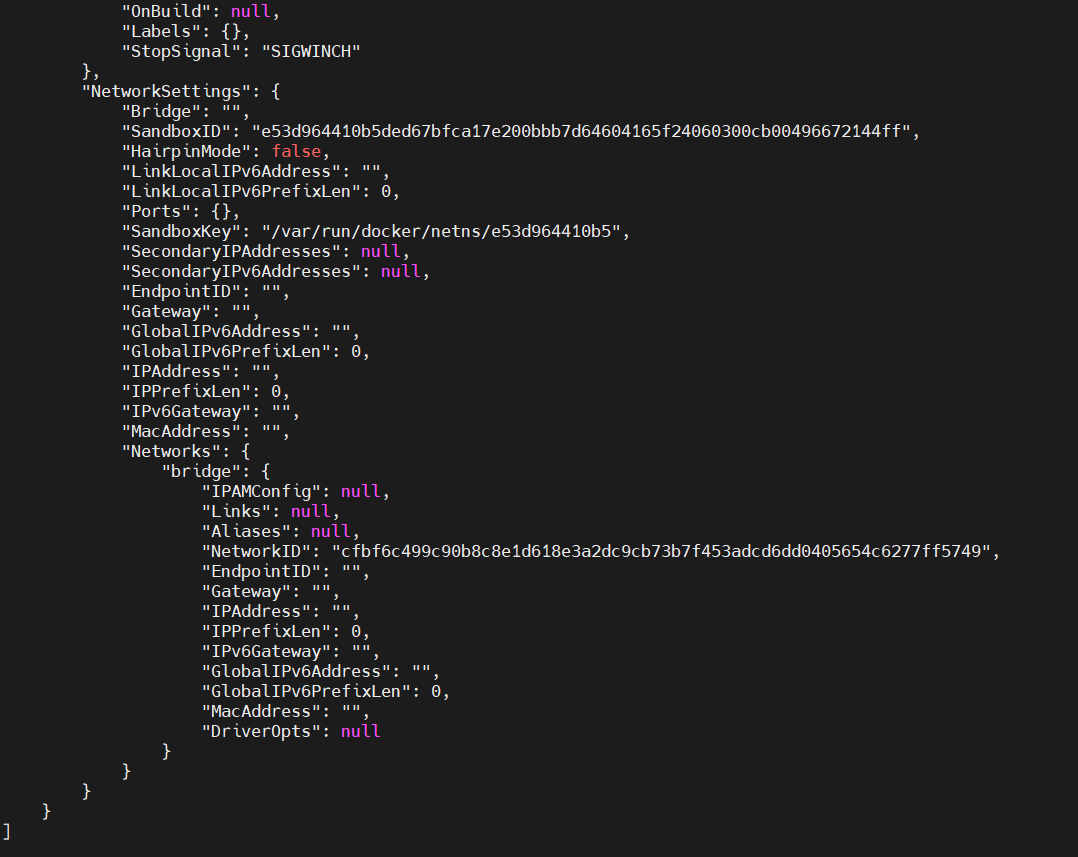
Here is the complete information about the containers





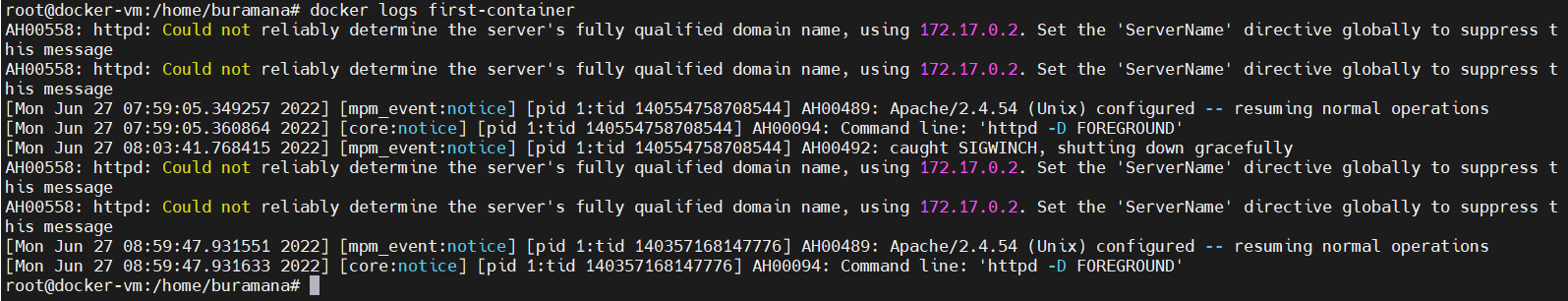






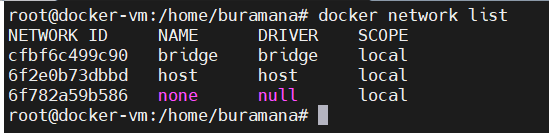
docker logs <container name or id> - to get the logs of container

docker logs first-container

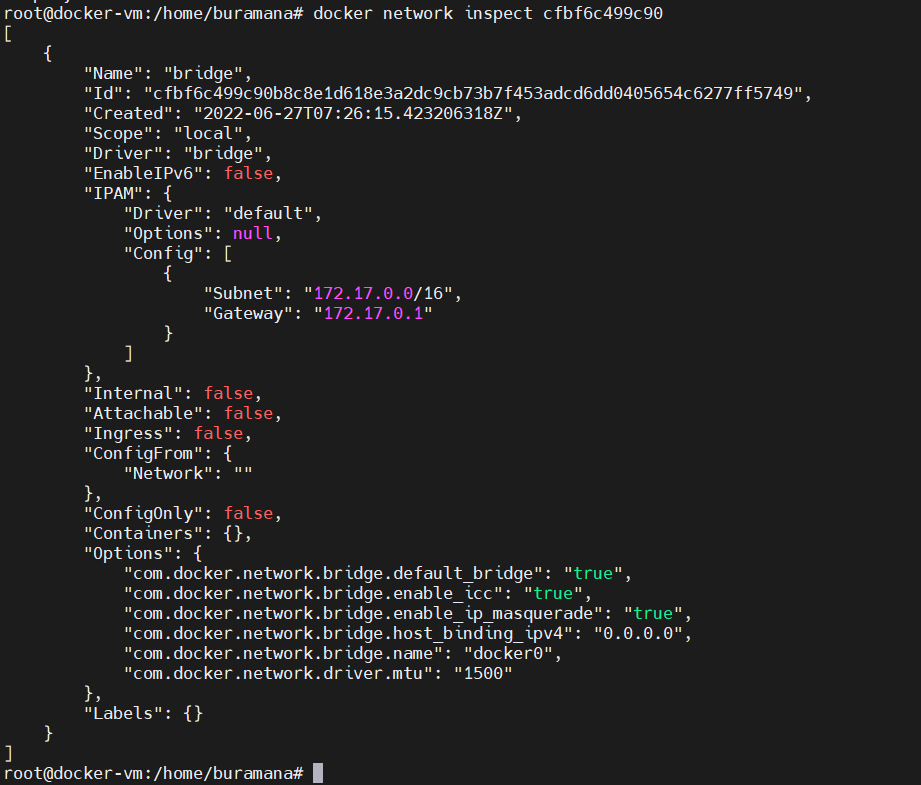


Networking:

docker network list – to list the network layers.



docker network inspect <network id> - To display the detailed information of network



docker network connect <network-id> <container id or name> - Connect the container to network.



docker network disconnect <network-id> <container id or name> - Disconnect the container to network.

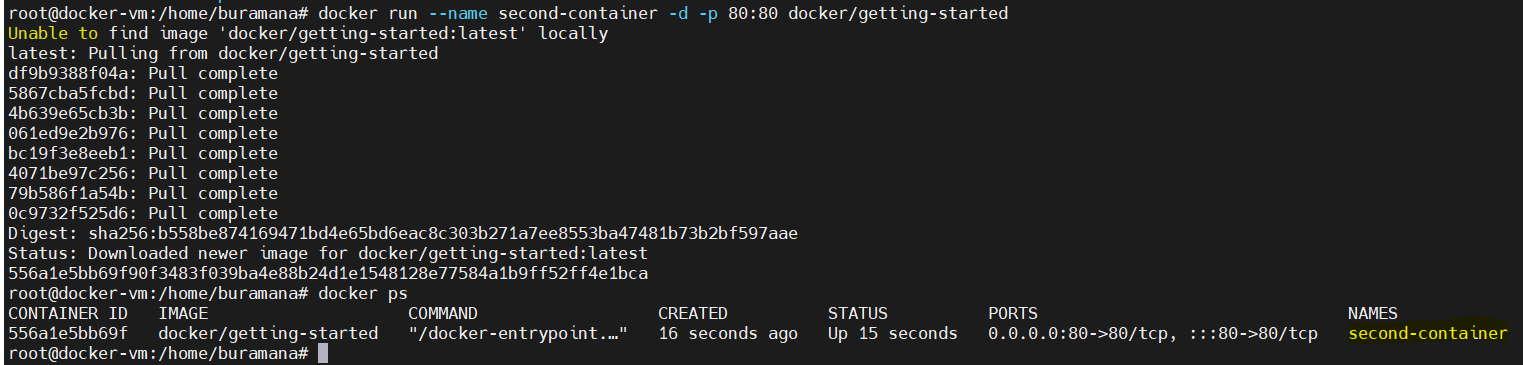


**Information:**

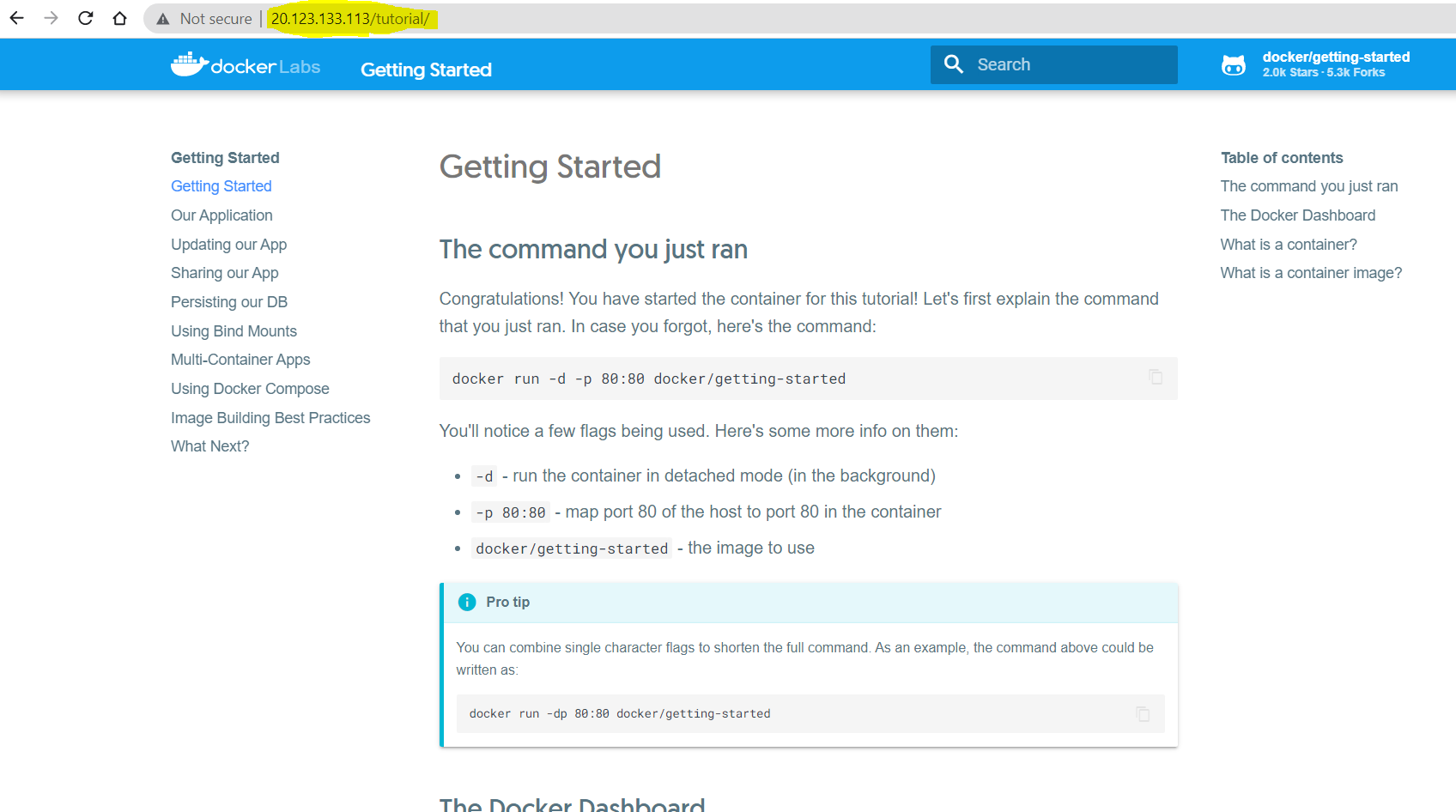
I created new container for this section

docker run –name second-container -d -p 80:80 docker/getting-started

It will create new container with name second-container and expose the port 80 to external through host on 80 and using image name is docker/getting-started.

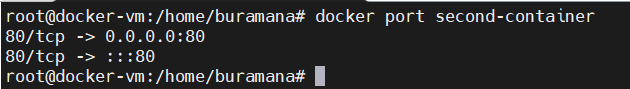


Now I am able to access the application running in container from external network using host ip(where container is running).

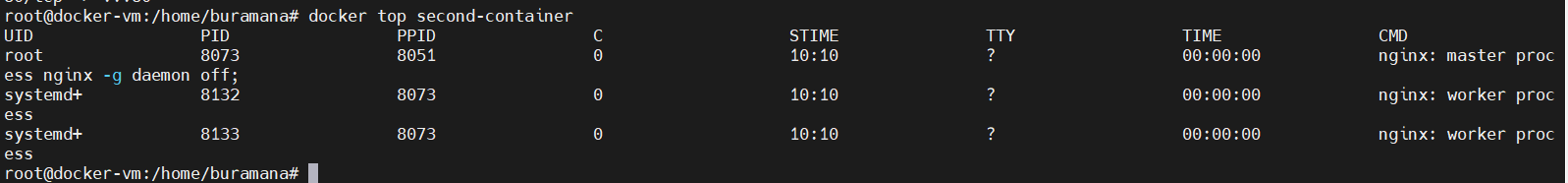


Docker port <container name/id> - to display port number on container and to which port on host itexposed.

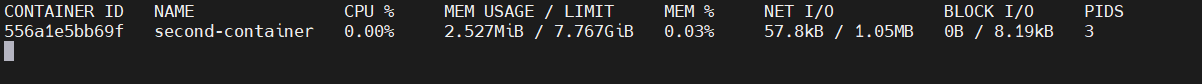
docker port second-container



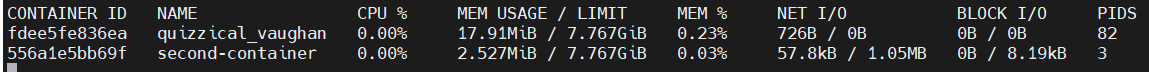
docker top second-container – to display running processes in container



docker stats second-container – to display compute utilisation of resource by particular container

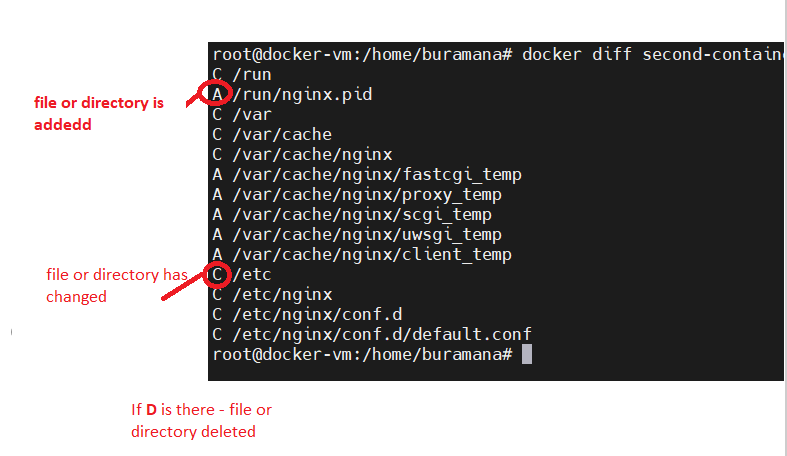


docker stats – to display utilization of resources by each container.



docker diff <container-name/Id> - It will display the changes in container file system.

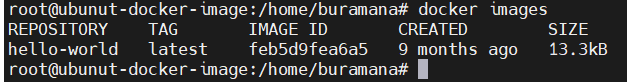
docker diff second-container



# Docker Images

 Docker image is **a file used to execute code in a Docker container**. Docker images act as a set of instructions to build a Docker container, like a template.

***docker images -*** To list images in local



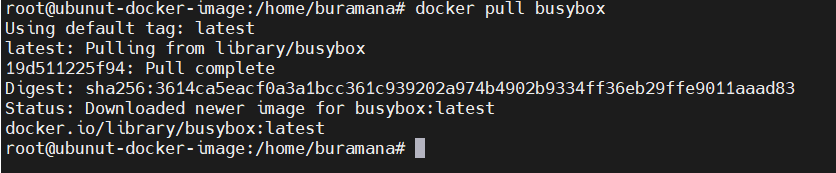
***docker pull <image name>:<tag>*** - to download the image from docker hub public repository.

Note: default tag name will be “latest”

***docker pull nginx*** – pulling nginx image

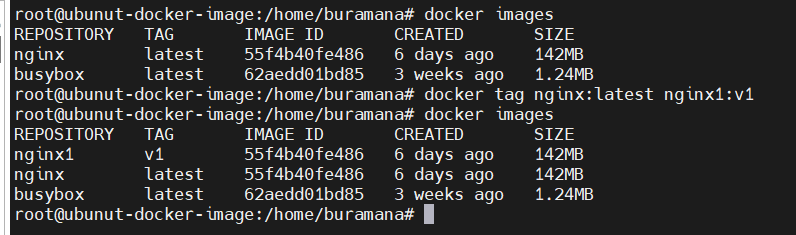


***docker pull busybox*** – pulling busbox image

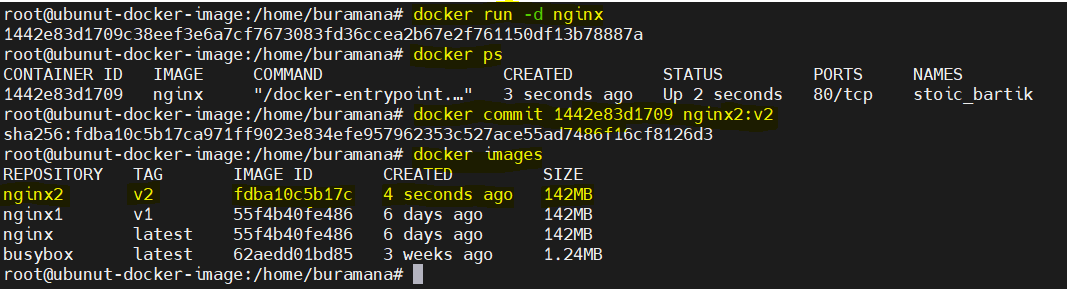


***docker tag <imagename>:<tag> <new-image-name>:<new-tag-name>*** - to retag the existing images

***docker tag nginx:latest nginx1:v1***



***docker commit <container-id> <image-name>:<tagname> -*** Creating image of running container



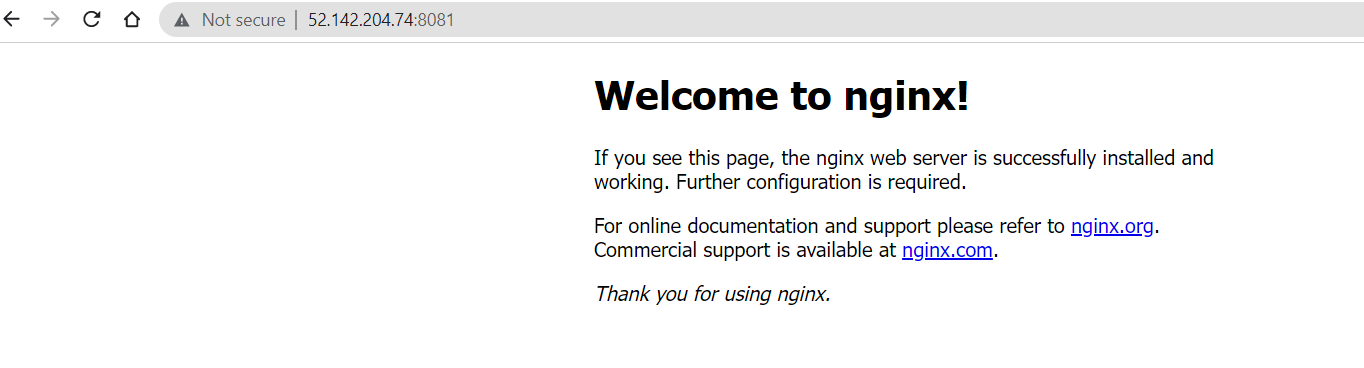
***Creating docker images using Dockerfile***

Dockerfile is simple text file that consists of instructions to create images.

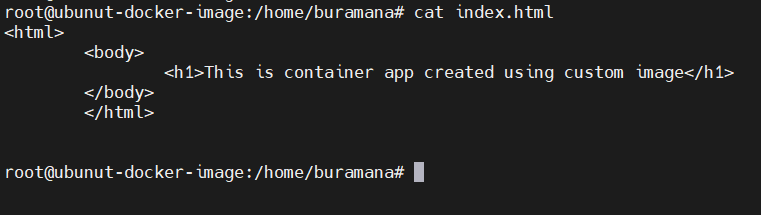
The file name should ***Dockerfile*** without any ***extension***

We will see how to create Dockerfile and creating image from that.

The default web page displayed by nginx container is looks like below.

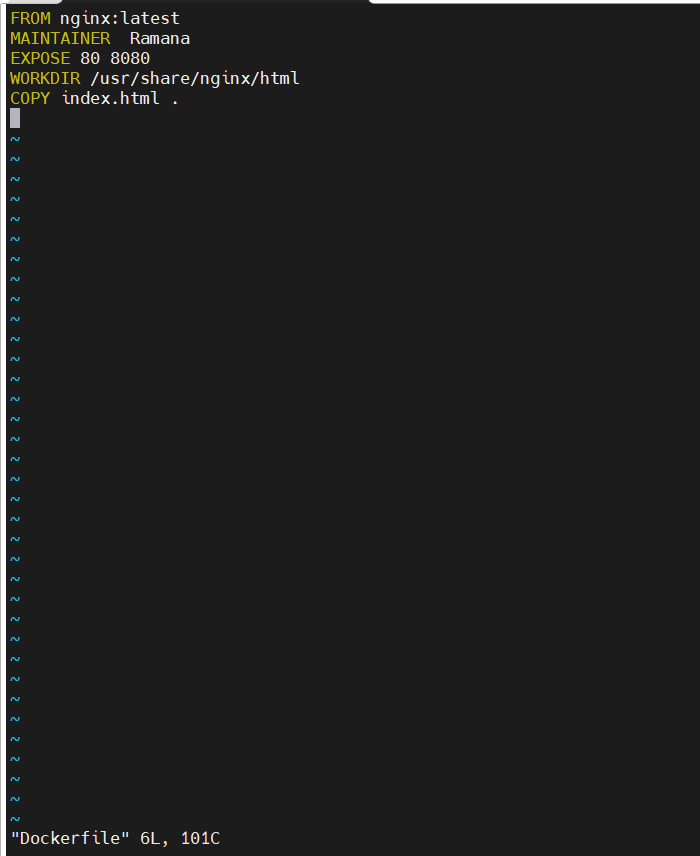


I want to create a new container image that will display the content like below



Dockefile:

Here is the content of dockerfile which I created.



***FROM nginx:latest*** – from instruction to mention the base image to create new image

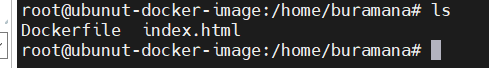
***MAINTAINER Ramana*** – Maintainer instruction is to give the name who creating this dockerfile

***EXPOSE 80 8080*** – Expose instruction to allow the container ports to external network to access the container application

***WORKDIR /usr/share/nginx/html*** – Workdir instruction make the given path as current directory

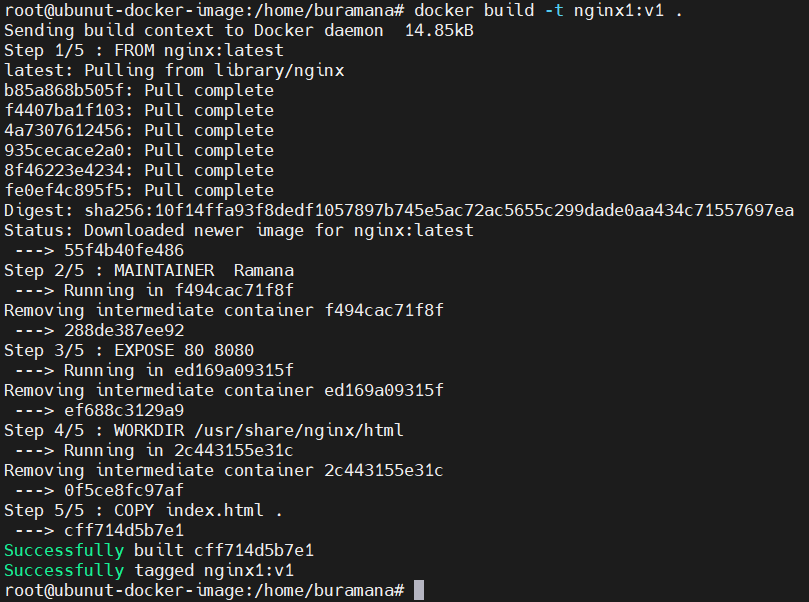
***COPY index.html . –*** Copy instruction to copy the files to docker image

Now I have two files

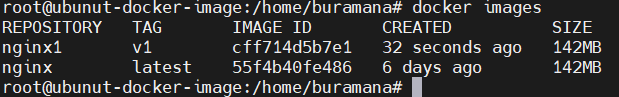


now to build the new image using Dockerfile

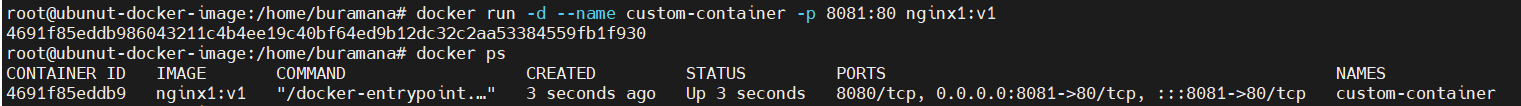
***docker build -t <imagename>:<tag-name> .(dot)***



Here is the new image just now created



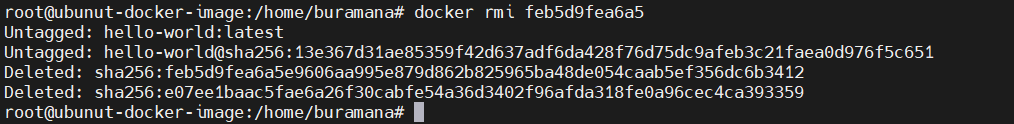
Now I created the container using the newly created image



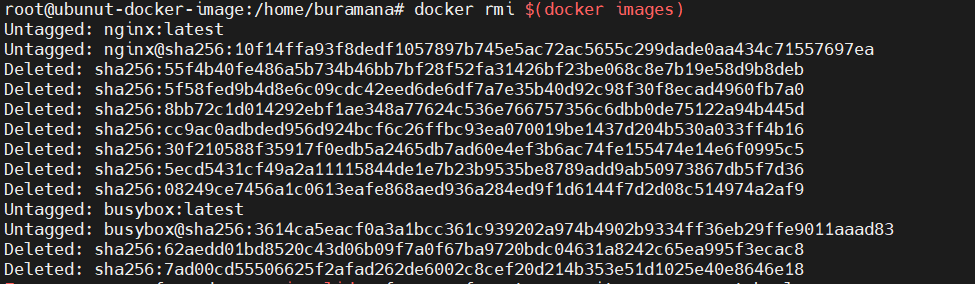
Now try to access the app with host ipaddress



***docker rmi <image name or ID > -*** to remove particular image.it deleted the hello-world image

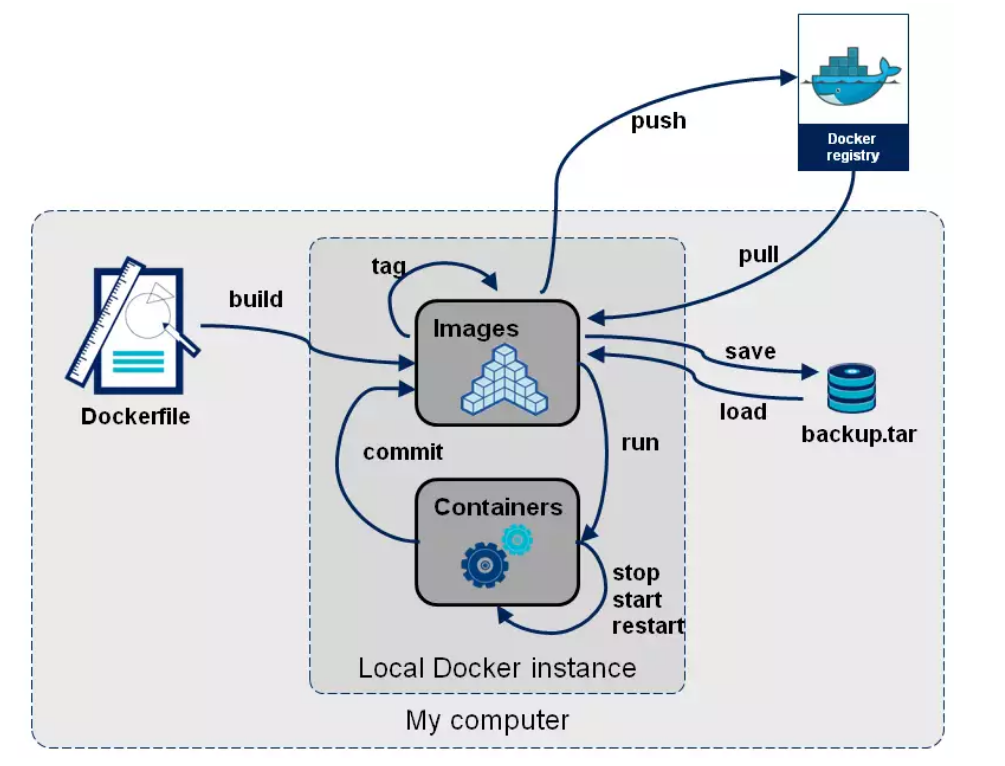


***docker rmi $(docker images) –*** to remove all images in local. It deleted nginx and busybox images



# Docker Registry

The Registry is a stateless, highly scalable server-side application that stores and lets you distribute Docker images



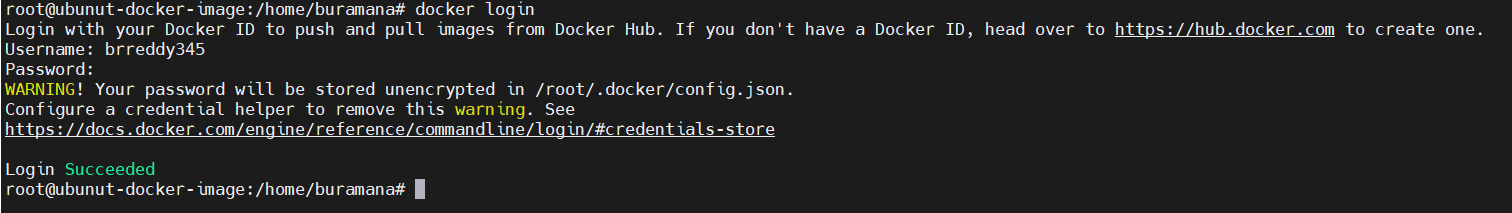
We will interact with registry while pull and push the images form docker client.

To create private registry in docker hub [docker hub signUp](https://hub.docker.com/).

By default when docker installed docker client connect to public registry, from public registry we only pull the images. <http://docker.io/library> is the default registry.

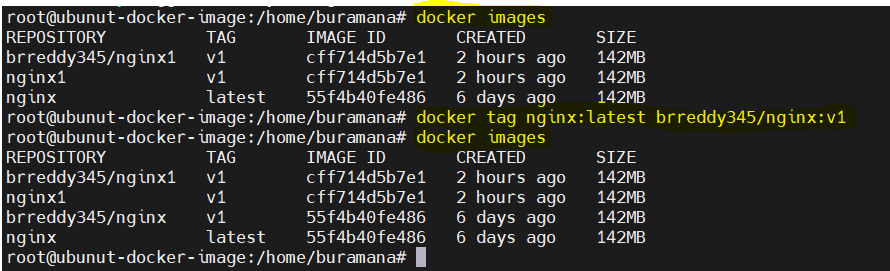
After signed up to docker hub, need to login from cli.

***docker login*** and hit enter, it will ask for ***username*** and ***password***



To push existing image to private registry we need to re-tag the image as below

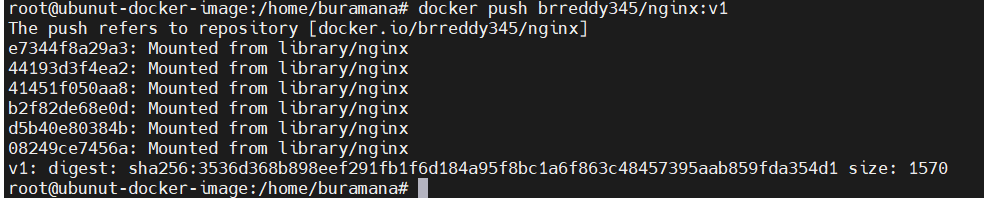
***docker tag <old-image-name>:<tag-name> <registry-name>/<new-image-name>:<tag-name>***



To push image to registry

***docker push <registry-name/<image-name>:<tag-name>***

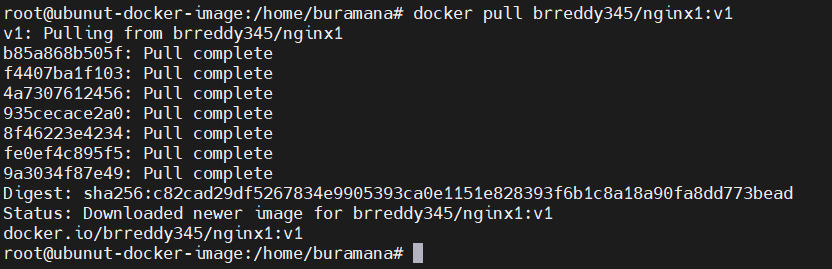
***docker push brreddy345/nginx:v1***



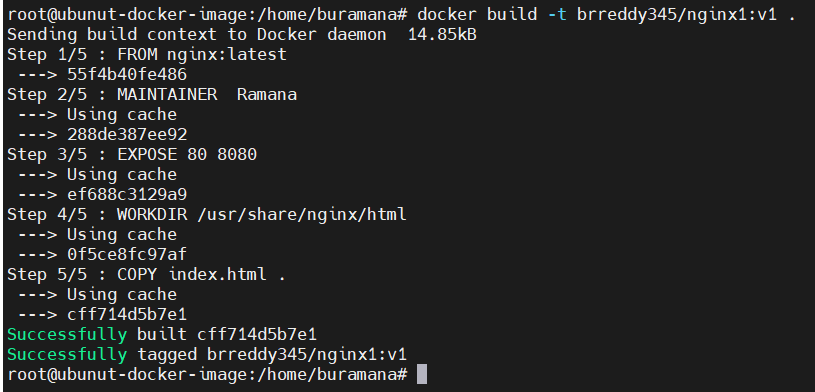
To pull the image from remote registry

***docker pull <registry-name/<image-name>:<tag-name>***

***docker pull brreddy345/nginx:v1***



Create new build image using Dockerfile as below.



# Dockerfile Instructions

* A Dockerfile is a text file which contains a series of commands or instructions.
* These instructions are executed in the order in which they are written.
* Execution of these instructions takes place on a base image.
* On building the Dockerfile, the successive actions form a new image from the base parent image.

***FROM:***

The FROMinstruction initializes a new build stage and sets the *base image* for subsequent instructions. A valid Dockerfile must start with a FROM instruction. The image can be any valid image – it is especially easy to start by pulling an image from the Public Repositories.



[***COPY***](https://dockerlabs.collabnix.com/beginners/dockerfile/lab4_dockerfile_copy.html)

The COPYinstruction copies files or directories from source and adds them to the filesystem of the container at destination

COPY [--chown=<user>:<group>] <src>... <dest>

COPY [--chown=<user>:<group>] ["<src>",... "<dest>"] (this form is required for paths containing whitespace)

[***ADD***](https://dockerlabs.collabnix.com/beginners/dockerfile/Lab-2-Create-an-image-with-ADD-instruction.html)

COPY and ADDare both Dockerfile instructions that serve similar purposes. They let you copy files from a specific location into a Docker image.

COPY takes in a src and destination. It only lets you copy in a local file or directory from your host (the machine building the Docker image) into the Docker image itself.

ADD lets you do that too, but it also supports 2 other sources. First, you can use a URL instead of a local file / directory. Secondly, you can extract a tar file from the source directly into the destination.

FROM alpine:3.5

RUN apk update

ADD http://www.vlsitechnology.org/pharosc\_8.4.tar.gz .

[***CMD***](https://dockerlabs.collabnix.com/beginners/dockerfile/lab4_cmd.html)

CMD instruction is used to set a command to be executed when running a container. There must be only one CMD in a Dockerfile. If more than one CMD is listed, only the last CMD takes effect.  
CMD instruction has two forms – Shell Form and Executable Form.

Executable form:

FROM alpine:3.6

RUN apk update

CMD [“ping” ,“google.com”]

Shell form:

FROM alpine:3.6

RUN apk update

CMD ping google.com

[***ENTRYPOINT***](https://dockerlabs.collabnix.com/beginners/dockerfile/Dockerfile-ENTRYPOINT.html)

The ENTRYPOINT instruction make your container run as an executable.  
ENTRYPOINT can be configured in two forms:

* **Exec Form**  
  ENTRYPOINT [“executable”, “param1”, “param2”]
* **Shell Form**  
  ENTRYPOINT command param1 param2

If an image has an ENTRYPOINT if you pass an argument it, while running container it wont override the existing entrypoint, it will append what you passed with the entrypoint.To override the existing ENTRYPOINT you should user **–entrypoint** flag when running container.

***Exec form:***

FROM alpine:3.5

LABEL maintainer="Collabnix"

ENTRYPOINT ["/bin/echo", "Hi, your ENTRYPOINT instruction in Exec Form !"]

***Shell form:***

FROM alpine:3.5

LABEL maintainer="Collabnix"

ENTRYPOINT echo "Hi, your ENTRYPOINT instruction in Shell Form !"

[***WORKDIR***](https://dockerlabs.collabnix.com/beginners/dockerfile/WORKDIR_instruction.html)

The WORKDIR directive in Dockerfile defines the working directory for the rest of the instructions in the Dockerfile. The WORKDIR instruction wont create a new layer in the image but will add metadata to the image config. If the WORKDIR doesn’t exist, it will be created even if it’s not used in any subsequent Dockerfile instruction. you can have multiple WORKDIR in same Dockerfile. If a relative path is provided, it will be relative to the previous WORKDIR instruction.

If no WORKDIR is specified in the Dockerfile then the default path is /. The WORKDIR instruction can resolve environment variables previously set in Dockerfile using ENV.

***Relative path:***

FROM alpine:3.9.3

LABEL maintainer="Collabnix"

WORKDIR /opt

***Environment variable:***

FROM alpine:3.9.3

LABEL maintainer="Collabnix"

ENV DIRPATH /myfolder

WORKDIR $DIRPATH

[***RUN***](https://dockerlabs.collabnix.com/beginners/dockerfile/Lab%237_RUN_instruction.html)

The RUN instruction execute command on top of the below layer and create a new layer.

FROM alpine:3.9.3

LABEL maintainer="Collabnix"

RUN apk add --update

RUN apk add curl

RUN rm -rf /var/cache/apk/

[***ARG***](https://dockerlabs.collabnix.com/beginners/dockerfile/arg.html)

ARG instruction is also used to set environment variables with key and value, but this variables will set only during the image build not on the container

The ARG directive in Dockerfile defines the parameter name and defines its default value. This default value can be overridden by the --build-arg <parameter name>=<value> in the build command docker build.

FROM alpine:3.9.3

LABEL maintainer="Collabnix"

#Setting a default value to Argument WELCOME\_USER

ARG WELCOME\_USER=Collabnix

RUN echo "Welcome $WELCOME\_USER, to Docker World!" > message.txt

CMD cat message.txt

Passing values to arg while building the images

$ docker image build -t arg:v2 --build-arg WELCOME\_USER=Savio .

[***ENV***](https://dockerlabs.collabnix.com/beginners/dockerfile/Lab_%239_ENV_instruction.html)

ENV instruction is used to set environment variables with key and value. Lets say, we want to set variables APP\_DIR and app\_version with the values /data and 2.0 respectively. These variables will be set during the image build also available after the container launched.

FROM alpine:3.9.3

LABEL maintainer="Collabnix"

ENV WELCOME\_MESSAGE="Welcome to Docker World"

CMD ["sh", "-c", "echo $WELCOME\_MESSAGE"]

*Override environment values while running container:*

$ docker container run --env WELCOME\_MESSAGE="Welcome to Docker Workshop" env:v1

Welcome to Docker Workshop

[***VOLUME***](https://dockerlabs.collabnix.com/beginners/dockerfile/Lab%2310_VOLUME_instruction.html)

VOLUME instruction is used to create or mount a volume to the docker container from the docker host filesystem.

FROM nginx:alpine

LABEL maintainer="Collabnix"

VOLUME /myvol

CMD [ "nginx","-g","daemon off;" ]

[***EXPOSE***](https://dockerlabs.collabnix.com/beginners/dockerfile/Lab%2311_EXPOSE_instruction.html)

EXPOSE instruction is used to inform about the network ports that the container listens on runtime. Docker uses this information to interconnect containers using links and to set up port redirection on docker host system.

FROM nginx:alpine

LABEL maintainer="Collabnix"

EXPOSE 80/tcp

EXPOSE 80/udp

CMD [ "nginx","-g","daemon off;" ]

[***LABEL***](https://dockerlabs.collabnix.com/beginners/dockerfile/Label_instruction.html)

LABEL instruction is used to specify metadata information to an image. A LABEL is a key-value  pair.

LABEL "com.example.vendor"="ACME Incorporated"

LABEL com.example.label-with-value="foo"

LABEL version="1.0"

LABEL description="This text illustrates

[***ONBUILD***](https://dockerlabs.collabnix.com/beginners/dockerfile/onbuild.html)

ONBUILD instruction is used to specify a command that runs when the image in the Dockerfile is used as a base image for another image.

FROM busybox

ONBUILD RUN echo "You won't see me until later"

[***USER***](https://dockerlabs.collabnix.com/beginners/dockerfile/user.html)

USER instruction is used to set the username, group name, UID and GID for running subsequent commands. Else root user will be used.

**RUN** groupadd -r redis **&&** useradd -r -g redis redis

**USER** redis

**RUN [** "redis-server" **]**

Complete Dockerfile:

Have to create two files *hello.py* and *a.py* in docker host

**#Dockerfile to create docker image which execute python hello app**

FROM ubuntu

MAINTAINER Ramana "b.rreddy703@outlook.com"

RUN apt-get update

RUN apt-get install python

ADD hello.py /home/hello.py

ADD a.py /home/a.py

CMD ["/home/hello.py"]

ENTRYPOINT ["python"]

Build command

**#To create docker images**

**docker build -t py-image:v1 .**

**#To list images**

**docker images**

# Docker Volume

Volumes are basically directories of host system managed by Docker

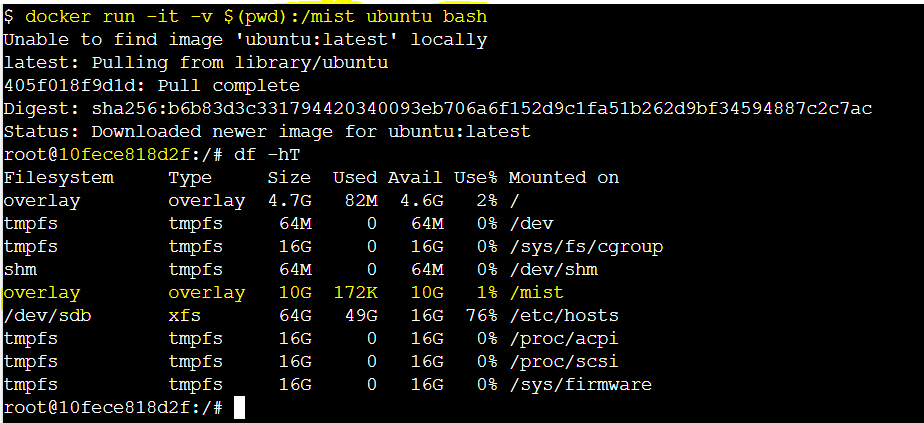
There are three main use cases for Docker data volumes:

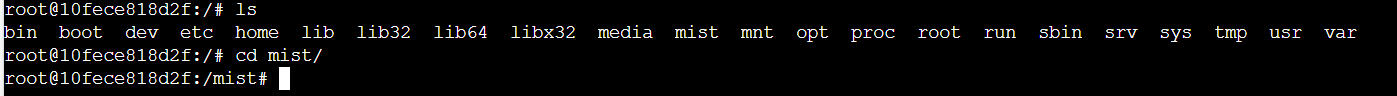
1. To keep data around when a container is removed
2. To share data between the host filesystem and the Docker container
3. To share data with other Docker containers

Docker run has a -v flag which allows to set data volumes that are mounted inside of our container

1. Run the docker container by mapping the current directory as /mist in the container. We are adding a local drive as a volume to the contianer

***docker run -it -v $(pwd):/mist ubuntu bash***





Now once container is started we can check the file system using “df -hT” which will show the /mist as a drive attached.

Create a test file in the /mist location

root@dea9f1c0a043:/# cd mist/

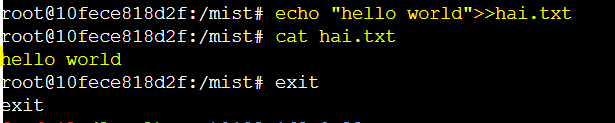
root@dea9f1c0a043:/mist# echo "hello World" >> hai.txt

root@dea9f1c0a043:/mist# cat hai.txt

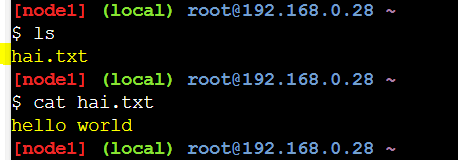
hello World

root@dea9f1c0a043:/mist# exit

exit



Once we exit ,we can see the created file in the container in our local repository.



# Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.

***Features:***

The features of Compose that make it effective are:

* Multiple isolated environments on a single host
* Preserve volume data when containers are created
* Only recreate containers that have changed
* Variables and moving a composition between environments

***Docker compose installation:***

In ubuntu:

To update the registry

***sudo apt update -y***

To install docker compose

***sudo apt install docker-compose -y***

To run multiple containers and link up them in single host we use below commands.

Here I am using redis and postgres images to create container and linkup them.

Creating two containers individually

***docker run -d –name=redis redis***

***docker run -d –name=db -e POSTGRES\_PASSWORD=password -e POSTGRES\_HOST\_AUTH\_METHOD=trust postgres***

creating another two containers and link previous containers to new ones

***docker run -d –name=vote -p 5000:80 –link redis:redis eesprit/voting-app-vote***

***docker run -d –name=result -p 5001:80 –link db:db eesprit/voting-app-result***

This is hectic process to create and link up container in single host using docker run commands. By using docker-compose file we can overcome the difficulties.

Docker-compose file is the Yaml file in which we can define the containerized application.

Here is the example docker-compose file and attaching in this document as well 

version and services are the mandatory sections in docker-compose.yml file.

***version: "3.1"***

***services:***

***db:***

***condition: service\_healthy***

***volumes:***

***- ./result:/app***

***ports:***

***- "5001:80"***

***- "5858:5858"***

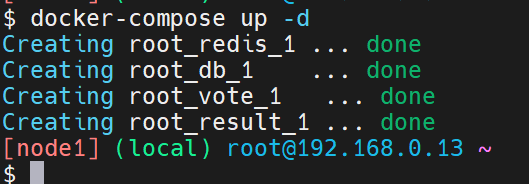
***networks:***

***- front-tier***

***- back-tier***

After creating the docker-compose.yml file, need to create containers b running below command.

docker-compose up -d

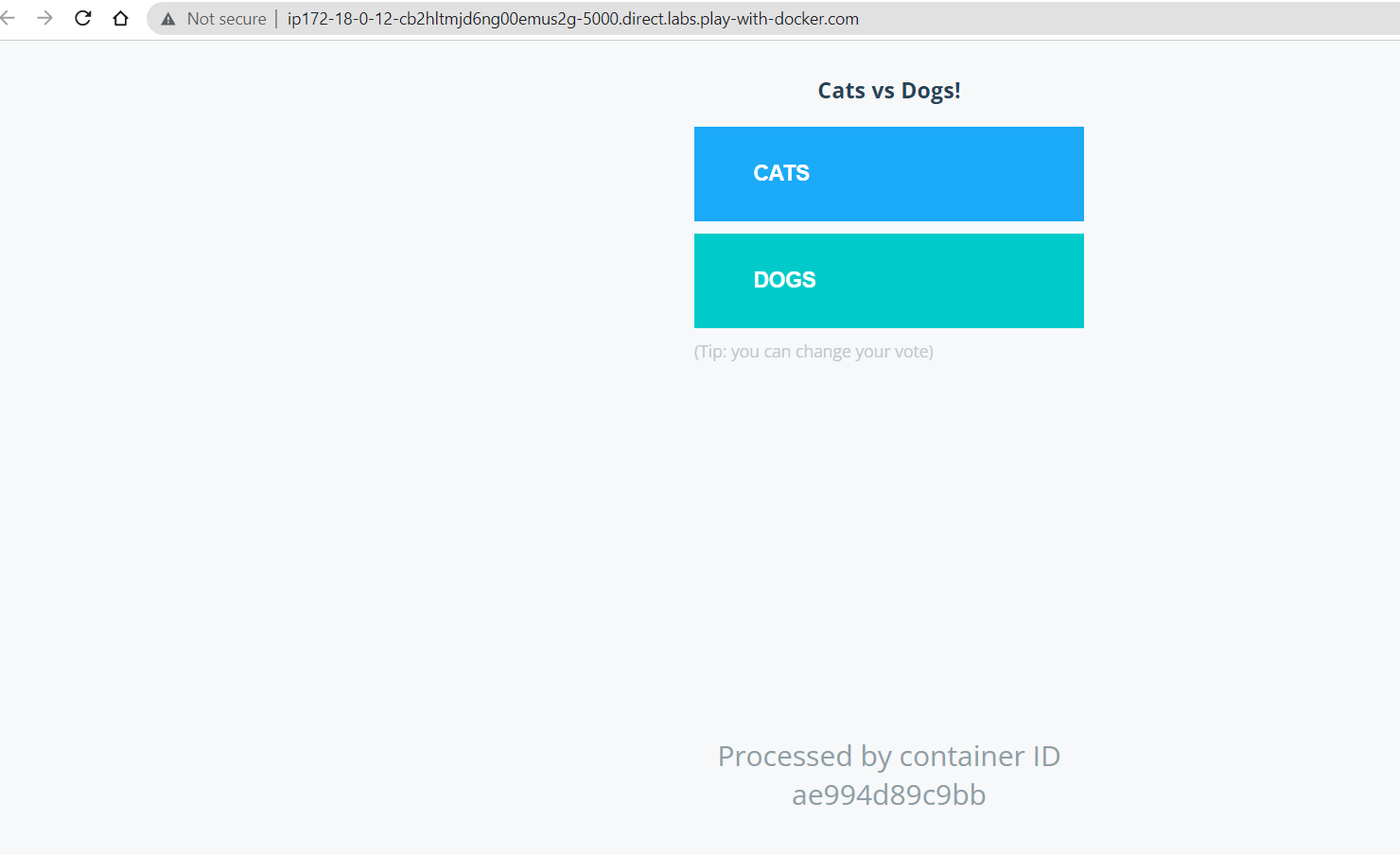


**docker ps** – to list the containers just now created

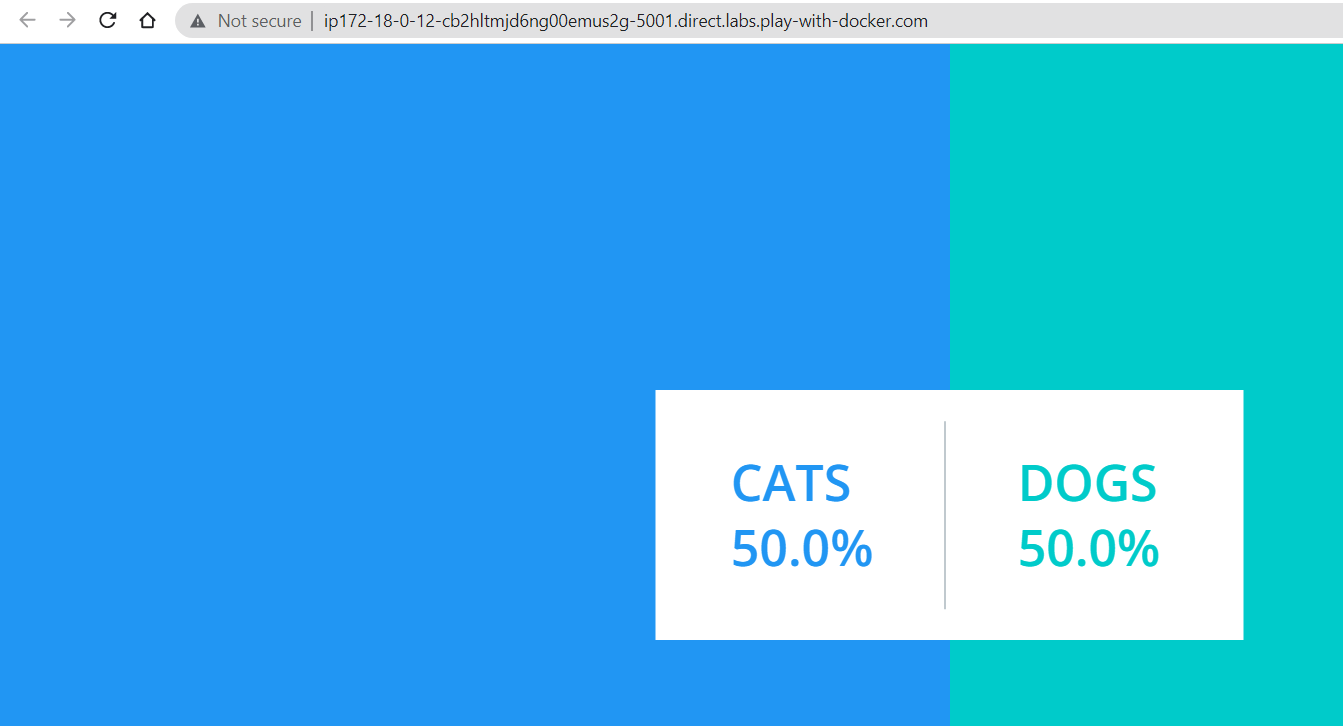


Access the application with docker host ip

Voting app:



Result app:

s