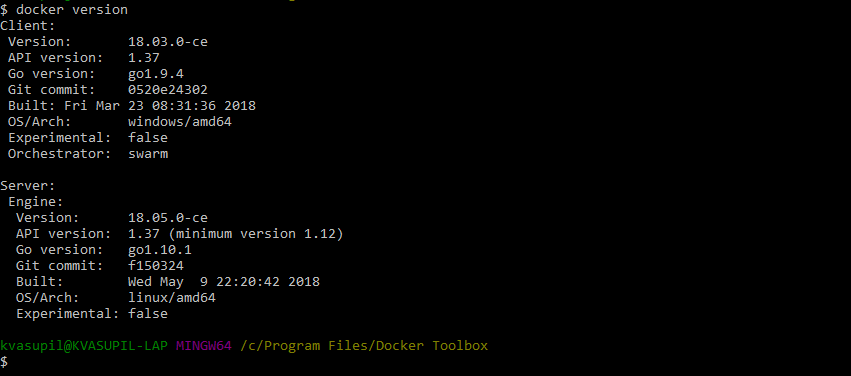
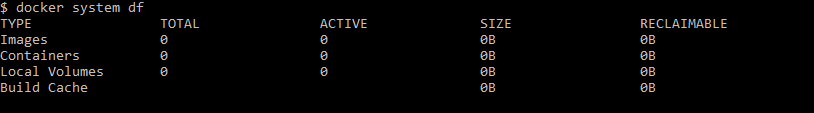
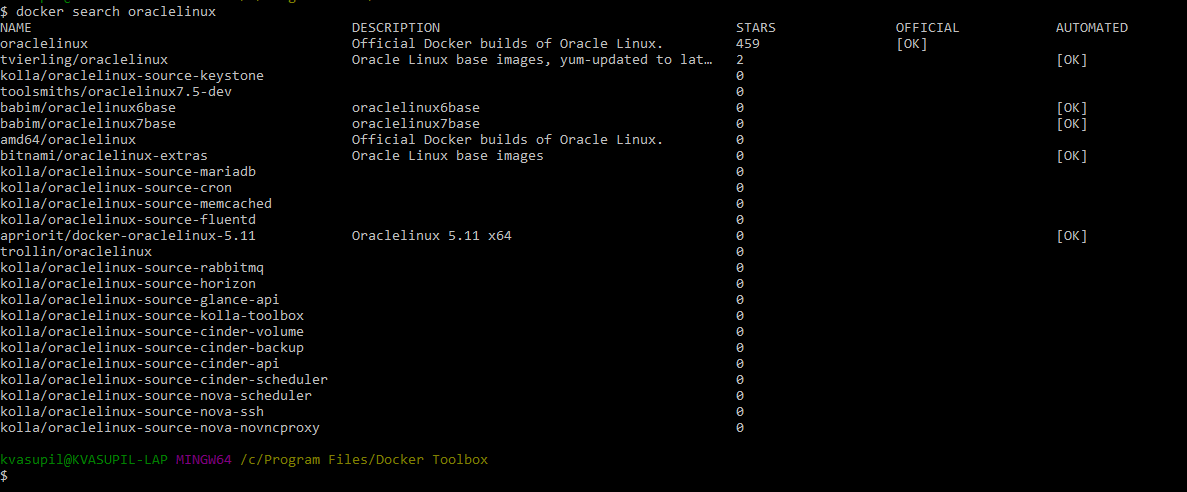
**docker info:**



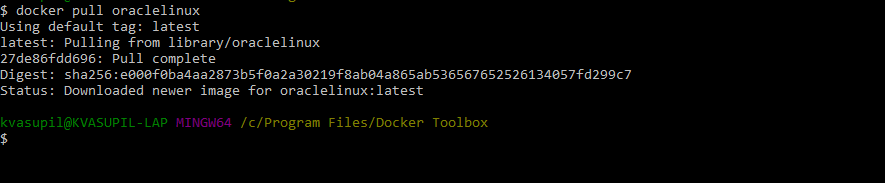
**docker system df :** Summary Report on how much space each container has occupied.



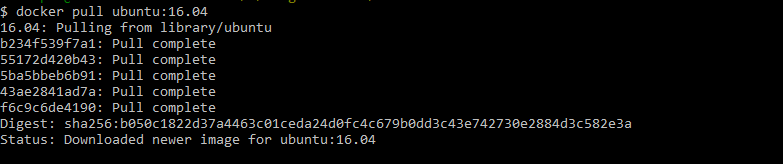
**docker search oraclelinux: To Search Oracle Linux images.**



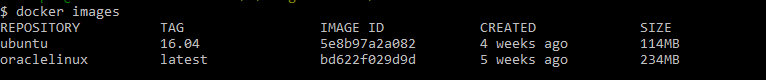
**docker pull oraclelinux:** To pull an Oracle Linux Image



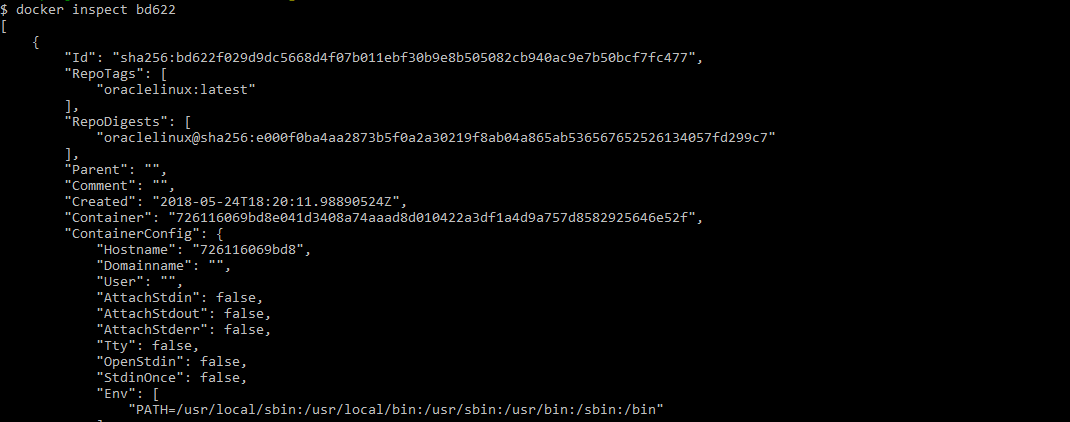
**docker pull ubuntu:16.04:** To pull Ubuntu OS version 16.04



**docker images: To list all the docker images on the Docker Host**



**docker inspect:** To inspect a container



JSON basically hosts 3 types of properties

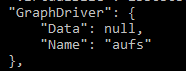
1. Name-value Pairs



2. An array



3. A collection of items.



docker inspect –f bd622



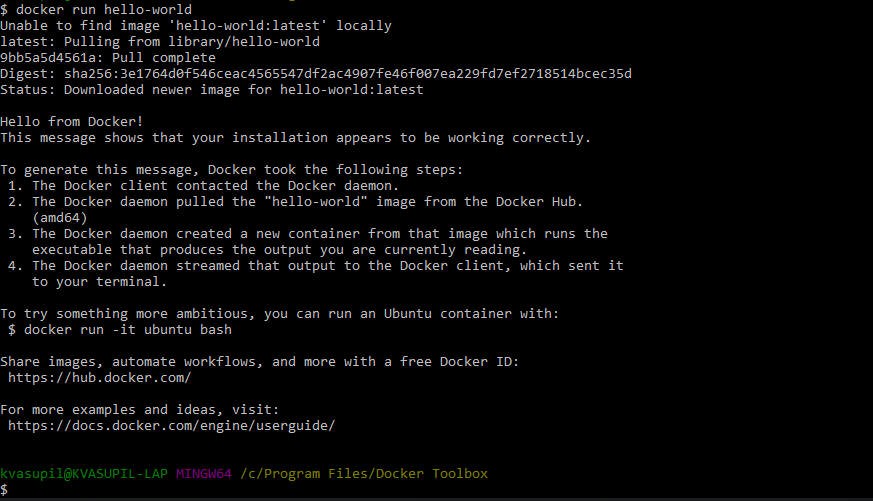




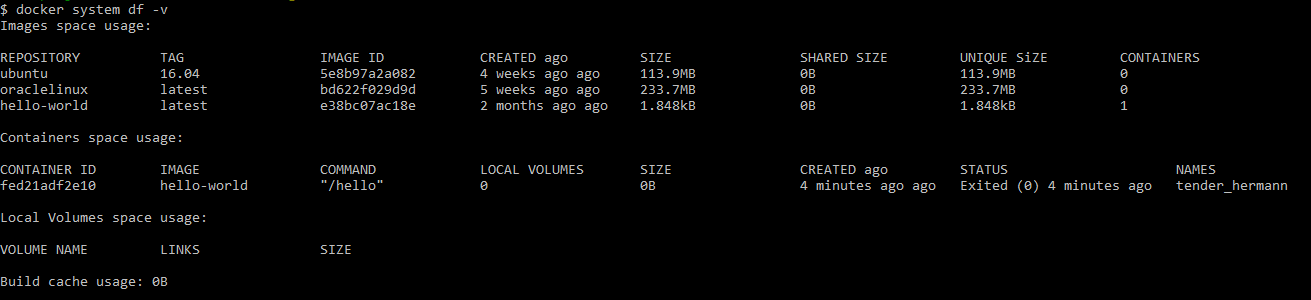


**docker run:** To run a docker container.

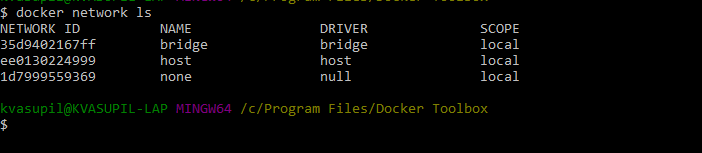
Docker will search for hello-world image locally, if not found, will automatically pull it from docker hub.

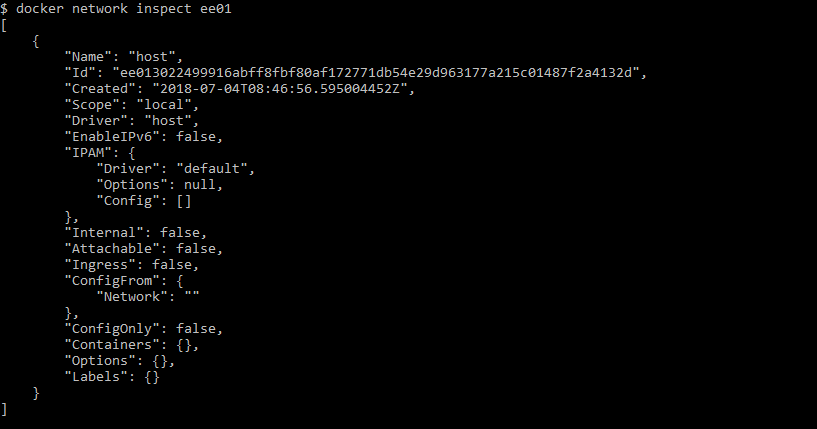


**docker system df –v**: To check the size of each container:



**docker network ls:** To list the available networks





**docker run -it --name myOracleLinuxCont oraclelinux bash**

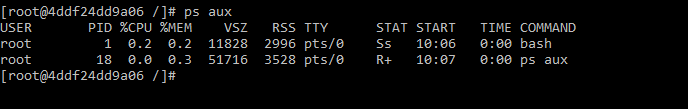
To instantiate an instance of oraclelinux OS

-i -> Interactive Mode

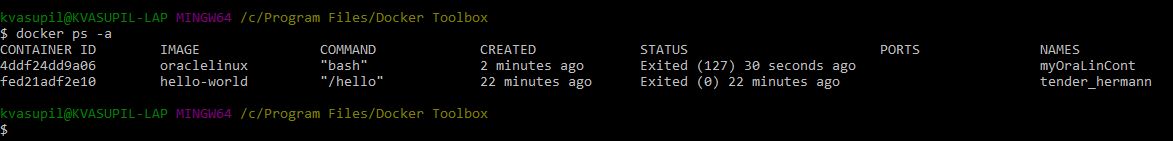
-t -> connect to the terminal

--name -> Explicitly specify a name to the container

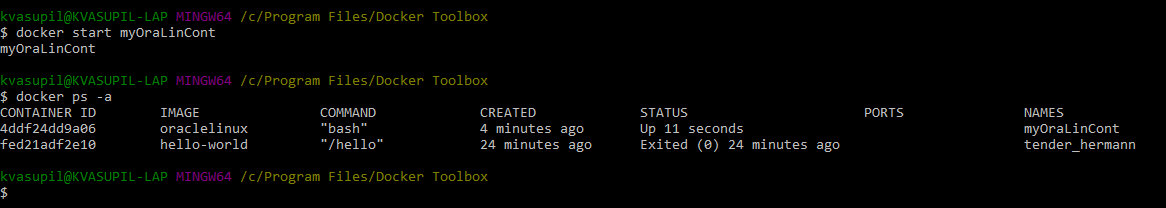
**Note: By default when an image is instantiated, we ‘ll be logged in as a Root user.**



Notice the user is “root” and also the output of ps aux command has two processes, one being the ps command and the other is the instance that was run.



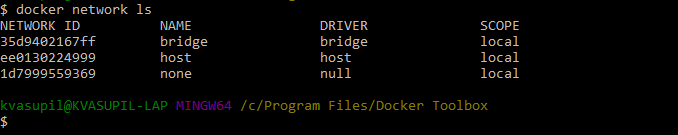
**docker start myOraLinCont:** To start the container named **myOraLinCont**

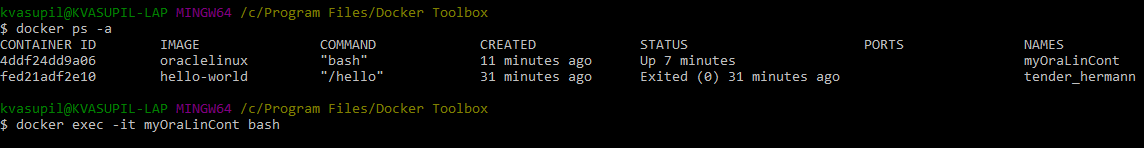


**docker network inspect bridge:** To list the containers connected to the network “bridge”

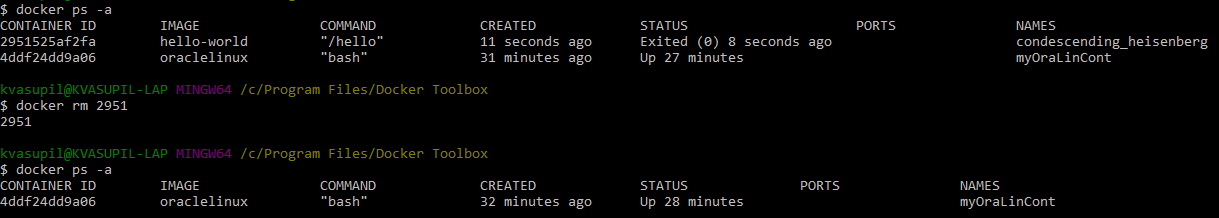


**docker network ls:** To list the available networks on the docker host

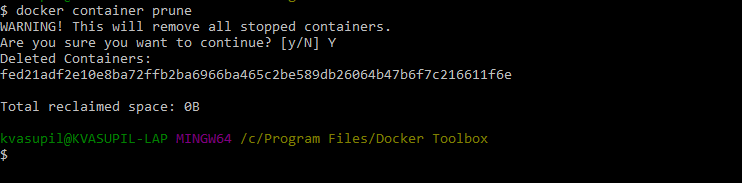


**docker exec -it myOraLinCont bash:**  To connect to a running container.****

**docker rm 2951:** To remove a container which starts with 2951.



**docker container prune:** To remove all stopped containers

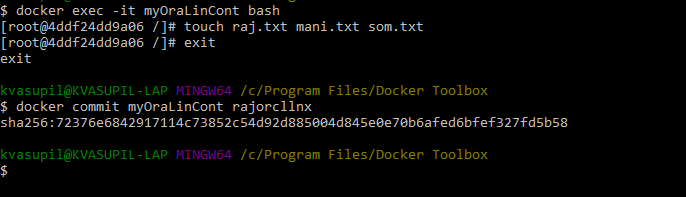


**docker rmi myOraLinCont:** To remove an image named myOraLinCont.

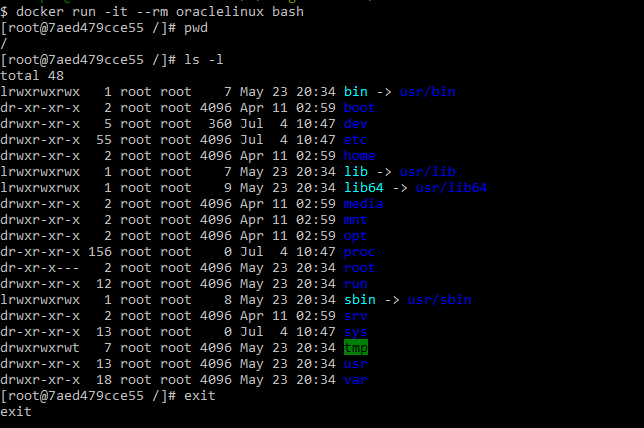


Customize the container

Docker commit name of cont name of img



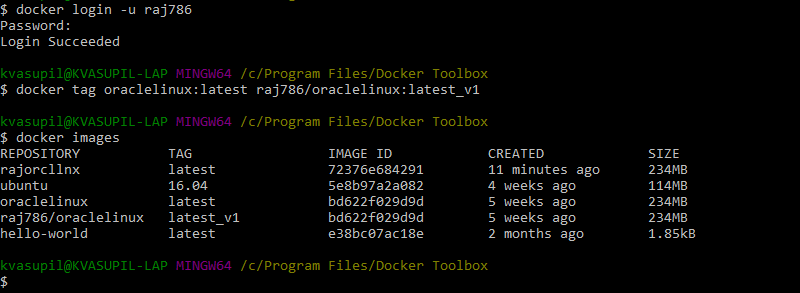
Instantiate another container for the same image and see if it contains the same files or not.

  
docker run --it --rm oraclelinux bash

**--rm is used to remove the container as soon as the container is exited.**

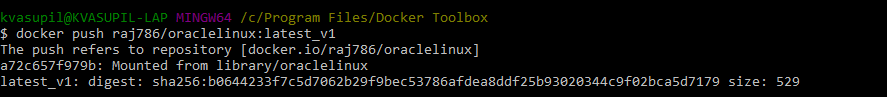
**docker rmi –force imagename:** To remove an image forcefully.

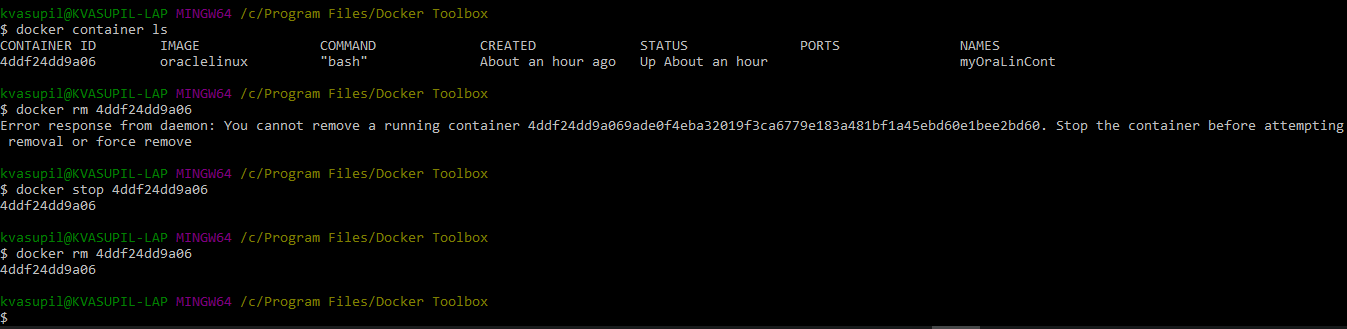
**Create a tag:**



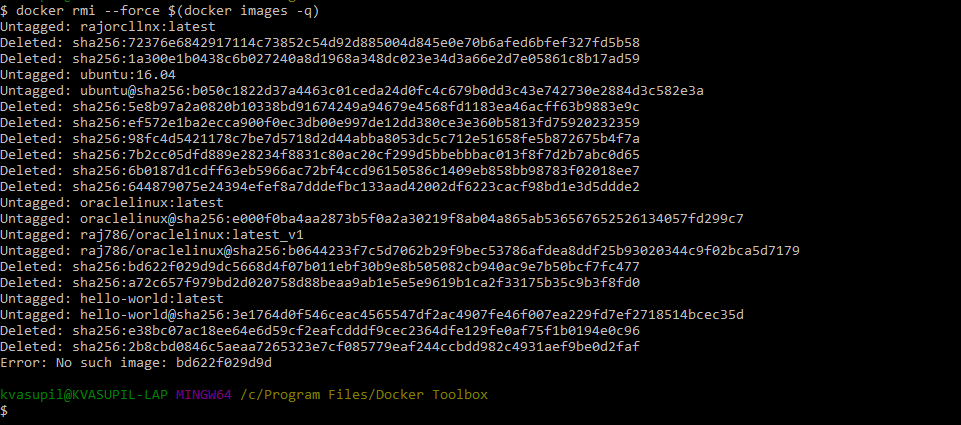
Notice the Image ID ‘s of the 3rd and 4th entries which are the same, because they are tagged to each other.

**docker push raj786/oraclelinux:latest\_v1:** To push the customized image into the Repo.

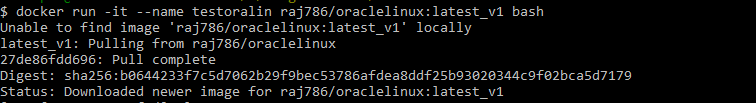


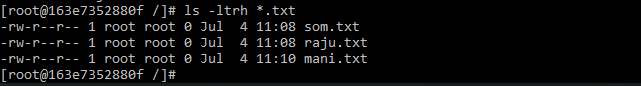


**docker rmi --force $(docker images -q):**To remove all images forcefully

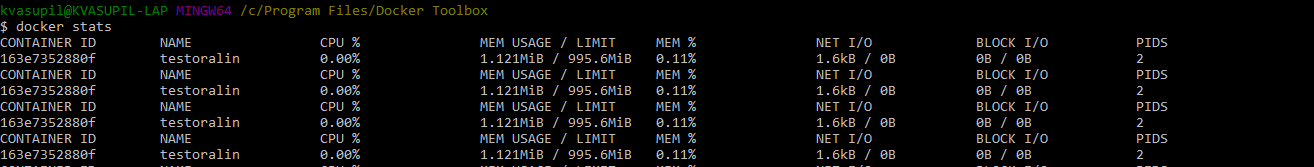


Instantiate a container from the customized image and connect to it, and check if the previously created files exist or not.





**docker stats:** To measure the docker engine utilization etc.,



There are 3 types of logs in Docker

1. **docker diff <container\_name>**

Gives the additions, changes and deletions made on the Base image.

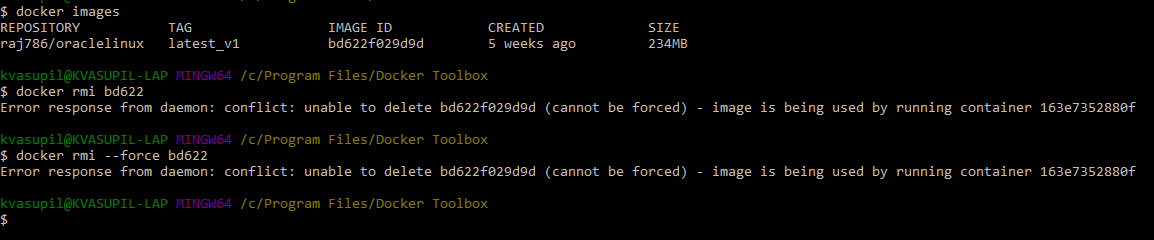
1. **docker inspect <container\_name>**

In the output of the above command there would be a logpath, contains info on how the container was accessed and etc.,

1. **docker logs <container\_name>**

Shows what the shell has activated on the container.

**Remove an image forcefully, when it has some symbolic references to it in some running containers.**



Use --force option to forcefully remove an image which has active references in one or more running containers.

**To copy the data from physical host machine into the container:**

docker cp <source\_path/file1> <source\_path/file2> <container\_name/id>:<destination\_path>

**Note:** Data cannot be copied from one container to another container. If that needs to be done, we have to copy the data from C1 onto the Host machine and then from C1 to C2.

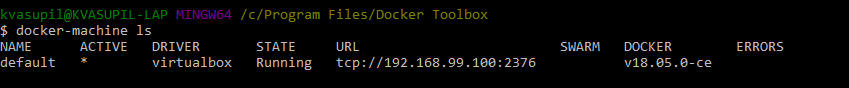
Docker machine can talk to another Docker machine using the below networks:

1. BRIDGE -> Containers within a Docker Machine use this to communicate.
2. Overlay -> For communication between Docker Machines.
3. MacVLAN -> Gateway, Lightweight Network Bridge.

**Default port of docker URL:**

Depending upon the kind of network used, the port may vary between 2376 and 6162.

2376 is the default port via which the communication is done between the host and docker machine.



**For an Overlay network the default port is 6162.**

**Dockerfile:**

A Dockerfile contains commands/instructions which specify what should an Image contain.

Recommendation is to write the docker commands in Capital case and the Linux commands in small case to distinguish between them.

**FROM** Ubuntu -> Will download the latest version of Ubuntu, by default

FROM Uubuntu:16.04 -> Downloads a specific version.

A Container can be based upon an Image which can be formed by a combination of multiple Images which are interlinked.

**MAINTAINER** -> To specify who maintains the Image.

**ENV** X=”20” -> To specify environment variables.

ENV X=”20 100” -> Multiple variables must be separated by a space, or else, X binds to the value 20, and skips 100.

**COPY** -> To copy data from the Local Host File System. It will automatically extract the tar files.

**ADD** -> To copy data from any source like Local Host System, Cloud, Network. If you don’t want to extract the tar files automatically, go with the “ADD” command.

**Note: Environment variables defined by ENV will be available only after the Image is built.**

**ARG** -> To define variables which are available prior to the Image creation or before the execution of FROM keyword.

**Eg: FROM UBUNTU:$VER**

Since the version keeps changing, if we want to download a version, we cannot use the ENV keyword, coz it is available only after the **FROM** keyword is executed. Solution is to use the keyword “**ARG**” which is available prior to the keyword “**FROM**”

**ENV = $VER**

**CMD** -> Command to be executed as soon as we enter a container. It executes the command and exits the container.

Eg: CMD {“/XYZ.sh”}

If there are multiple CMD’s, only the last CMD would be executed, skipping the rest. So, in order to execute a script with multiple command line parameters, better to go for ENTRYPOINT.

**ENTRYPOINT vs CMD:**

**Understand how CMD and ENTRYPOINT interact**

Both CMD and ENTRYPOINT instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of CMD or ENTRYPOINT commands.
2. ENTRYPOINT should be defined when using the container as an executable.
3. CMD should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container.
4. CMD will be overridden when running the container with alternative arguments.

If ENTRYPOINT and CMD are used together, then CMD should add a meaning/value to the ENTRYPOINT. I we want parameters to be passed as environment variables, use CMD along with it, or else, use it as is.

**Eg:1**

**ENTRYPOINT** [“java”]

**CMD** [“java class”, $VER, $ABC]

**Eg:2**

**ENTRYPOINT [“ls”]**

**CMD [“-l”, “-a”, “/usr/bin”]**

Here, CMD is adding a value to the ENTRYPOINT, if not the ENTRYPOINT execution would fail.

**RUN:** To perform some operations at the Image level, RUN is used.

**Eg: RUN apt-get update**

**To build a Docker Image:**

docker build –t <name of the Image> . -> Picks the Dockerfile from current location as it has the dot at the end.

docker build –t <name of the Image> -f <path to the Dockerfile>

**To exclude files from context:**

**.dockerignore** file contains the list of files that need to be excluded. First this file gets executed and then the Dockerfile would get executed.

1. Create a file named DockerFile.txt and copy the below content and save it.

ARG CODE\_VERSION=latest

FROM ubuntu:$CODE\_VERSION

MAINTAINER Raju Vasupilli

ENV desc="This is new version"

RUN ["/bin/bash","-c","echo $desc"]

CMD echo $desc | wc –c

1. Build an image out of the above file

$ docker build -t newimgcnt -f "D:/newdireg/DockerFile.txt" .



**If you want to access a file within an Image from another Image, Docker has an option “--from” .** However, communication from a Container to another Container is not possible.

**Objects within a Docker Machine:**

1. Container -> run, start, execute, pause, unpause, stop, top, rm, prune
2. Image -> pull, build, tag, push, rmi
3. Volume ->
4. Network
5. Repository
6. Engine -> info, version, stats

ls, ps and top are common commands for all the above.

**To export a container onto a different machine:**

docker save –o a.tar Ubuntu

docker rmi –force Ubuntu

docker load < a.tar

docker export/import will save/load the container without the layers.

**Note:** Docker being Open Source, Layers are handy as we can customize a particular layer accordingly as per our need.

**For Windows Container Objects:**

CMD [“powershell.exe”, “cmd”]

**Ports:**

1-29,999 are Static Ports and 30,000 – 49,999 are Dynamic Ports.

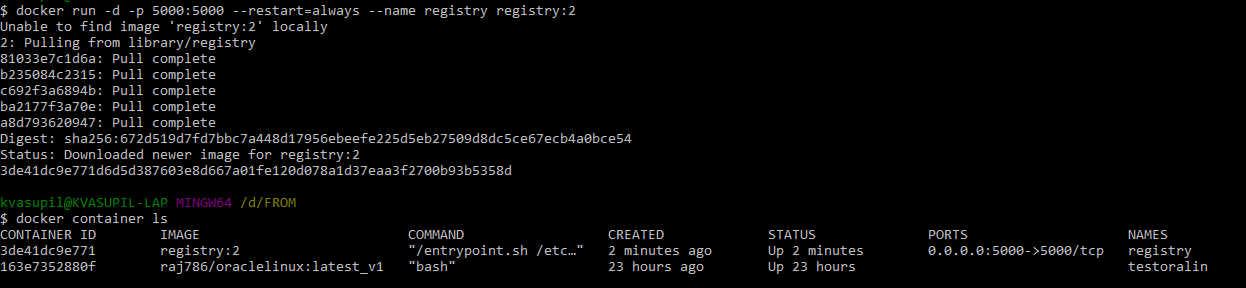
Restricted Ports: 2376, 6162 and 4002

Docker uses -p for static ports and –P for Dynamic Ports.

The default Container port is 5000

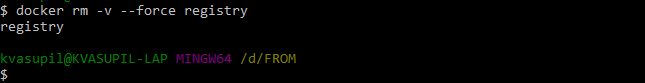
-p -> We have to assign a port to expose the services hosted within a container.

-P -> We don’t have to assign any ports

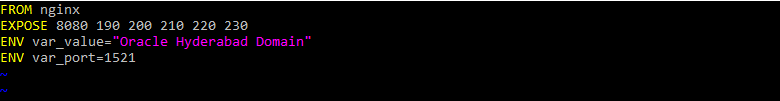


Anything which is inside the registry once, stays there forever. Even though we remove the images, the service would still be accessible. Because the Local Registry (UDF) still servers your DML Softwares across your distribution.

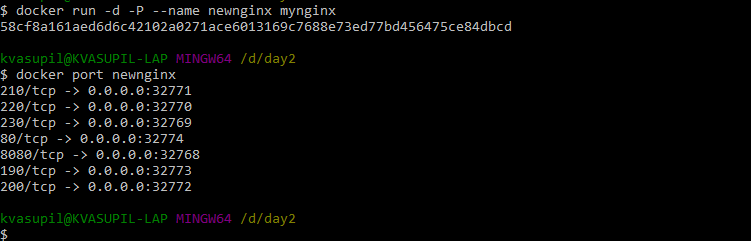
docker rm –v --force myregistry



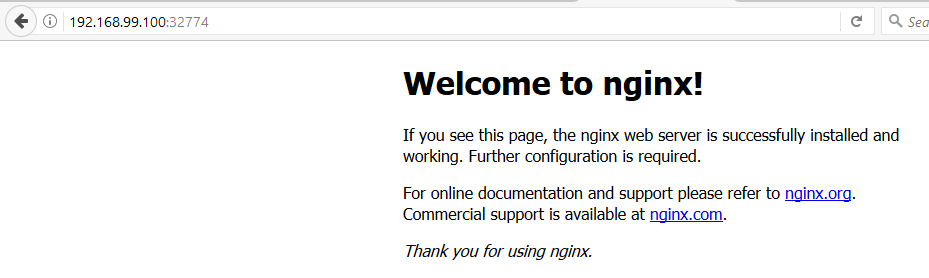
**Assign ports Dynamically:**



In the above screenshot, port 8080 is the service port and the rest all are reserved for future use.



Now access the Nginx service in the browser on the port 32774.32774 is exposed to port 80.



docker run -d --rm -e var\_port=1000 -e new-var=HYD newnginx

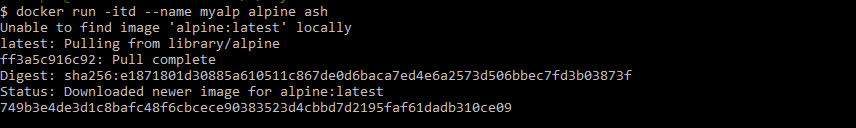
The option –e can be used to edit the value of a variable within a container.

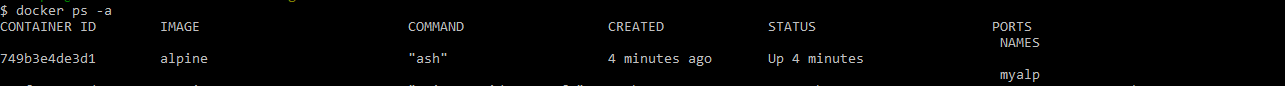
**Static Binding:**

docker run –d –p 7001:80 –p 7100:100 –p 7200:120 –name testnginx nginx

**How to keep a container always running:**

docker run -dit --name myalp alpine ash

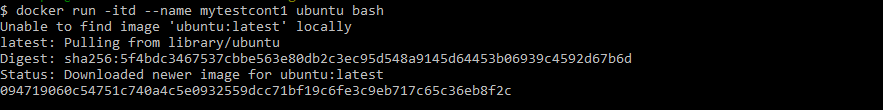




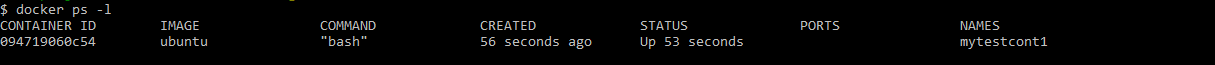
If you want to connect to the container and execute any commands within it, use the exec command

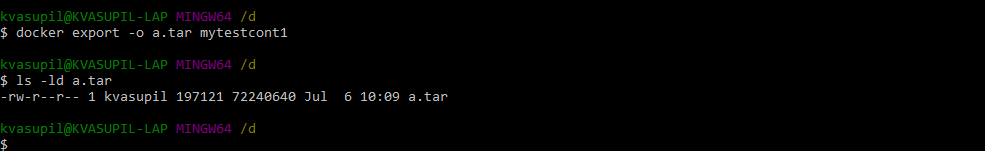


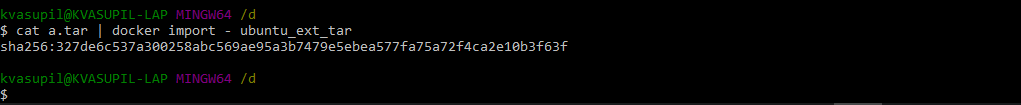
ash is the shell for Alpine



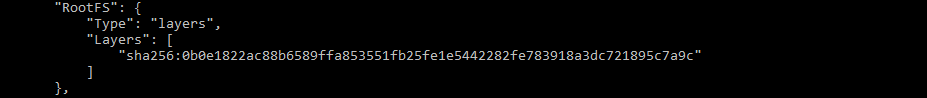
docker ps –l lists the last container used







In the case of export and import, only one layer would be there, unlike save and load option.



**docker attach:**

docker attach <running\_container>

This comes handy especially when there is a performance issue with the container. We can attach to the current shell that is running in the container. In the case of exec , we get a new shell

Once we use attach command, the moment we exit the container the container stops executing. In the case of exec command, we get a new shell and the moment we exit, only the new shell exits.

Container in Restarting State for longer time:

If we see a particular Service restarting for a longer period of time, then there could be a problem with the Service attached to the Container and is not the fault of the Container.

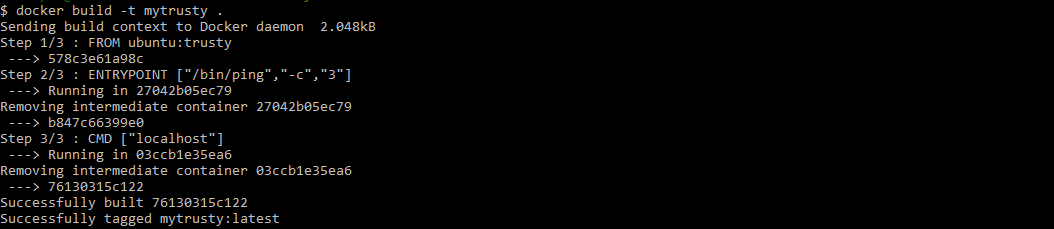
**Note:** A Docker machine can have multiple registries in it, but could only have just one Repository. The Local Repository.

**Note:** A Registry is nothing but the grouping of all images residing on the Docker Machine. If in case, we want to copy all images onto a different host, we can copy the registry alone onto the destination host.

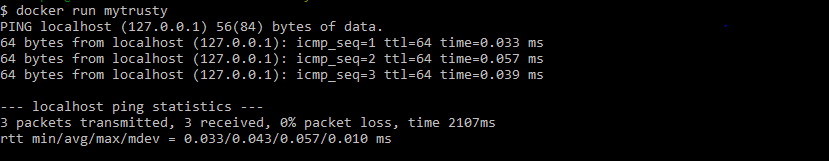
**Write a Docker File which will ping an IP, when passed as an argument, and pings the localhost when an argument is not passed.**



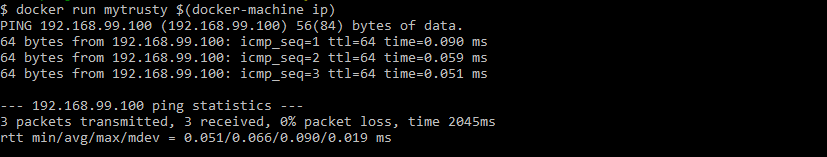
Build the image



Test the image without a parameter

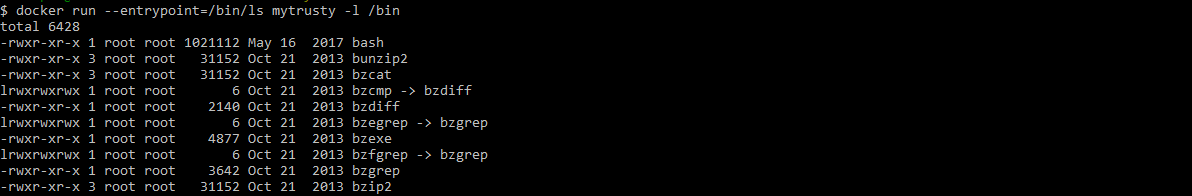


Test the image by passing an IP



**Ovveride an ENTRYPOINT:**

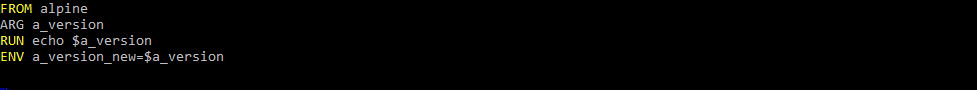
We can override an entrypoint using the –entrypoint option in the run command, however, if the subsequent entries doesn’t complement the entrypoint, the command would fail.



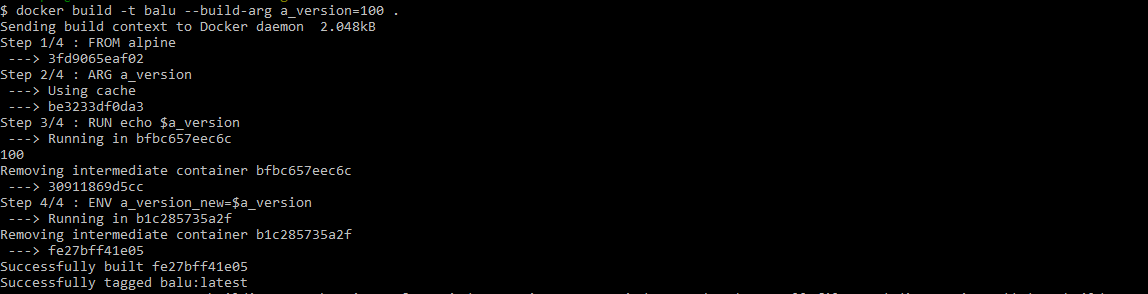
Instead of “-l /bin”, if something like pwd or something else is given which doesn’t add a meaning to “ls”, the command would nt work.

**Override Arguments:**

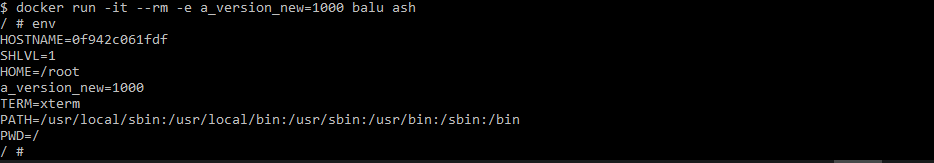
vi Dockerfile



Build the image by passing --build-arg



Here we have overridden the argument a\_version with a\_version\_new, so we don t see a\_version in the output of command “env”



If we don’t want anyone to update a Container, we can go with a Read-Only Containers, but we won’t be able to create any file within it.

**Move a java jar file into a Docker Container:**

vi DockerFile

FROM java:8

WORKDIR /

ADD HelloWorld.jar HelloWorld.jar

EXPOSE 8080

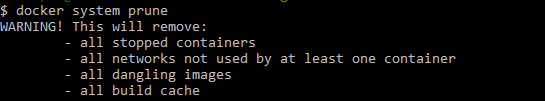
CMD java –jar HelloWorld.jar

CMD [“/bin/bash”,”-c”,”echo this is from JVM”]

“WORKDIR /” is similar to “RUN cd /”

ADD sourcefilepath destfilepath

**docker system prune:** It will remove all stopped containers, networks, dangling images and cache.



**Volumes:**

A Volume is nothing but a set of directories.

Docker has 3 types of Volumes:

1. Host Bind Volumes
2. Docker Volumes -> Managed by Docker Machine
3. Flocker Volumes

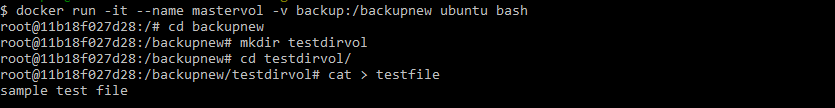
**Docker Volume:** Is managed by the Docker Machine. Ideally 40% of the Disk Space is occupied by the Docker Volume.

Doesn’t allow the Containers to be removed while the Volumes are Active. So there is a cohesiveness between Containers and the Volumes associated with it.

We can control a Volume to either be Read/Write or Read Only when two containers share a single Volume.

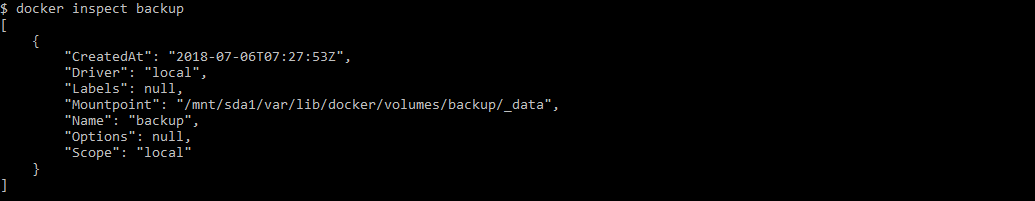
**How to create a Volume:**

-v and --mount options can be used to create a Volume.

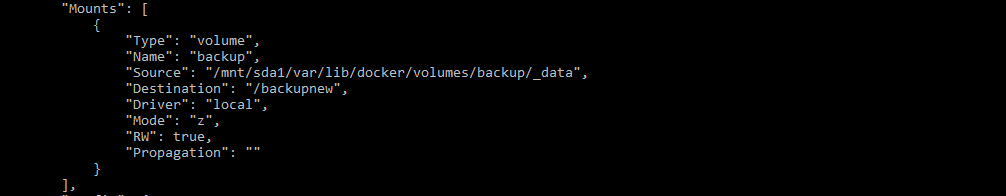




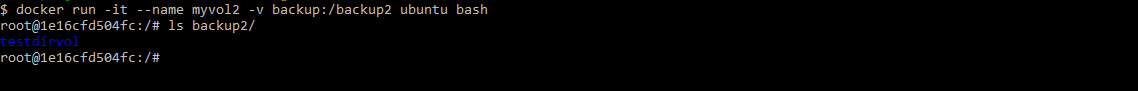
From now onwards, anything that gets created in backup folder would be available within the container in the backupnew folder and vice-versa.



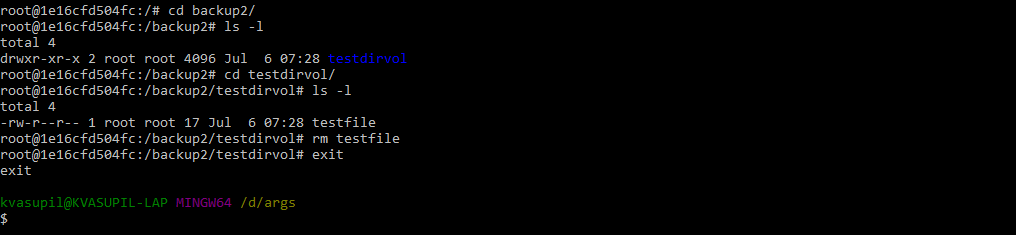
Now, if we want to find out to which folder this volume is bound to within the container, use the inspect command.

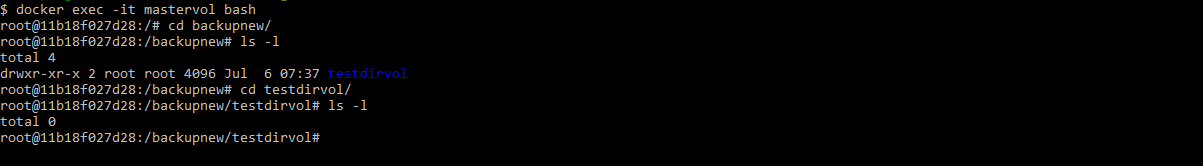


Now create another container and add the volume backup to it. It ‘ll create a link to the backup folder and won’ t create a new Volume as such.



Now delete the file testfile and attach to the previos container and see if the file exists or not.

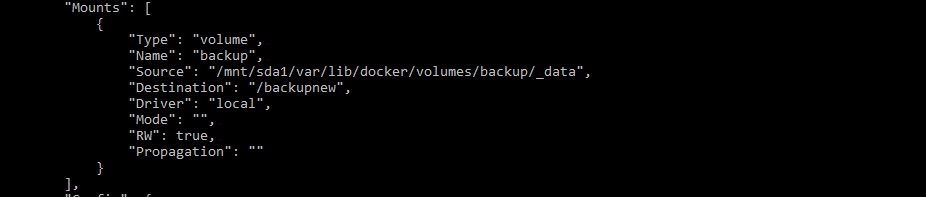




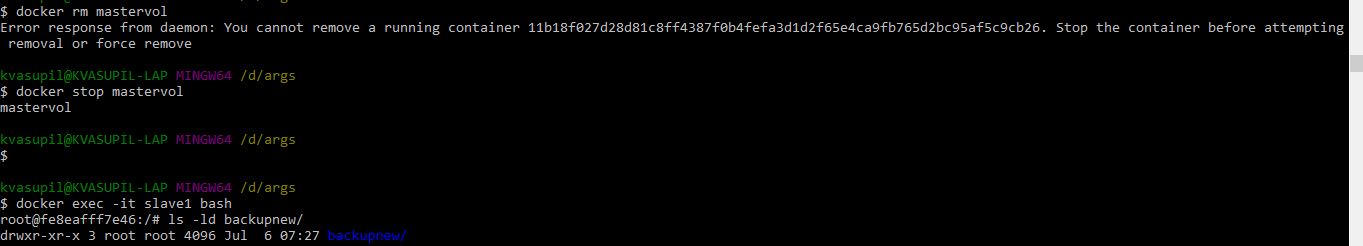


--volumes-from option will attach an existing volume that is linked to an active container to the new container.

docker inspect slave1



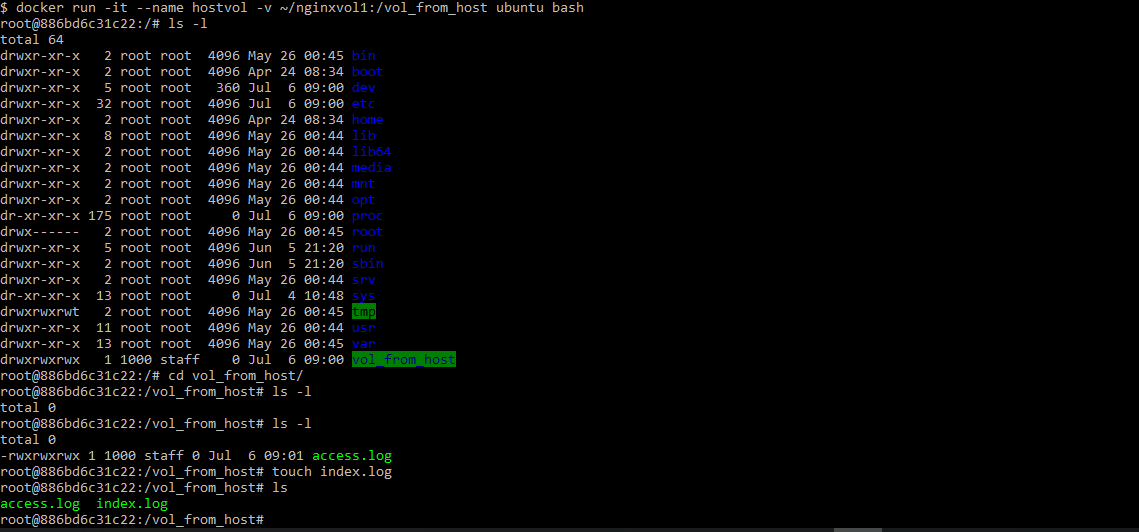
Now, if we remove the master container, the data would still be available in the new container.



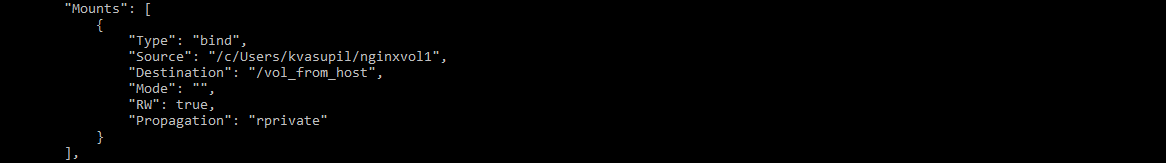
-v backup:/backup ->If the entry next to –v is not a path, then Docker deems it as a Docker Volume, if it has a path, then it considers it as a Host Bind Volume.

-v ~/userdata:/userdata

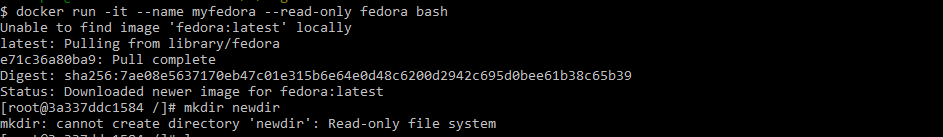
**Host Bind Volumes:** Bind a path on the local host with a Container.



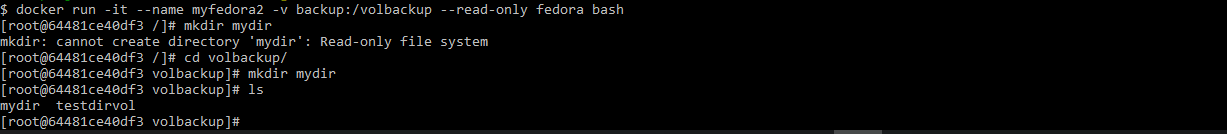
docker inspect hostvol will list the details pertaining to the mount as below:



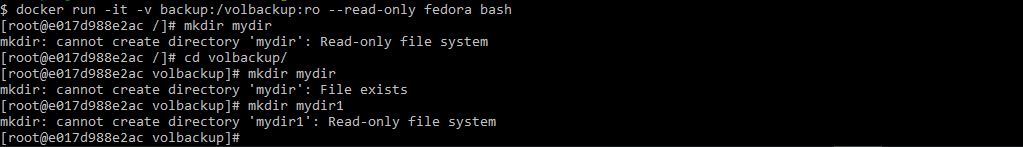
**Create a Read Only Container:**



**Create a read only Container with a Volume:**



**Create a Read Only Container with a Read Only Volume:**

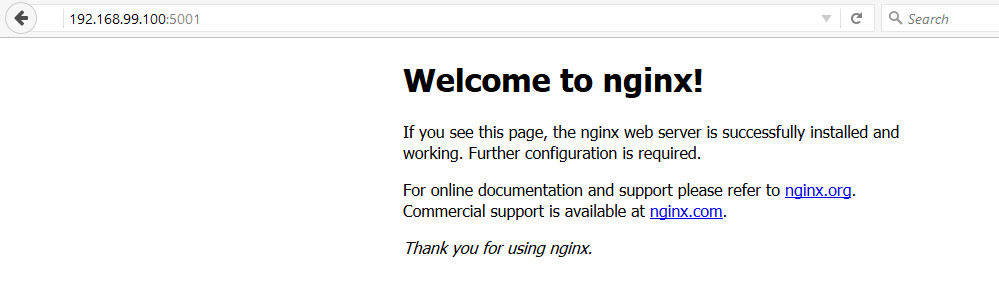


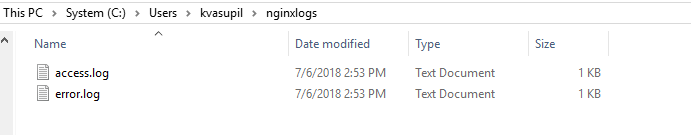
Create a container and a Volume



To fetch the IP pertaining to the Docker Machine, use the below command:







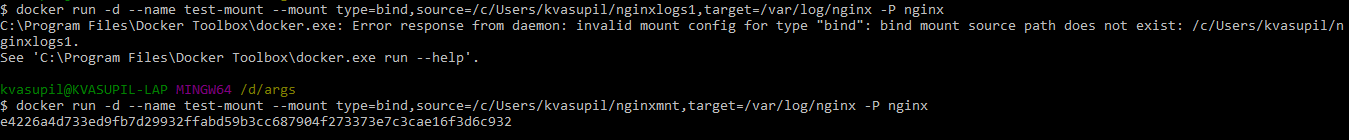
Now, let’s just change the command as below:

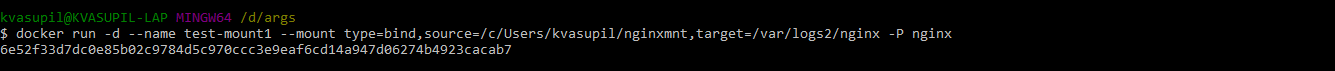


The problem with –v option is that it creates the path specified if that doesn’t exist either on the Local host or in the Container, but does not report an error, which is not always helpful.

/var/log/nginx is the right path but not /var/logs/nginx does not exist. Instead of reporting an error, -v creates it.

The solution for the above is to use **--mount** option.





If the target/destination path doesn’t exist, both “-v & --mount” options will create it.

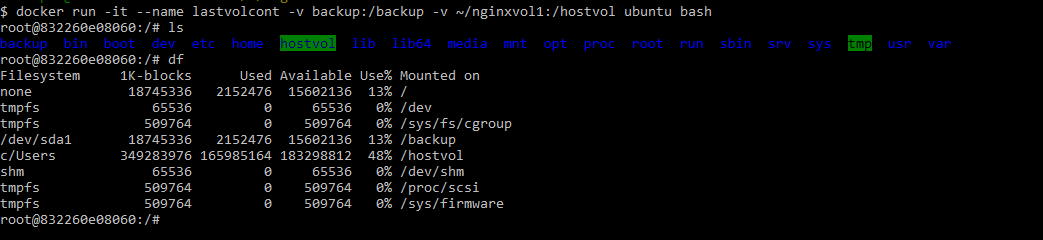
If the Source path doesn’t exist, then -v will create it, but --mount will report an error, but doesn’t create it.

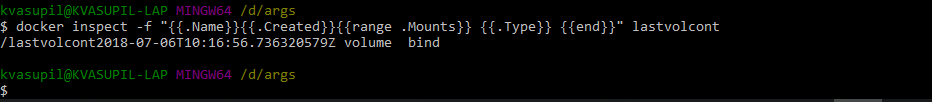
**To create a read only mount:**



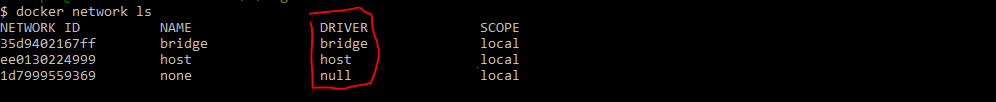
|  |  |  |
| --- | --- | --- |
|  | **-v** | **--mount** |
| Path | Relative Path | Absolute Path |
| Syntax | bind x:y | Verbose type-,src=,target= |
| File/Spec | Check/Creates if Not Available | Source is cheked/ Error if Not Available |
| Privilege | Root Permissions | - |
| Dependencies | OS Bind | - |

**Create two volumes in a container at one go:**





**Docker Network Drivers:**



Docker0 is the default network in a Docker Machine.

Bridges maintain only “UP” state containers.

By default, all containers communicate as they‘re part of the default bridge Docker0. If we don’t want the containers to communicate with each other, then we go for custom bridge.

**Provide volume related information for all running containers in a Docker Container.**

$ docker ps -a -q | xargs docker container inspect -f '{{.Name}}{{range.Mounts}}{{.Type}}{{.Name}}{{end}}'

/lastvolcontbindvolumebackup

/myalptestvolumebackup

/test-mount1bind

/test-mount2bind

/test-mountbind

/nginx2bind

/tnginxbind

/inspiring\_stallmanvolumebackup

/myfedora2volumebackup

/myfedora

/hostvolbind

/slave1volumebackup

/myvol2volumebackup

/mastervolvolumebackup

/elegant\_mirzakhani

/sharp\_bartik

/zealous\_tesla

/laughing\_ritchie

/mytestcont1

/myalp

/newnginx

/jovial\_goldberg

/angry\_saha

/vigorous\_sinoussi

/modest\_goldwasser

/boring\_stallman

/testoralin

kvasupil@KVASUPIL-LAP MINGW64 /c/Program Files/Docker Toolbox

$

**Docker Compose:**

Docker Compose is an Automation Utility. Services that need to be run are given as a command based instructions written in YAML.

YAML is a simple data serialization language.

Docker Compose must be installed separately.

**docker-compose.yml** is the file which contains the list of things that need to be done.

Docker version 17 & 18 will work on Yaml version 3

14-16 2

< 14 1

Docker Compose is not a Docker standard, but is a utility added for ease of administration.

Yaml is very specific of indentation, so better to use Notepad++

Kubernetes supports XML, JSON, YAML, Python, Java etc.,

In Yaml all keywords must be written in small case only.

Docker Compose is a CLI. It has doccker-compose.exe which when executed runs docker.exe.

docker-compose up

docker-compose down

docker-compose build & docker-compose rm

**vi docker-compose.yml**

version: '3'

services:

database:

image: mysql

ports:

- "3306:3306"

environment:

- MYSQL\_ROOT\_PASSWORD=admin

- MYSQL\_USER=user

- MYSQL\_PASSWORD=password

- MYSQL\_DATABASE=demodb

web:

image: nginx

docker-compose up

**When docker terminal stops responding/misbehaves, use the below command:**

C:\Users\kvasupil>docker-machine stop default

Stopping "default"...

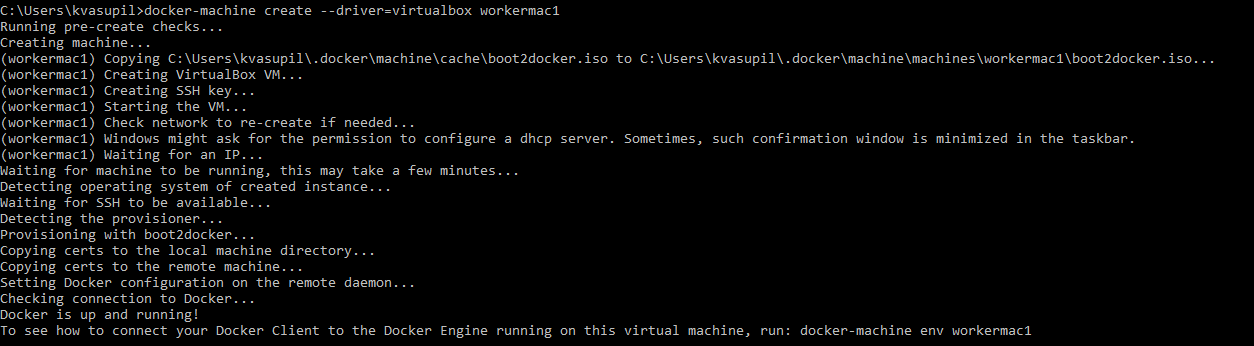
Machine "default" was stopped.

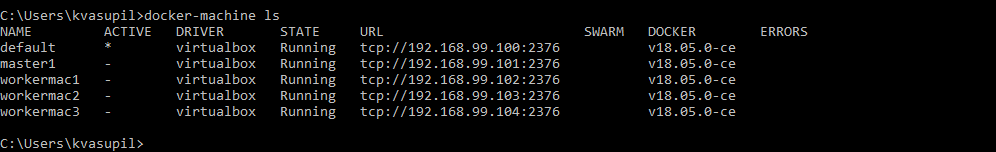
C:\Users\kvasupil>

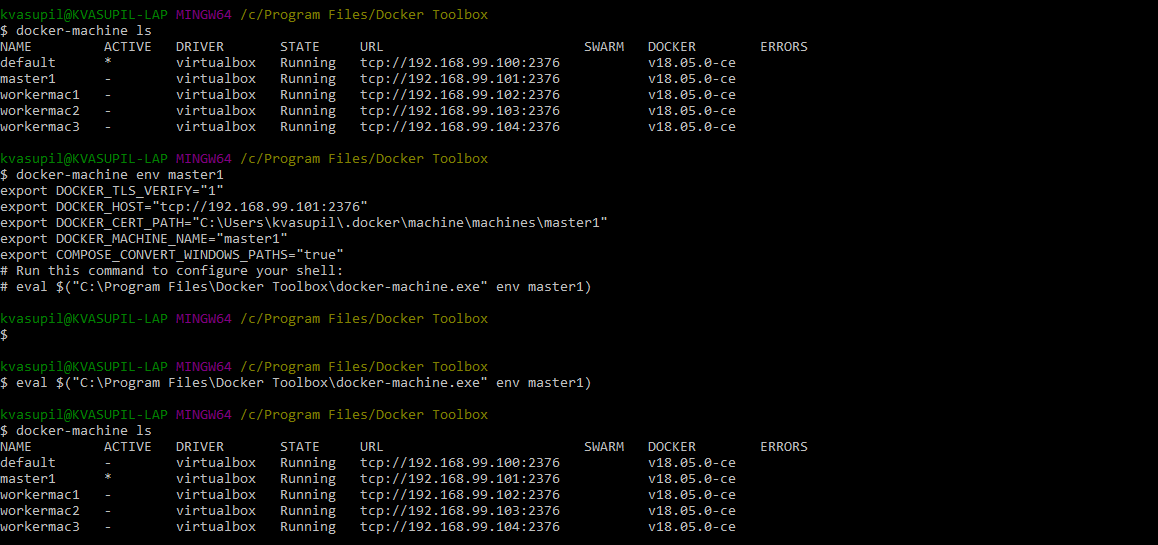
**Docker build vs Docker Compose:**

docker build is for a single image, but docker compose is for multiple images.

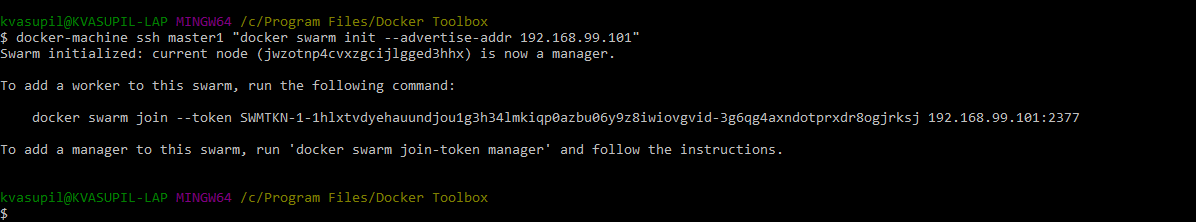
**Docker Swarm:**



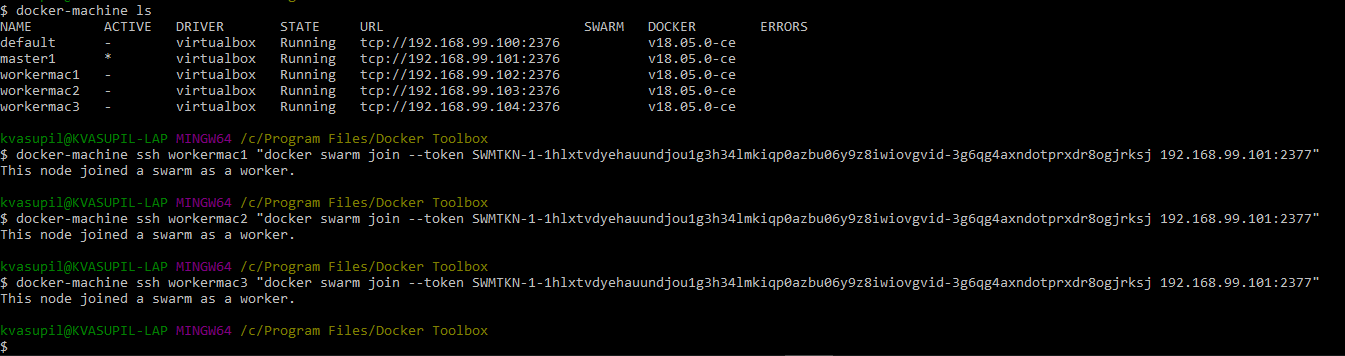




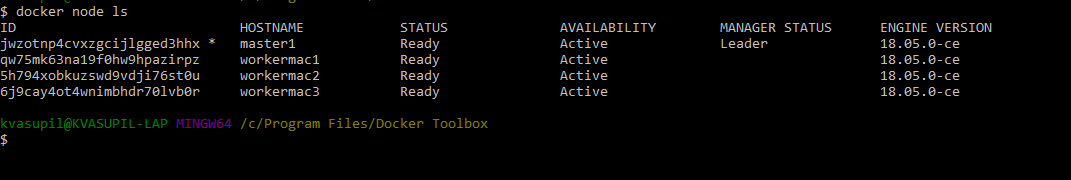
**Create a Docker Swarm:**



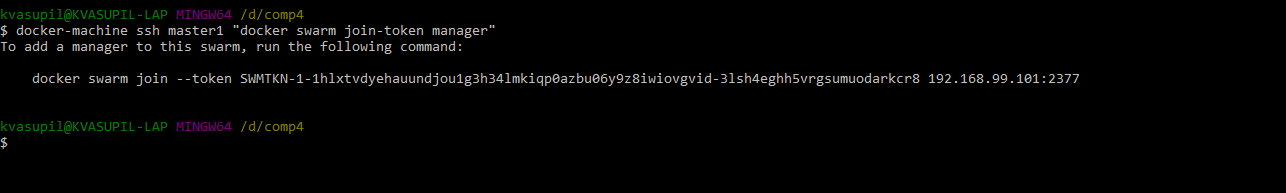
**Add three worker nodes to the swarm:**

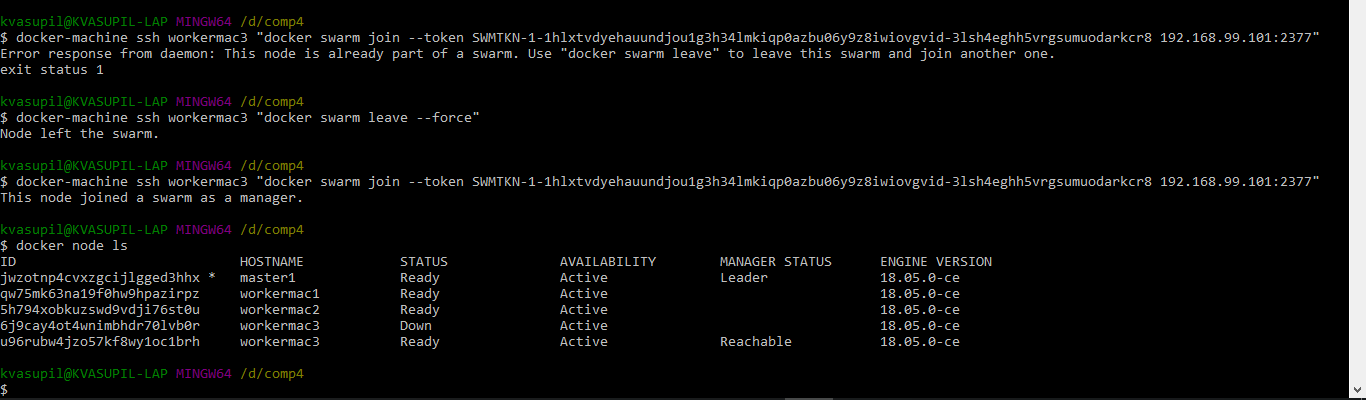


**Check Docker Swarm Status**



**To add a Reachable Manager Node**





**Deploy services onto the docker swarm:**

notepad docker-compose.yml

version: '3'

services:

redis:

image: redis:alpine

deploy:

resources:

limits:

cpus: "0.50"

memory: "50M"

reservations:

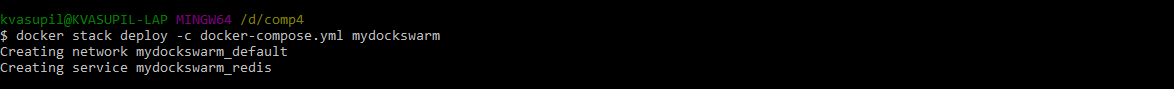
cpus: "0.25"

memory: "20M"

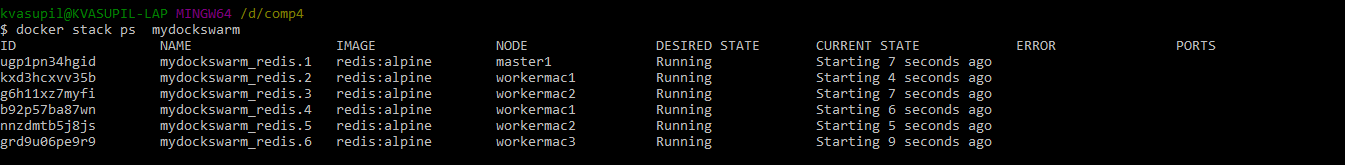
mode: replicated

replicas: 6

**Command to deploy:**



**Verify the deployed services:**



**To retrieve the swarm token, connect to the master machine and execute the below command**

docker-machine ssh masteermac1 “docker swarm join-token worker”

A Docker Daemon/Machine cannot be tarred and exported onto elsewhere.

**Issues with Docker:**

1. Patterns-limitations
2. Plugins
3. Security
4. Orchestrator-Automation
5. Rules
6. Too many concepts
7. Upgrade
8. Limitation – JSON/YAML
9. GUI Engine
10. Governance – 100% Availability is a challenge
11. Too many IP’s to manage
12. Ports
13. Binds
14. Volume is very weak
15. Need to buy Enterprise Edition to implement Docker when new versions come up

**Advantages to Docker:**

1. Portable
2. Scalable
3. Defined
4. Available Orchestrator
5. Move to 1x to 10x directly
6. Elasticity of Docker Machine
7. Rules/Protected

**7 Components of Docker:**

1. Image
2. Container
3. Volume
4. Network
5. Repository
6. Docker Machine
7. Stack/Swarm

Master can Orchestrate and Operate.

**Kubernetes:**

KN has been Google owned since 2001. It has a UPL/ULP License.

KN is not going to build an application, it will only host an application and manage it.

Docker Machine ~ KN **Node**

KN **Master** will only do Orchestration and will not do any operations, unlike Docker Master, and are delegated to the worker nodes.

Docker Reachable ~ KN **Shadow Master**

**MiniKube** is a machine whose Driver is VBOX.

KN Node contains a runtime, which could be any container mgmt. software. By default it is Docker, but can be modified to run other softwares as well.

**Kubelet** is the Agent.

**Fluentd** is a plugin which contains all the log entries of nodes.

**Kube Proxy** is a process, which diverts the traffic to the Nodes.

Master is connected to the Node by **Ingress Controllers**.

There are 4 Ingress Controllers:

1. Nginx
2. Kong
3. Trafeik
4. Ambassador – Used for MAC and IOS applications

**Common Repository:** It contains all the images managed by the Nodes. It is managed by nginx.

**POD:** Pod is a group of containers.

**Namespace:** Is a logical collection of PODs.

**Etcd:** It contains threshold availability of the Nodes. It contains the Real-Time state of the Nodes.

**Eg:** If the threshold is set to 70%, once the threshold hits 70% on a Node, the next incoming traffic would be redirected to a different node.

Confidential Object is called **SECRET**. Non confidential object is called **OPAQUE**. Both these information is stored in etcd.

The default port for Master is **8443**.

To install Kubernetes, use the below commands:

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

<https://github.com/kubernetes/minikube/releases>

<https://storage.googleapis.com/kubernetes-release/v1.10.0/bin/windows/amd64/kubectl.exe>

1. Rename minikube-amd-64.exe to minikube.exe
2. Open Command Prompt, run “minikube start”
3. cmd prompt, run “kubectl cluster-info”
4. minikube dashboard