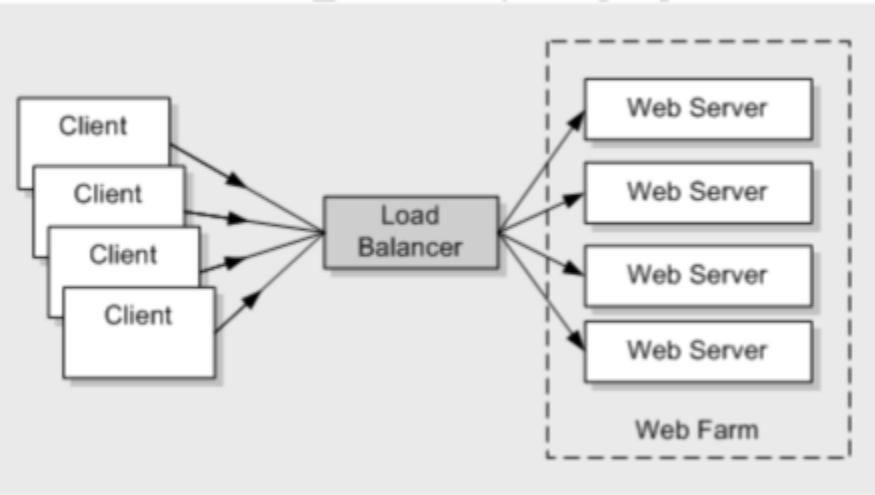
Docker Applications

In today's world, we are all surrounded by apps and websites. We use our smartphones and computers to browse around internet and use all the web services through our mobile apps or browsers. All these millions of web based data is coming somewhere far from some computers which would be located in some datacenter. We generally call them servers; these servers could be those physical machines that we see racked up in a datacenter with all those flashing lights and cables.

If we take some examples like Amazon, Google, Netflix, Goibibo etc, all these businesses are running on applications or we can say their applications are their business. This makes a very important point that we cannot separate their business with their application.

Application needs compute resource to run and that comes from the server where they hosted their application. In olden days when we did not have any virtualization or cloud computing we use to run them directly on a physical server.

So, if I want to host an application on 10 webservers, I need ten physical servers under load balancer serving the web traffic

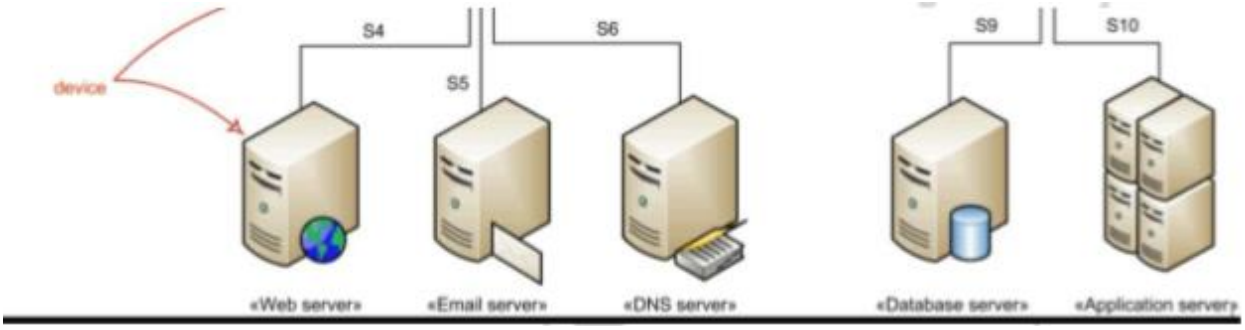


These servers are very expensive and we need to do lot of maintenance for them.

- We need to procure a server. A process where we place an order for the purchase.
- There is Capital expenditure or CapEx required.
- There is Operational expenditure (OpEx), like cooling, power, admins to maintain that server farm.

So, if I want to increase the capacity and add more servers I need to spend money and time on above mentioned process. This is very common as business starts from very small user base and then users/consumers traffic increases if business is doing well.

We deploy one application per server because we want our applications to be isolated. For example, if we need web app, db app and few backend apps. We may end up having multiple physical system each running a single instance of that app.



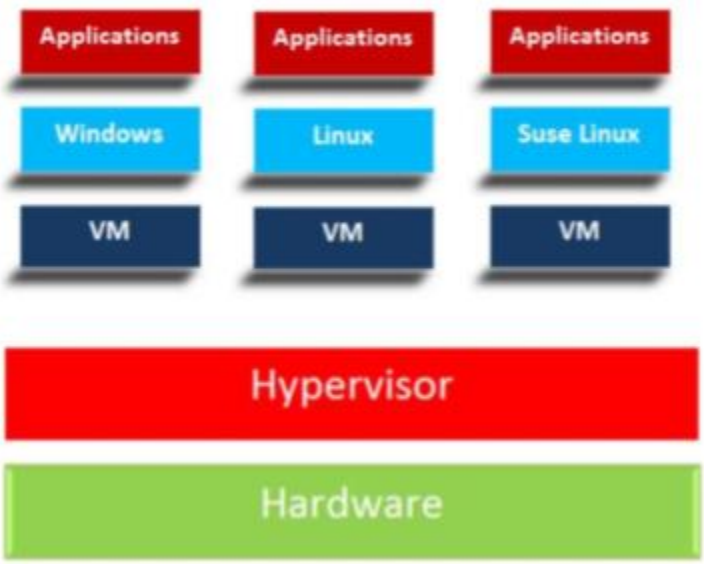
So, every time we need a new app to run we buy servers, install OS and setup our app on that. And most of the time nobody knew the performance requirements of the new application! This meant IT had to make guesses when choosing the model and size of servers to buy.

As a result, IT did the only reasonable thing - it bought big fast servers with lots of resiliency. After all, the last thing anyone wanted - including the business - was under-powered servers.

Most part of the time these physical server compute resource will be under-utilized as low as 5-10% of their potential capacity. A tragic waste of company capital and resources.

Virtualization Revolution.

VMware gave the world the virtual machine and everything changed after that. Now we could run multiple applications isolated in separate OS but in the same physical server. In the virtualization chapter, we discussed the benefits and features of virtualization, The Hypervisor Architecture.



Problems with Hypervisor Architecture :

Now we know that every VM has its own OS, which is a problem. OS needs fair amount of resources like CPU, Memory, Storage etc. We also maintain OS licences and nurse them regularly like patching, upgrades, config changes. We wanted to host an application but collected good amount of fats over our infra, we are wasting OpEx and CapEx here. Think about shipping a vm from one place to other place, this sounds a great idea that if we could bundle everything in a vm image and ship it so the other person doesn't need to setup vm from scratch can directly run the vm from image. We did it in Vagrant chapter where we download preinstalled vm and have just run it.

But these images are heavy and bulky as they contain OS with the app. Booting them is a slow process. So being portable it's not convenient to ship the vm every time.

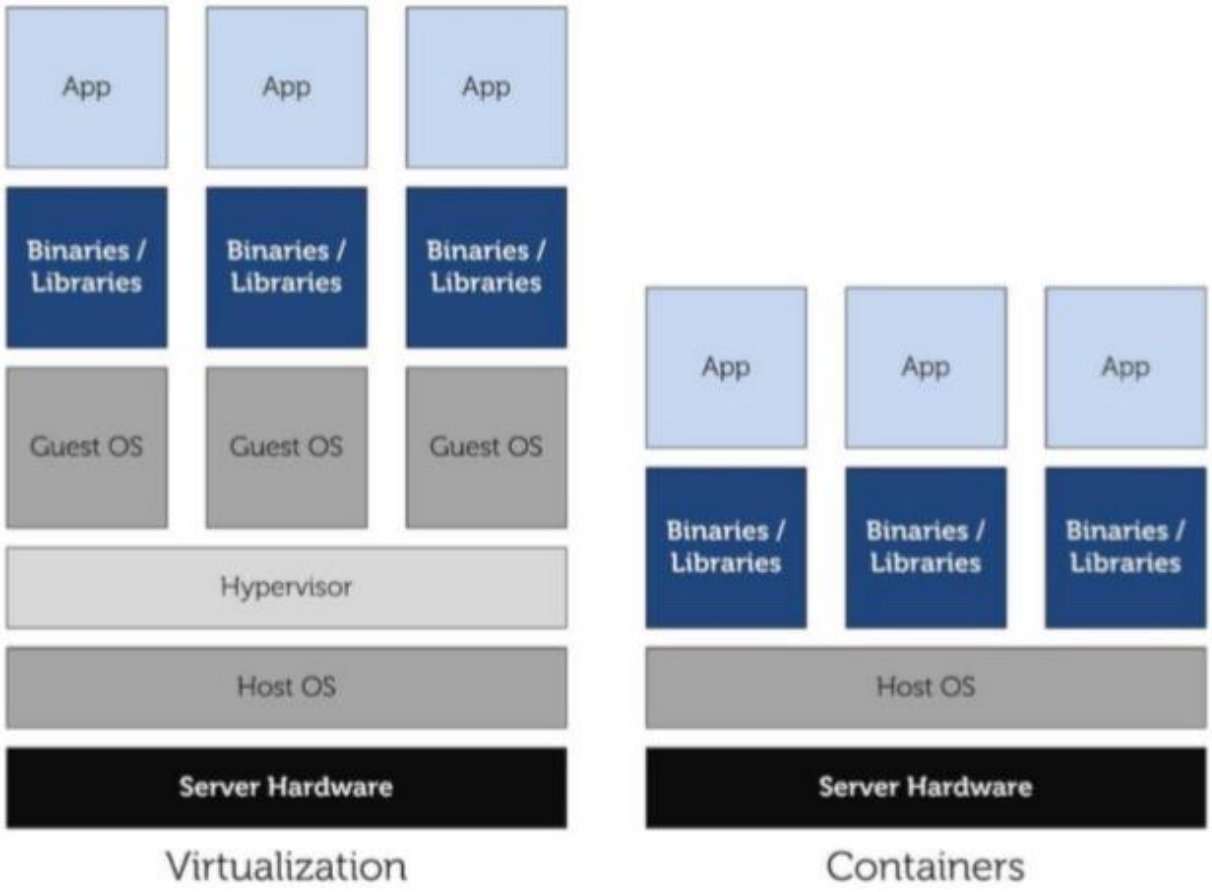
Shipping an application bundled with all the dependencies/libraries in an image without OS. Hmm, sounds like we solved a big problem there. That's what containers are.

Think about setting up an application in a vm or physical machine. We need OS setup, dependencies, application deployed and some config changes in the OS. We follow a series of steps to setup all these like setting up a LAMP stack. If we could bundle all these into one container and ship it, then admins don't need to do any setup on the target, all we need to do is pull a container image and run it.

Containers :

If virtual machines are hardware virtualization then containers are OS virtualization. We don't need a real OS in the container to install our application. Applications inside the containers are dependent on Host OS kernel where its running. So, if I have hosted java application like inside the container it will use all the java libraries and config files from container data, but for compute resource its relied on the Host OS kernel. Containers are like other processes that run in an Operating System but its isolated, its processes, files, libraries, configurations are contained within the boundaries of the container.

Containers have their own process tree and networking also. Every container will have an IP address and port on which the application inside container is running. This may sound like a virtual machine but it's not, remember VM has its own OS and containers does not.



Containers are very lightweight as it just has the libraries and application. So that means less compute resource is utilized and that means more free space to run more container's. So, in terms of resources also we are saving CapEx & OpEx.

Containers is not a modern technology, it was around us in different forms and technologies. But Docker has brought it to a whole new level when it comes to building, shipping and managing containers.

Dockers

Docker, Inc. started its life as a platform as a service (PaaS) provider called dotCloud. Behind the scenes, the dotCloud platform leveraged Linux contain-ers. To help them create and manage these containers they built an internal tool that they nick-named “Docker”. And that’s how Docker was born!

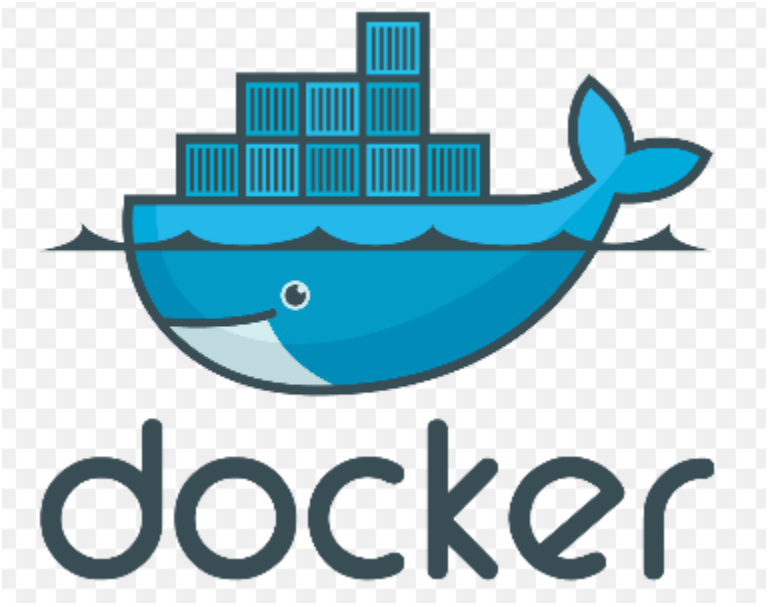
In 2013 the dotCloud PaaS business was struggling and the company needed a new lease of life. To help with this they hired Ben Golub as new CEO, rebranded the company as “Docker, Inc.”, got rid of the dotCloud PaaS platform, and started a new journey with a mission to bring to Docker and containers to the world.

Docker relies on Linux kernel features, such as namespaces and cgroups, to ensure resource isolation and to package an application along with its dependencies. This packaging of the dependencies enables

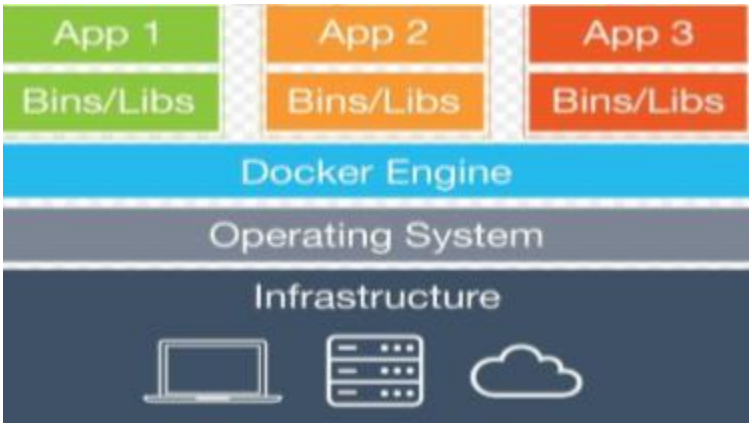
an application to run as expected across different Linux operating systems. It's this portability that's piqued the interest of developers and systems administrators alike.

But when somebody says "Docker" they can be referring to any of at least three things:

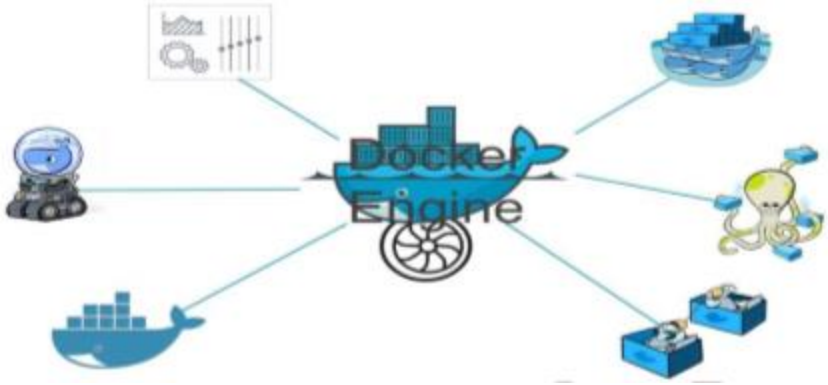
1. Docker, Inc. the company
2. Docker the container runtime and orchestration technology.
3. Docker the open source project

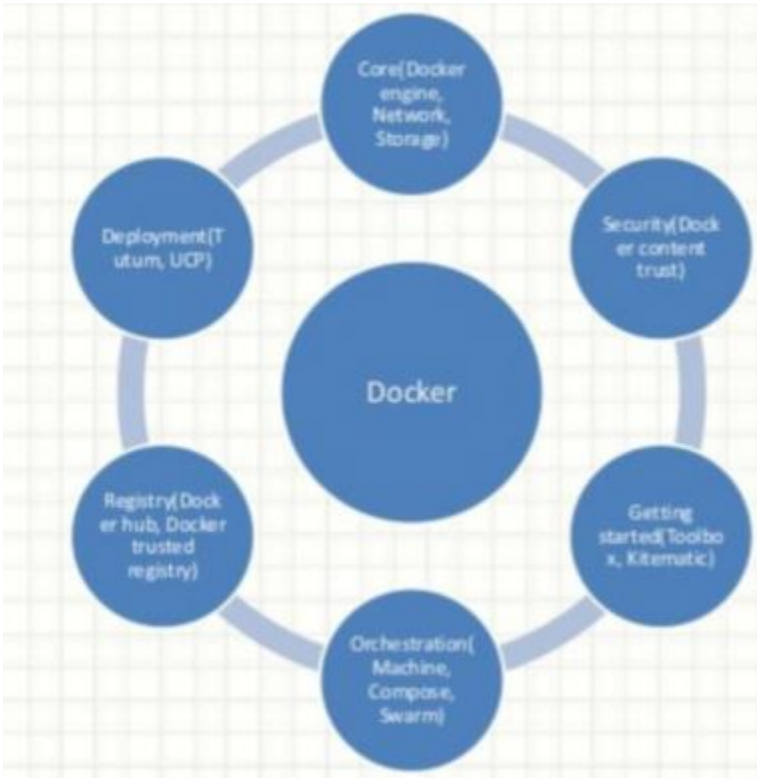


When most of the people talk about Docker they generally refer to the Docker Engine. Docker engine runs and orchestrate containers. As of now we can think docker engine like a hypervisor. The same way as hypervisor technology that runs virtual machines, the Docker Engine is the core container runtime that runs containers.



There are so many Docker technologies that gets integrated with the docker engine to automate, orchestrate or manage docker containers.





Installing Docker :

- Docker can be installed on Windows, Mac and Linux OS.
- We will install docker on Ubuntu 16.04 server in this tutorial.
- Docker can be installed directly from Ubuntu repositories but that may not be the latest version of Docker engine. To install the latest and greatest version, we will install it from official Docker repository.
- Create one Ubuntu EC2 16.4 instanace on AWS.
- Login with pem file with default user called Ubuntu
- Change Ubuntu user to root user
- sudo -i

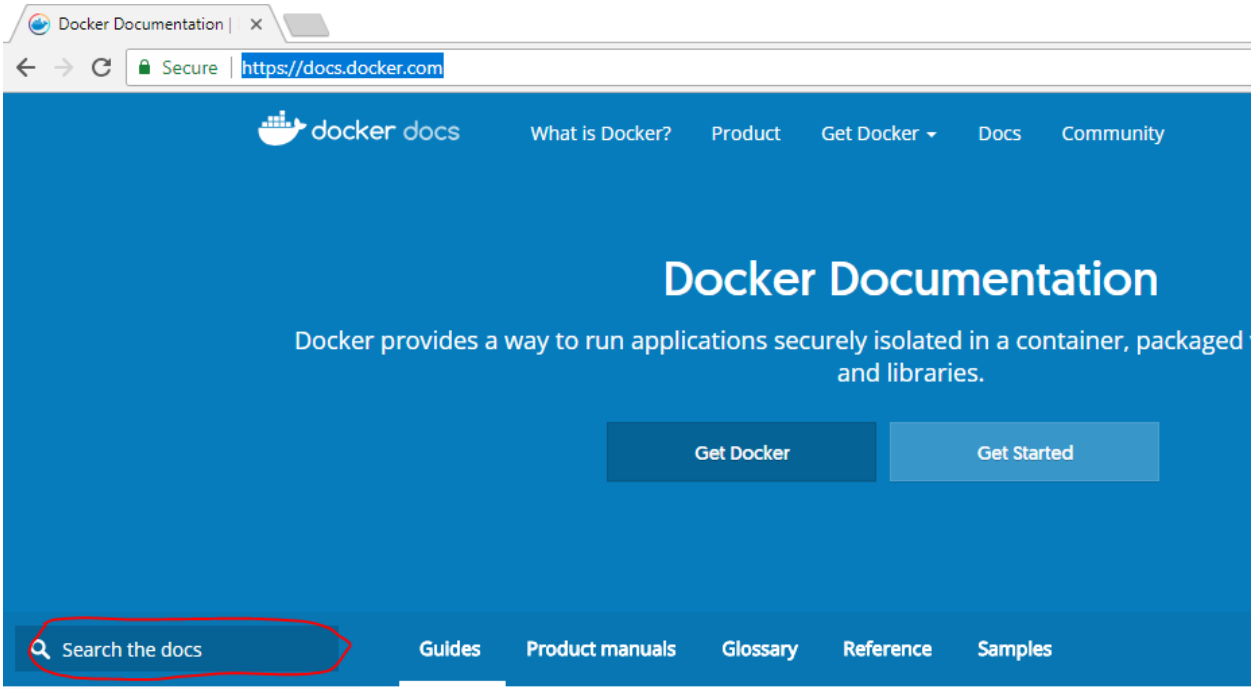
Uninstall old versions :

Older versions of Docker were called docker or docker-engine. If these are installed, uninstall them:

$ sudo apt-get remove docker docker-engine

Go to this site, go to google and type docker documentation and click on below website.

https://docs.docker.com/



There just type Ubuntu installation and click on below link,

https://docs.docker.com/install/linux/docker-ce/ubuntu/

Uninstall old versions

Older versions of Docker were called docker or docker-engine. If these are installed, uninstall them:

```
$ sudo apt-get remove docker docker-engine docker.io
```

Install Docker CE :

Install using the repository

Before you install Docker CE for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

SET UP THE REPOSITORY

1. Update the apt package index:

```
$ sudo apt-get update
```

2. Install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  software-properties-common
```

3. Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
```

4. Run following command

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

INSTALL DOCKER CE

1. Update the apt package index.

```
$ sudo apt-get update
```

2. Install the *latest version* of Docker CE, or go to the next step to install a specific version:

```
$ sudo apt-get install docker-ce -y
```

- Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

    - ✓ sudo systemctl status docker
    - ✓ sudo service docker status
- Docker commands can be executed by root user or by providing sudo. We can run docker command with normal user as well, to do it we need to add the user into the docker group.
    - ✓ sudo usermod -aG docker <user name>
    - ✓ ex : sudo usermod –aG docker ubuntu
- You need to logout and login to reflect the changes.
    - ✓ Exit form root
    - ✓ Now we will be in user Ubuntu
    - ✓ Exit from Ubuntu
    - ✓ And now just relogin with user name called ubuntu with pem file of our ec2 instance.
    - ✓ Or else one more way also we can add users to docker group
    - ✓ sudo –i
    - ✓ vi /etc/group
    - ✓ 

    - ✓ We can add another user with camma separated value.

- Run Docker command to check if it’s working.
    - ✓ docker –version
- When you install docker engine you get two components.
    - ✓ Docker Client
    - ✓ Docker Engine

Docker Engine’s Big Picture

Let’s quickly feel and taste the docker engine before we dive deep into it.

Broadly there are two areas where we operate in docker engines.

- Docker Images
- Docker containers

Images As of now you can think images as vagrant boxes. It's very much different from the vm images but it will feel as same initially. Vagrant boxes are stopped state of a VM and Images and stopped state of containers.

Run docker images command.

$ docker images

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest 7b9b13f7b9c0 2 weeks ago 118MB

This command will list the downloaded images on your machine, so you won't see anything now in the output. We need to download some images, in docker world we call it Pulling an image.

So where does it pull the image from? Again, same analogy as vagrant boxes. We download the vagrant boxes from vagrant cloud, docker images are downloaded from Docker Registries, the most famous docker registry is DockerHub. There are other registries as well, from google, redhat etc.

Containers

Now that we have an image pulled locally on our Docker host, we can use the docker run command to launch a container from it.

$ docker run -it ubuntu:latest /bin/bash

root@b8765d3a67a9:/#

Look closely at the output form the command above. You should notice that your shell prompt has changed. This is because your shell is now attached to the shell of the new container - you are literally inside of the new container!

Let's examine that docker run command.

- ✓ docker run tells the Docker daemon to start a new container.
- ✓ The -it flags tell the daemon to make the container interactive and to attach our current shell to the shell of the container.
- ✓ Next, the command tells Docker that we want the container to be based on the ubuntu:latest image.
- ✓ We tell it to run the /bin/bash process inside the container.

- ✓ Run the following ps command from inside of the container to list all running processes

Lets work with Images and few containers :

Now we don't have any images here so now am just logging with my default user name called Ubuntu with pem file and running few docker commands.

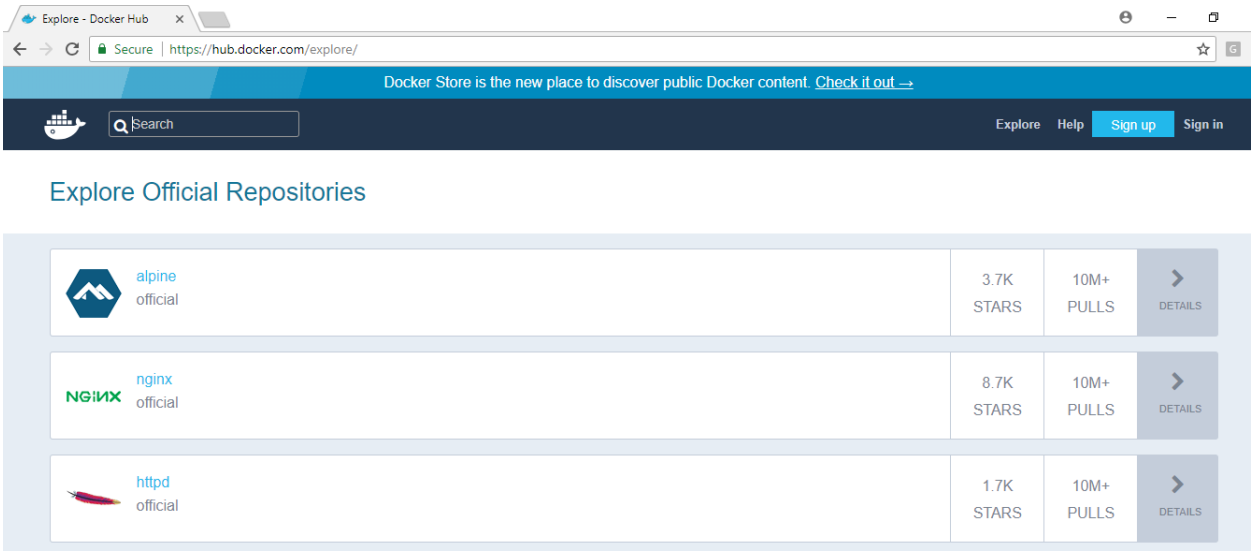
docker images

```
ubuntu@ip-172-31-33-226:~$ docker images
Got permission denied while trying to connect to the Docker daemon socket at unix
:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.35/images/json:
 dial unix /var/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-33-226:~$ sudo -i
root@ip-172-31-33-226:~# usermod -aG docker ubuntu
root@ip-172-31-33-226:~# id ubuntu
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),24(cdrom
),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),110(lxd
),999(docker)
root@ip-172-31-33-226:~# su - ubuntu
ubuntu@ip-172-31-33-226:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),24(cdrom
),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),110(lxd
),999(docker)
ubuntu@ip-172-31-33-226:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             S
IZE
```

- ❖ docker images : It will just list the what are images are downloaded to our local machine.
- ❖ For the VMs we have used vagrant and vagrant has its own cloud
- ❖ Same like For container Images we have very popular registry called dockerhub.
- ❖ Go to google and typedockerhub
- ❖ https://hub.docker.com/

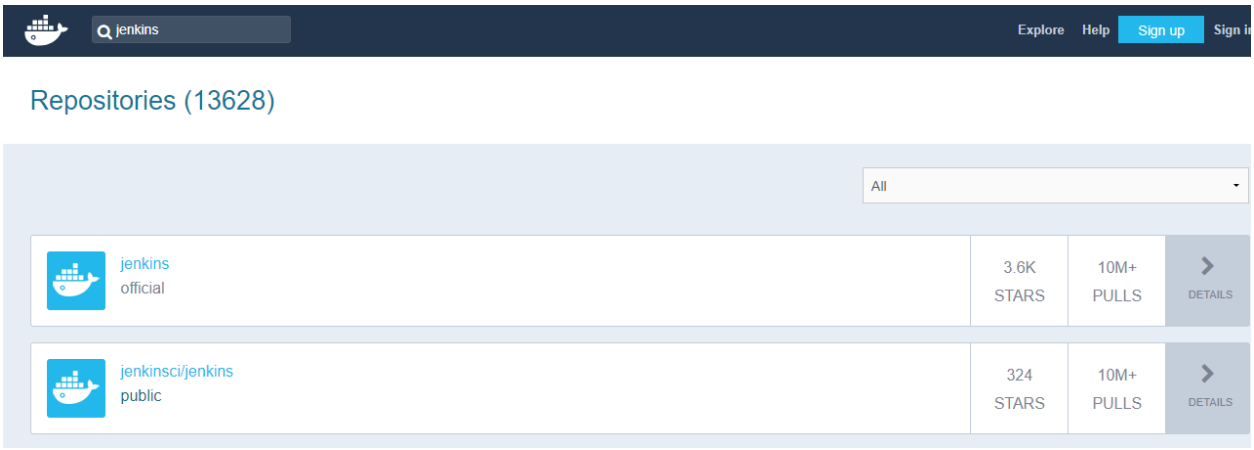
Docker HUB :

- ❖ Here we can see lot of images are here.
- ❖ These images are free to be download and we can use those images.
- ❖ And we can cutomise them as per our requirement.
- ❖ And then we can create containers from all these images.



We can search for any kind of image to create container

So we will search for Jenkins image from docker hub first



And we can pull Jenkins images from docker hub to our local machine.

docker pull Jenkins

Pull is a command where it can pull image from docker hub registry to our local machine

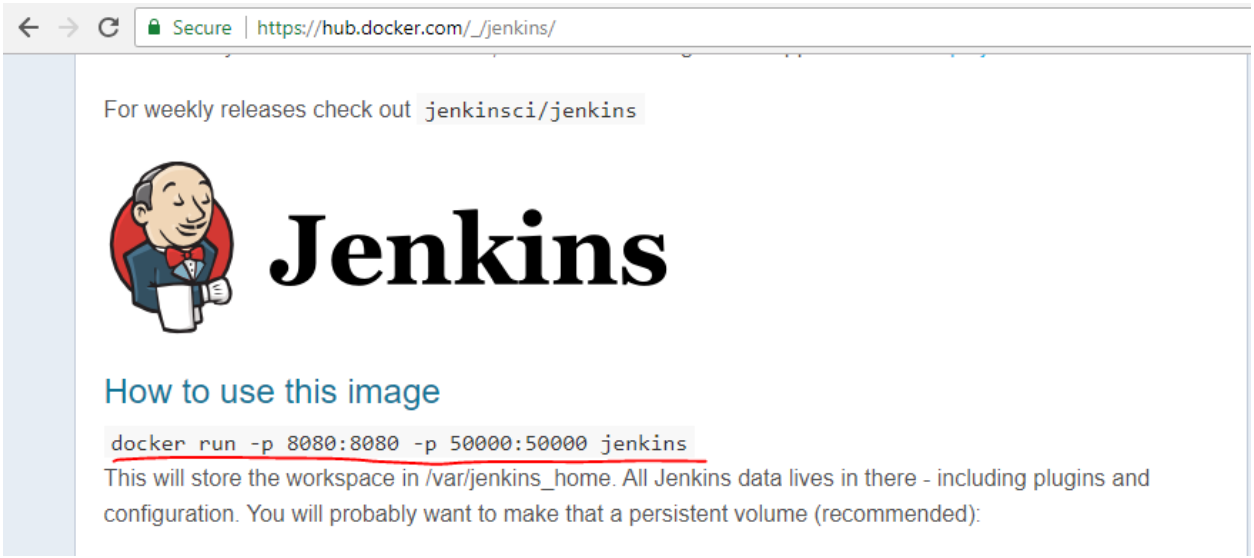
Run is a command where image can run and become a container

```
root@ip-172-31-33-226:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
root@ip-172-31-33-226:~# docker pull jenkins
Using default tag: latest
latest: Pulling from library/jenkins
723254a2c089: Extracting  23.86MB/45.12MB
abe15a44e12f: Download complete
409a28e3cc3d: Download complete
503166935590: Download complete
043a12c29ea4: Download complete
303620452447: Download complete
c61f95baa024: Download complete
3f2018472a1f: Downloading  29.81MB/182.9MB
a25f8a69c882: Waiting
ef0799915650: Waiting
d9a52178f3f2: Waiting
21d8e85eda47: Waiting
```

- In this Jenkins image what ever dependencies are needed to run Jenkins like java and all everything can be available inside of this image.
- So we don't do any configuration here everything can come up with our Jenkins image from docker hub registry

How to run this Image :

We just need to click Jenkins image in docker hub



docker run means : Create a container

```
root@ip-172-31-33-226:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jenkins             latest              5fc84ab0b7ad        8 weeks ago         809MB
root@ip-172-31-33-226:~#
```

Docker image can have repository name and TAG means version of image and ID and created data and size of image can be listed there.

docker run –p 8080:8080 –p 50000:50000 jenkins

```
root@ip-172-31-33-226:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jenkins             latest              5fc84ab0b7ad        8 weeks ago         809MB
root@ip-172-31-33-226:~# docker run -p 8080:8080 -p 50000:50000 jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
Feb 08, 2018 3:15:25 AM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file /tmp/winstone/jenkins.war
Feb 08, 2018 3:15:25 AM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Logging initialized @843ms
Feb 08, 2018 3:15:25 AM winstone.Logger logInternal
INFO: Beginning extraction from war file
Feb 08, 2018 3:15:27 AM org.eclipse.jetty.util.log.JavaUtilLog warn
WARNING: Empty contextPath
Feb 08, 2018 3:15:27 AM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: jetty-9.2.z-SNAPSHOT
Feb 08, 2018 3:15:28 AM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
Jenkins home directory: /var/jenkins_home found at: EnvVars.masterEnvVars.get("JENKINS_HOME")
```

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

572dcd7b307744958fa0834e61df4eeb

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************

Feb 08, 2018 3:15:41 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Feb 08, 2018 3:15:41 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Feb 08, 2018 3:15:42 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Feb 08, 2018 3:15:42 AM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 8,839 ms
```

That’s it we can access our Jenkins now

Here To access our Jenkins with my host ip and the port number 8080 we should be enable port 8080 on ec2 server so here we can allow 8080 port number or if you want we can give all traffic enabled for my ip since no need to change port number all the time.

But don’t do that all traffic In real time since it is our personal means it is ok.

Goto browser and type ip with 8080 port number

Here we don't need to pull before running Jenkins image we can directly run the image. When we run image withour pull it wil check image is already downloaded or not if not downloaded it is going to be download and running the image so we need to run only one single command to set up our Jenkins with docker images.

Now It is running foreground later we will see how can we run images background.

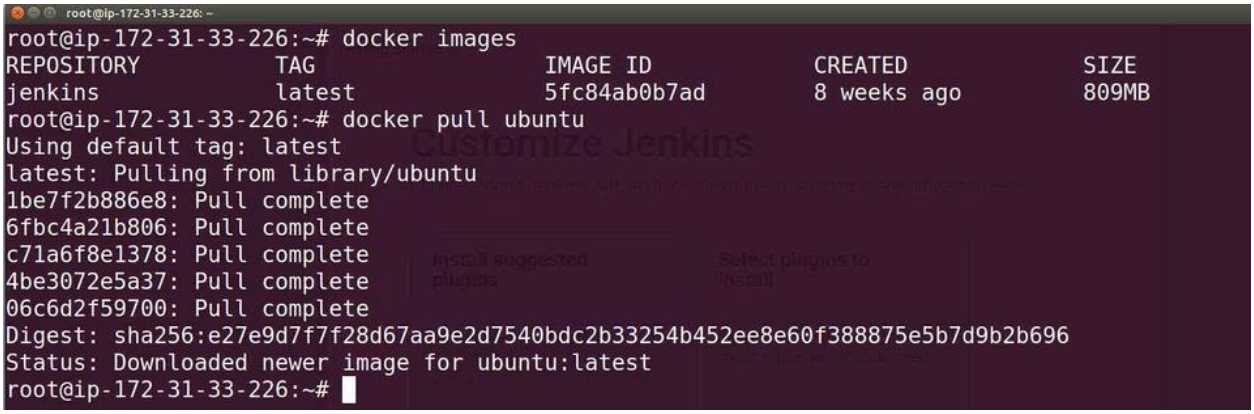
So generally if we need Jenkins set up

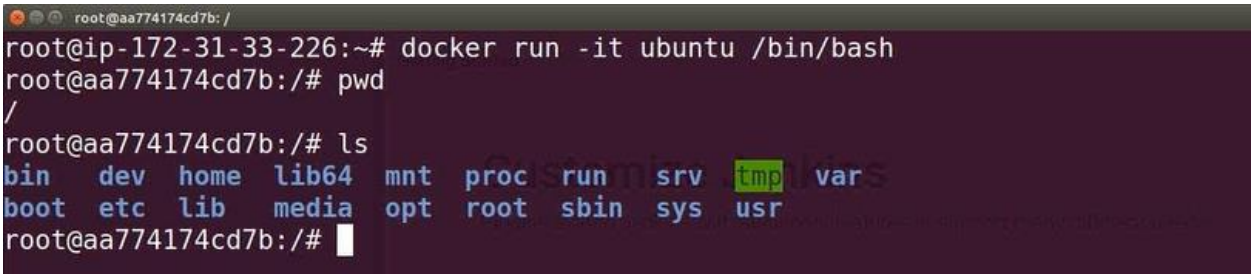
1. We have create system
2. We have to install java
3. We have to install Jenkins
4. And we have to install all dependencies

But now we just need to run single command that's how docker is so powerfull.

Now we can go with some other image called Ubuntu

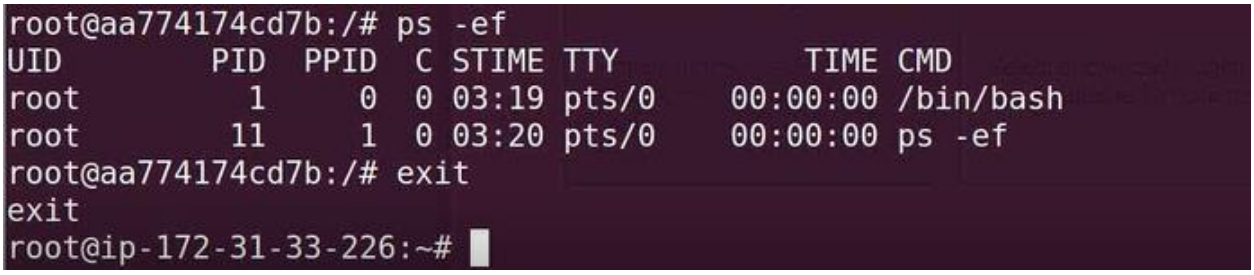
It is not actually Ubuntu operating system but when we see look and full can be it is a Ubuntu system.





It is very fast to create our Ubuntu service to act as a Ubuntu system.

But it is not Ubuntu operating system it is just a service running as a process

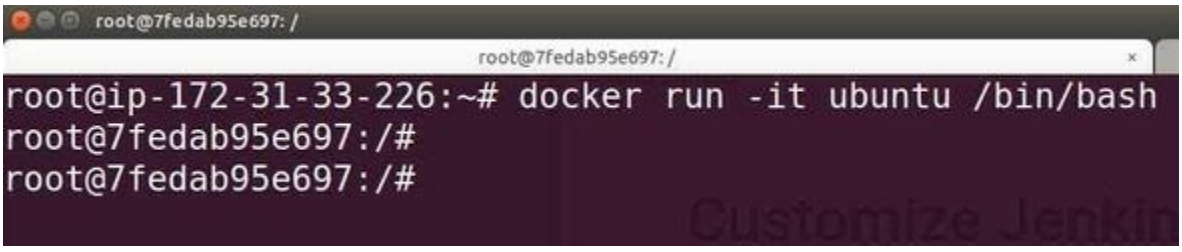


When we say exit container can be killed and we just came out from the container.

docker ps : running containers

docker ps –a : It shows all the containers which were dead also

when we re run the Ubuntu images here it is not going to be start existing container, It is going to be create a new container like below,



We can update inside

apt-get update

We can install any tool

apt-get install git

But it wil have minimum things it wont have all the things like operating system

Service command can be available

Ifconfig cant be available.

Now if we press control+pq that is going to be detaching from the shell container still can be run in background.

docker ps

```
root@7fedab95e697:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:22 pts/0    00:00:00 /bin/bash
root       226     1  0 03:24 pts/0    00:00:00 ps -ef
root@7fedab95e697:/# root@ip-172-31-33-226:~#
root@ip-172-31-33-226:~#
root@ip-172-31-33-226:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
       NAMES
7fedab95e697        ubuntu              "/bin/bash"         About a minute ago   Up About a minute
       unruffled_lamarr
root@ip-172-31-33-226:~#
```

Container will get

ID

IMAGE --- from which image got created

COMMAND --- starting command or initial command

CREATED ---- created time

STATUS --- up or down

PORTS ---- any ports attached

NAME --- container name created automatically by docker engine

But we can name our container that is very important to name our container.

```
root@ip-172-31-33-226:~# docker run -it --name myfrstcont ubuntu /bin/bash
root@48b45559717a:/# root@ip-172-31-33-226:~#
root@ip-172-31-33-226:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
      NAMES
48b45559717a        ubuntu              "/bin/bash"         7 seconds ago       Up 6 seconds
      myfrstcont
7fedab95e697        ubuntu              "/bin/bash"         2 minutes ago       Up 2 minutes
      unruffled_lamarr
root@ip-172-31-33-226:~#
```

Every time when we run image it is going to create a new container

docker ps : It wil just display running containers

How to attach conatiners :

```
      NAMES
48b45559717a        ubuntu              "/bin/bash"         40 seconds ago      Up 39 seconds
      myfrstcont
7fedab95e697        ubuntu              "/bin/bash"         3 minutes ago       Up 3 minutes
      unruffled_lamarr
root@ip-172-31-33-226:~# docker exec myfrstcont ls /
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
root@ip-172-31-33-226:~# docker exec -it myfrstcont /bin/bash
root@48b45559717a:/#
```

exec : It is a command to execute any commands on running containers.

First example we just run ls command

Second example we just done attaching inside of bash shell means accessed our running containers.

```
root@ip-172-31-33-226:~# docker exec -it myfrstcont /bin/bash
root@48b45559717a:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:25 pts/0    00:00:00 /bin/bash
root        13     0  0 03:26 pts/1    00:00:00 /bin/bash
root        21    13  0 03:27 pts/1    00:00:00 ps -ef
root@48b45559717a:/# exit
exit
root@ip-172-31-33-226:~# docker exec myfrstcont ls^C
root@ip-172-31-33-226:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
      NAMES
48b45559717a        ubuntu              "/bin/bash"         2 minutes ago       Up 2 minutes
      myfrstcont
7fedab95e697        ubuntu              "/bin/bash"         4 minutes ago       Up 4 minutes
      unruffled_lamarr
root@ip-172-31-33-226:~# docker exec 48b45559717a ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 03:25 pts/0    00:00:00 /bin/bash
root        22     0  0 03:27 ?        00:00:00 ps -ef
root@ip-172-31-33-226:~#
```

As long as one process is running inside a container so that container can exist as a running container.

Stop and Start containers :

docker stop <container id >

```
root@ip-172-31-33-226:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
      NAMES
48b45559717a        ubuntu              "/bin/bash"         4 minutes ago       Up 4 minutes
      myfrstcont
7fedab95e697        ubuntu              "/bin/bash"         6 minutes ago       Up 6 minutes
      unruffled_lamarr
root@ip-172-31-33-226:~# docker stop 48b45559717a
48b45559717a
root@ip-172-31-33-226:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
      NAMES
7fedab95e697        ubuntu              "/bin/bash"         7 minutes ago       Up 7 minutes
      unruffled_lamarr
root@ip-172-31-33-226:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
 PORTS               NAMES
48b45559717a        ubuntu              "/bin/bash"              4 minutes ago       Exited (0) 5 seconds ago
                    myfrstcont
7fedab95e697        ubuntu              "/bin/bash"              7 minutes ago       Up 7 minutes
                    unruffled_lamarr
aa774174cd7b        ubuntu              "/bin/bash"              10 minutes ago      Exited (0) 8 minutes ago
                    frosty_austin
90964d3aa4e8        jenkins             "/bin/tini -- /usr/l…"   14 minutes ago      Exited (130) 12 minutes ago
                    quizzical_hypatia
root@ip-172-31-33-226:~#
```

How to change name of the container

docker run --name myname imagename

if you want only container ids to display

docker ps -a -q

if you want only some formats to display

docker ps --format "{{.ID}}: {{.Command}}" –a

docker ps --format "{{.ID}}: {{.Names}}" -a

docker ps --format "{{.Names}}" -a

How to remove containers

docker rm <container id or container name>

How to remove containers forcefully

docker rm -f <container id>

For multiple container removal

docker rm <container id> <container id> <container id> <container id>

If you want to remove more than 200 containers

docker ps -a -q

docker rm $(docker ps -a -q)

docker rm 'docker ps -a -q'

docker ps --- it is only for running containers

docker ps -a --- it is for all running and exited containers

for exited container ?

docker ps -a | grep 'Exited'

docker ps -a | grep 'Exited' | awk '{print $1}'

if suppose some container may have name called exited so that case that container also wil get listed

For any kind of help you can use man command

man docker ps

docker ps --filter status=exited

if i want only container id can use below command

docker ps --filter status=exited -q

so can we remove all exited conatiners now except running containers ?

docker rm 'docker ps --filter status=exited -q'

how to stop the containers

it is a graceful way

docker stop <container_id>

it is a forceful way

docker kill <container_id>

List of files we wil get inside of conatiner

docker run centos ls

if you just go to list of containers

docker ps -a

now you can identify which command have you run ls