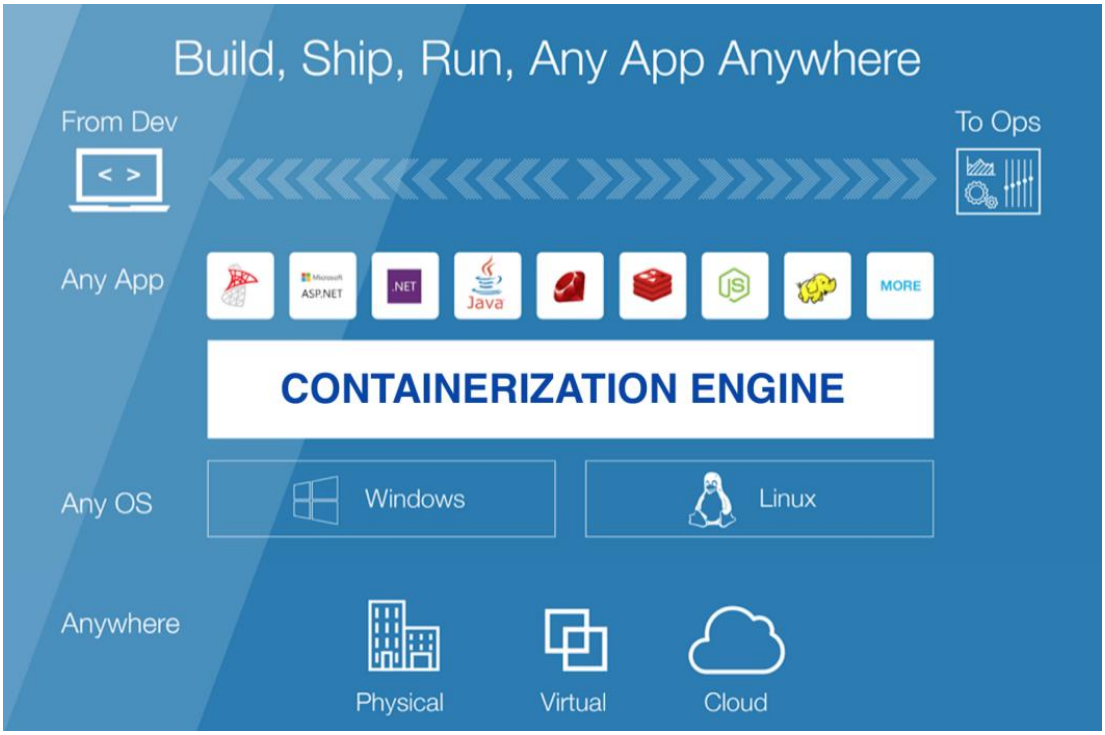
CONTAINARIZATION WITH DEVOPS

We have understood about importance of DevOps for software development. As we know DevOps is a practice which makes development and operation work together. Containerization is a part of DevOps which makes implementation of DevOps easier. It is a virtualization method to deploy and applications without using entire virtual machine for each application. More than one service can run on single host and can access the same Operating System Kernel.
You can just install Docker desktop for Windows pro and Docker toolbox for windows home and can start using Docker. No complications like setting up Virtual Machine. Windows Pro user can run Docker commands on PowerShell and Command Prompt. Windows Home user can run Docker command on their Docker toolbox terminal.

What is containerization?

A process of packaging of application, software with its retirements, framework and configuration file together required for the application. So that it can run in various environments easily. This is a very useful tool as it provides a facility to created once and run anywhere which makes easier to share between many groups and members.

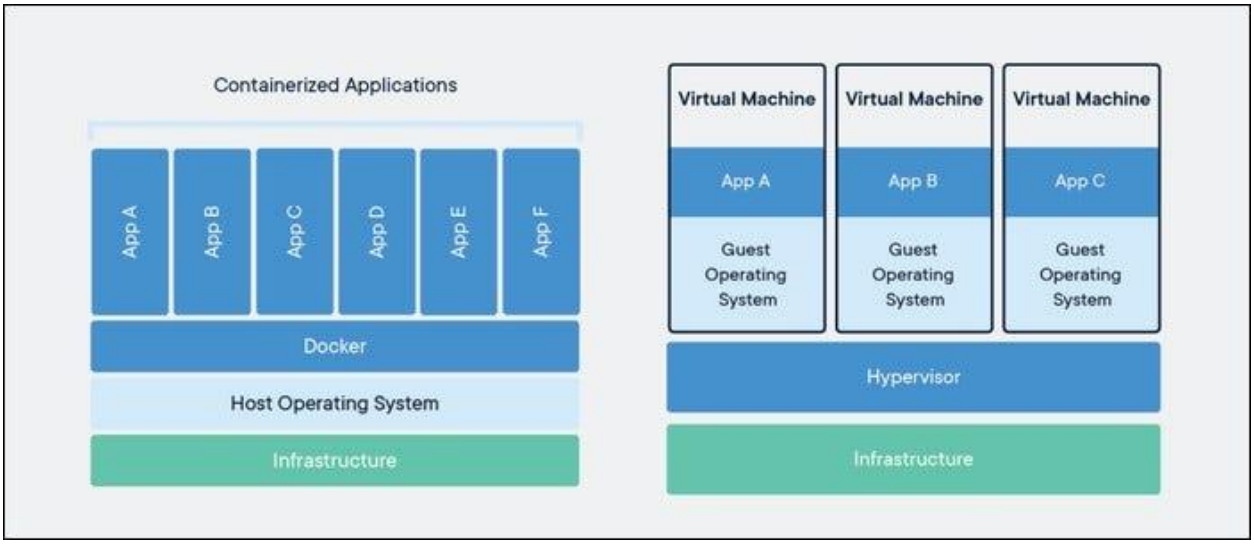


Some popular container providers are:-
Docker, Windows Server Container and Linux Containers like LXC and LCD. It took place the job of Virtual Machine in DevOps.

Virtual Machine VS Docker

How Virtual Machine and Docker container works

VM and container both need a host to run on. It can be your laptop, server in a data center or instance on the cloud.
In Virtual Machine when server is powered on and Hypervisor boots. When boots completed, it interacts with all physical resources on the system like CPU, RAM, Storage and NICs. Then the Hypervisor slice these hardware resources into virtual versions which further packaged into Virtual Machine (VM). Then we install Operating System and applications on each VMs.
In Containers when server is powered on Operating System boots. On top of the OS, the container engine i.e. Docker is installed. Container engine slice the OS resources like process tree, filesystem and network stack into Containers. Each container is small and is like a real OS. We can run an application in each container.



1. A Virtual Machine works occupies whole OS with its own memory and process inside the host Operating System and needs a proper amount of hardware. It takes lots of time as well as efforts and resources just for setting up the environment. Whereas Docker executed with Docker Engine as container. Containers are smaller, and enable faster with better performance and with greater compatibility.

2. Container is faster, and uses less resources and can be heavy as much user wants and can be launched just in seconds. Whereas A Virtual Machine could take time to create and launch.

3. Docker containers are just user space of Operating System whereas, Virtual Machine are user space and Kernel Space of Operating System

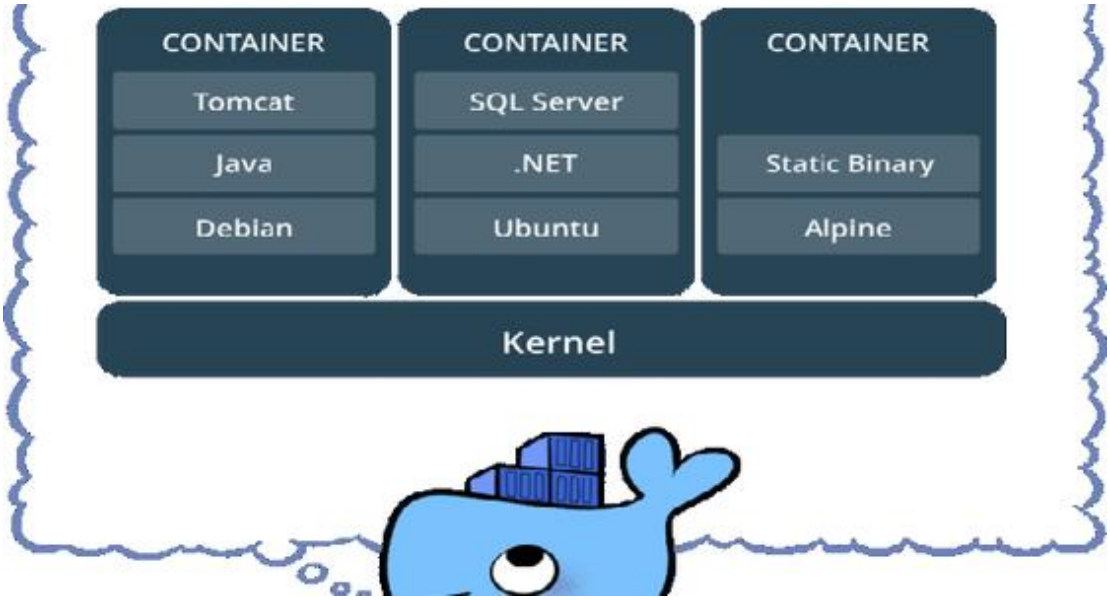
4. Docker Containers share the host Operating System Kernel while running whereas, Virtual Machine shares hardware resource from the host

5. In Docker multiple workloads can run on single Operating System whereas, In Virtual Machine each workload needs a complete Operating System to work

6. Virtual Machine's Hypervisor perform Hardware Virtualization whereas Containers perform Operating System Virtualization.

What is Docker?

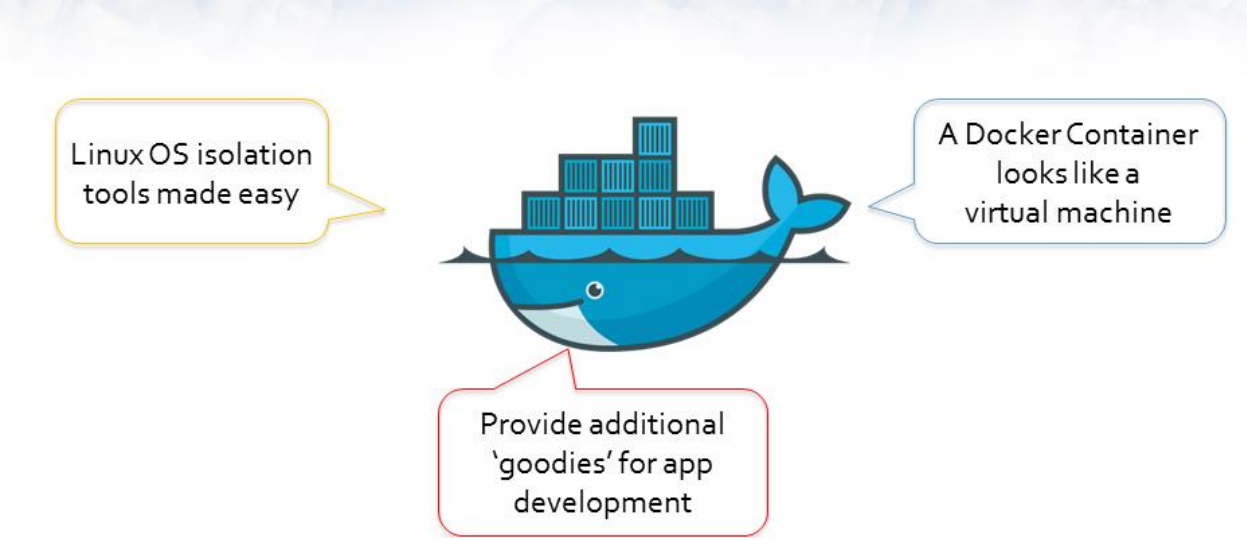
Docker is a technology that provides containerization which simplifies building, testing, securing and deploying application within container. It helps for easy and faster delivery of services. Containerization is a packaging an application with all of their dependencies into a container. This makes easier to analyze the output by running on different machines other than Local Environment or machine where to be tested. Docker file is easy to transfer and tested among the developers.



An application in Docker Containers can run on multiple operating systems. Running apps over Docker Containers and then deploying anywhere is more in reality now.
Companies like Facebook, Netflix, Google and Salesforce adopted Docker to improve utilization of computer resources.

To implement containerization successfully using Docker:-
1. Code in repository by developer.
2. Proper Packaging
3. Plugin requirements and dependencies
4. Create Container images using Docker
5. Shift to other environment
6. Use clouds like AWS OR Azure for deployment.

BENEFITS OF DOCKER



1. Multiple Cloud Platforms: - Containers can run on different cloud platforms like Amazon ECS, GCS, and Amazon DevOps Server

2. DevOps-friendly: - Containers are with environmental dependencies that make sure working of application developed in an environment works in another environment easily. Which will help developers and testers to work collaboration easily in DevOps.

3. No Separate Operating System needed: - Docker image utilize kernel of the host OS. So doesn't require different OS.

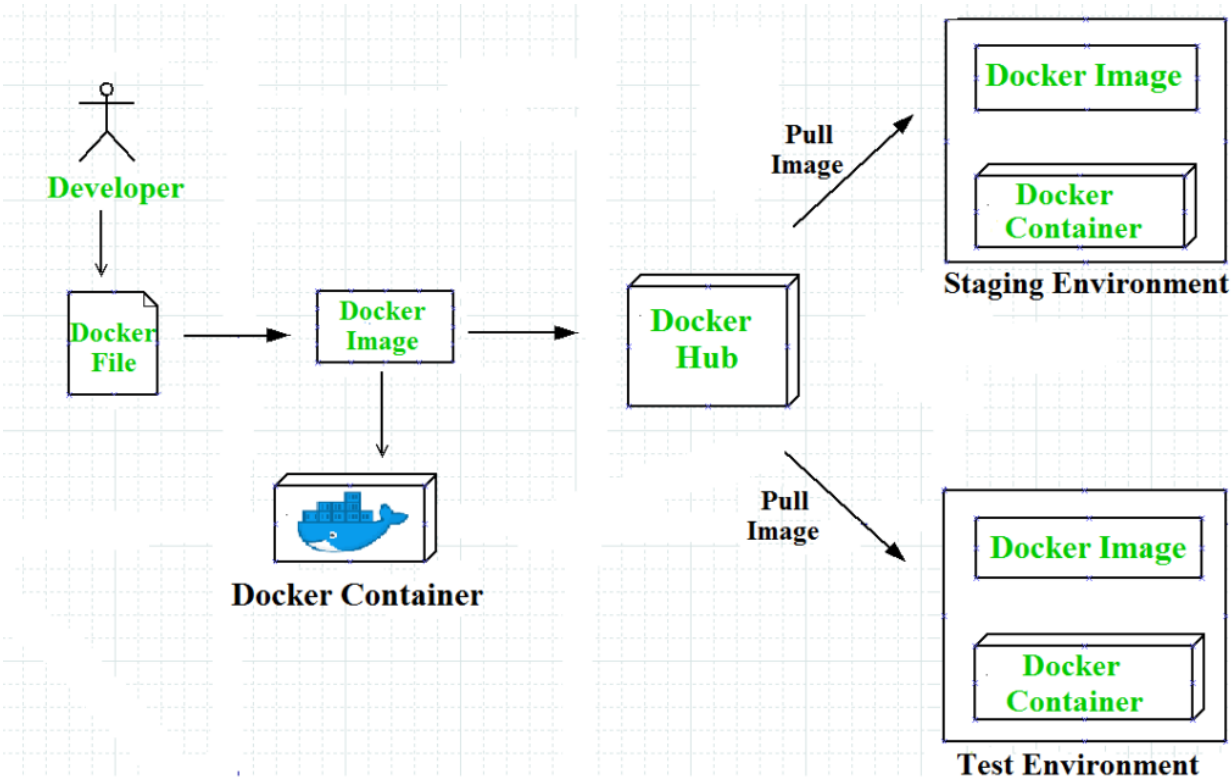
4. Faster Scalability: - Isolated containers can be scaled up faster.

5. Fast-Spinning of Apps: - Delivery is faster

6. Portable - Container can be deployed to a new system easily

7. Not Costly: - Containerization can support multiple containers on singular infrastructure. So, don't need to buy many tools, CPU, Memory and storage.

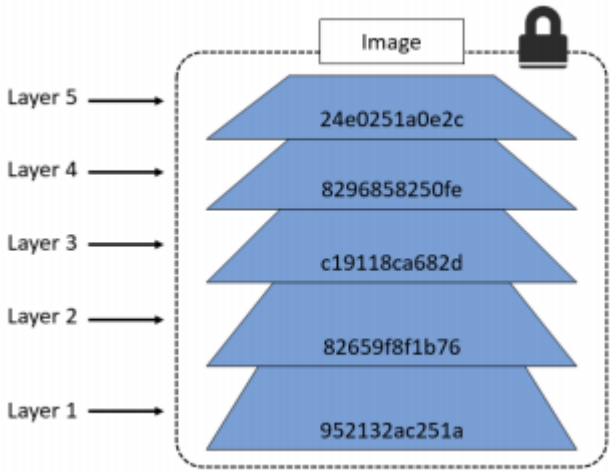
Docker Containers, Images, and Registries



+60

Docker Images-

A virtual or can say a blue print of an Operating system with services, and its dependencies of an application. Images are the connected read-only layers. Each layer represents a unified object.



We can create an Image of any Operating System by using “pull” command and can remove by using "rmi” command

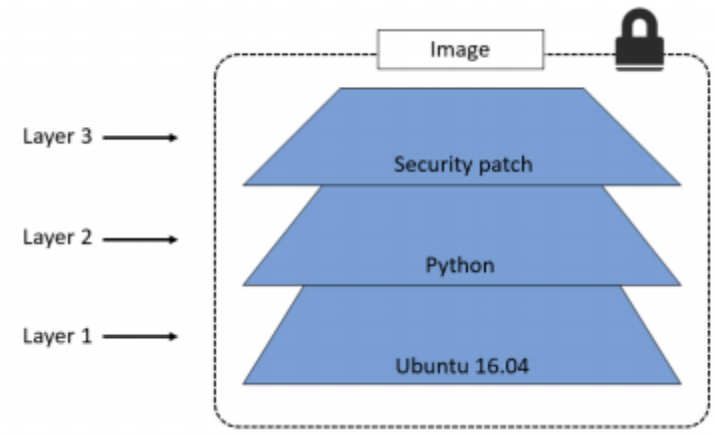
```
$ docker pull Ubuntu
$ docker rmi Ubuntu
```

```
7249723069d8: Pull complete
6465c437f020: Pull complete
954c67861e66: Pull complete
6a14c8afbb3a: Pull complete
ec070f7e511e: Pull complete
983246da862f: Pull complete
998d1854867e: Pull complete
Digest: sha256:878e055f96c90af9281fd859f7c69ac289e0178594ff36bbb85e53b78969
Status: Downloaded newer image for jenkins:latest
demo@ubuntuserver:~$
demo@ubuntuserver:~$
```

When we pull image we observe more than one “pull complete” while pulling. Each “Pull Complete” represents a layer of the image with their IDs. All inages starts with the base image and when we add something they don’t change anything from previous layers they just add another layer on the top.

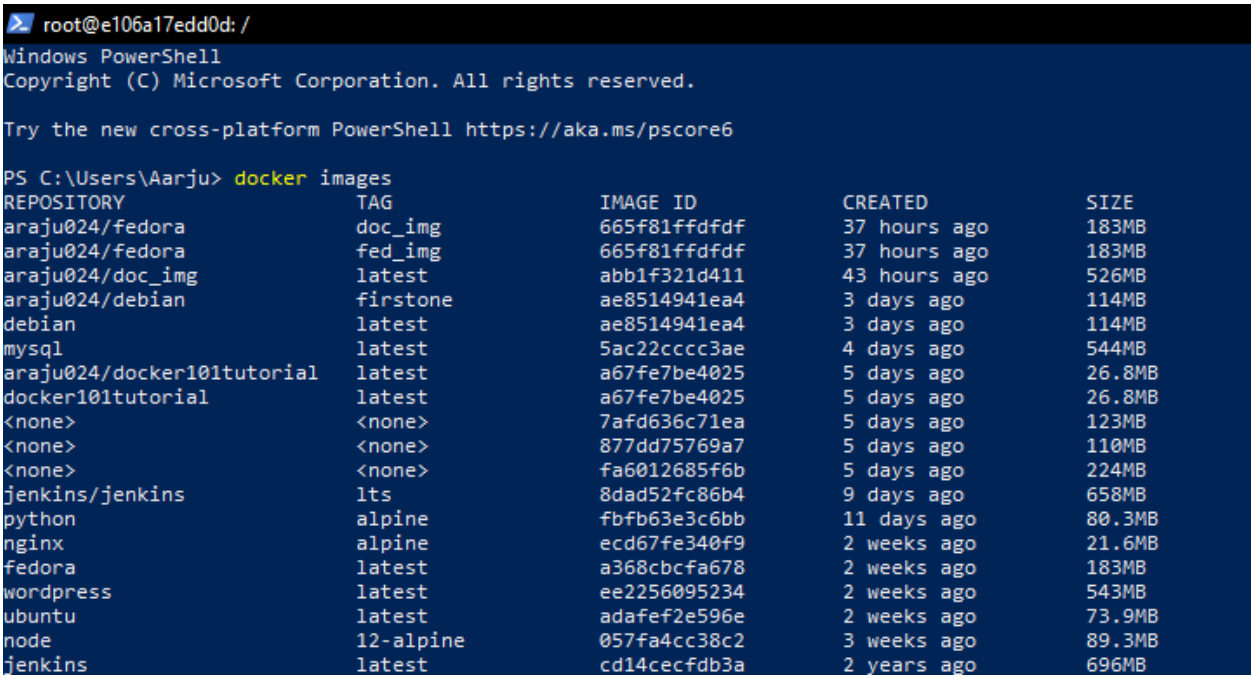
What are these ID’s ?
You must have thought about the ID with each layer. These are not just arandow number these are crypto ID that is a hash of the contents it contain. Changing content of the image will change the crypto ID. So it is easy to identify any changes made.

Eg :-
Create a new image of Ubuntu Linux. This will be first layer of the image also called “Base Layer”. Later when python package will be installed it will create second layer on top of the base layer. Then if you add any security package, third layer will be added on top and so on. Image is the combination of all layers.



This command will create a virtual Ubuntu Operating system and can remove the image.
You can list all images available on your system by using

```
$ docker images
```



```
root@e106a17edd0d: /
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Aarju> docker images
REPOSITORY                   TAG                 IMAGE ID            CREATED             SIZE
araju024/fedora              doc_img             665f81ffdfdf        37 hours ago        183MB
araju024/fedora              fed_img             665f81ffdfdf        37 hours ago        183MB
araju024/doc_img             latest              abb1f321d411        43 hours ago        526MB
araju024/debian              firstone            ae8514941ea4        3 days ago          114MB
debian                       latest              ae8514941ea4        3 days ago          114MB
mysql                        latest              5ac22cccc3ae        4 days ago          544MB
araju024/docker101tutorial   latest              a67fe7be4025        5 days ago          26.8MB
docker101tutorial            latest              a67fe7be4025        5 days ago          26.8MB
<none>                       <none>              7afd636c71ea        5 days ago          123MB
<none>                       <none>              877dd75769a7        5 days ago          110MB
<none>                       <none>              fa6012685f6b        5 days ago          224MB
jenkins/jenkins              lts                 8dad52fc86b4        9 days ago          658MB
python                       alpine              fbfb63e3c6bb        11 days ago         80.3MB
nginx                        alpine              ecd67fe340f9        2 weeks ago         21.6MB
fedora                       latest              a368cbcfa678        2 weeks ago         183MB
wordpress                    latest              ee2256095234        2 weeks ago         543MB
ubuntu                       latest              adafef2e596e        2 weeks ago         73.9MB
node                         12-alpine           057fa4cc38c2        3 weeks ago         89.3MB
jenkins                      latest              cd14cecfdb3a        2 years ago         696MB
```

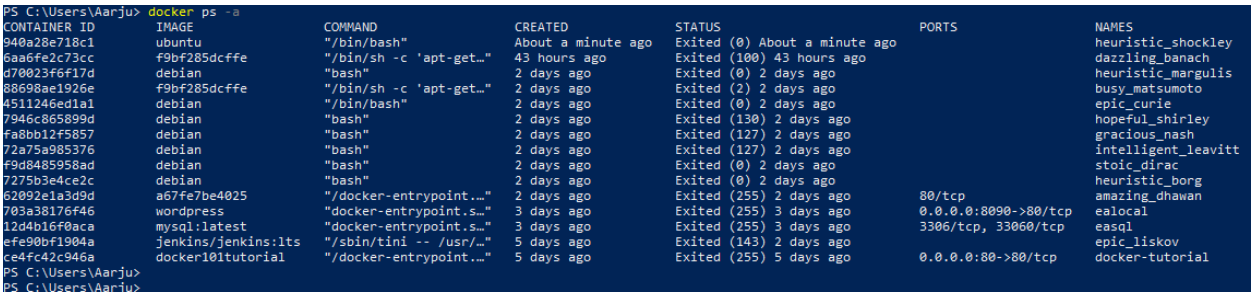
Docker Containers -

Docker container is a runtime instance of an Image. We can run multiple containers with single Image. Instance of the image is used to create a container. When the image run on Docker Engine using "run" command which creates container of the image. Command to run Ubuntu image created is

```
$ docker run Ubuntu
```

To list all the containers on the system

```
$ docker ps -a
```



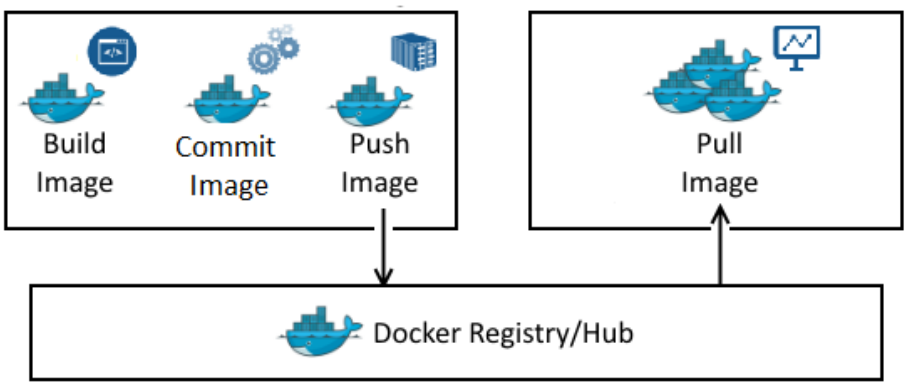
```
PS C:\Users\Aarju> docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS                        PORTS                    NAMES
940a28e718c1        ubuntu                 "/bin/bash"              About a minute ago  Exited (0) About a minute ago                          heuristic_shockley
6aa6fe2c73cc        f9bf285dcffe           "/bin/sh -c 'apt-get…"   43 hours ago        Exited (100) 43 hours ago                          dazzling_banach
d70023f6f17d        debian                 "bash"                   2 days ago          Exited (0) 2 days ago                              heuristic_margulis
88698ae1926e        f9bf285dcffe           "/bin/sh -c 'apt-get…"   2 days ago          Exited (2) 2 days ago                              busy_matsumoto
4511246ed1a1        debian                 "/bin/bash"              2 days ago          Exited (0) 2 days ago                              epic_curie
7946c865899d        debian                 "bash"                   2 days ago          Exited (130) 2 days ago                            hopeful_shirley
fa8bb12f5857        debian                 "bash"                   2 days ago          Exited (127) 2 days ago                            gracious_nash
72a75a985376        debian                 "bash"                   2 days ago          Exited (127) 2 days ago                            intelligent_leavitt
f9d8485958ad        debian                 "bash"                   2 days ago          Exited (0) 2 days ago                              stoic_dirac
7275b3e4ce2c        debian                 "bash"                   2 days ago          Exited (0) 2 days ago                              heuristic_borg
62092e1a3d9d        a67fe7be4025           "/docker-entrypoint.…"   2 days ago          Exited (255) 2 days ago       80/tcp                   amazing_dhawan
703a38176f46        wordpress              "docker-entrypoint.s…"   3 days ago          Exited (255) 3 days ago       0.0.0.0:8090->80/tcp     ealocal
12d4b16f0aca        mysql:latest           "docker-entrypoint.s…"   3 days ago          Exited (255) 3 days ago       3306/tcp, 33060/tcp      easql
efe90bf1904a        jenkins/jenkins:lts    "/sbin/tini -- /usr/…"   5 days ago          Exited (143) 2 days ago                            epic_liskov
ce4fc42c946a        docker101tutorial      "/docker-entrypoint.…"   5 days ago          Exited (255) 5 days ago       0.0.0.0:80->80/tcp       docker-tutorial
PS C:\Users\Aarju>
PS C:\Users\Aarju>
```

We can work on Ubuntu OS bash by

```
$ docker run –it ubuntu /bin/bash
```

```
Windows PowerShell
PS C:\Users\Aarju>
PS C:\Users\Aarju> docker run -it ubuntu /bin/bash
root@e106a17edd0d:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@e106a17edd0d:/# ll
total 56
drwxr-xr-x   1 root root 4096 Jul 25 04:16 ./
drwxr-xr-x   1 root root 4096 Jul 25 04:16 ../
-rwxr-xr-x   1 root root    0 Jul 25 04:16 .dockerenv*
lrwxrwxrwx   1 root root    7 Jul  3 01:56 bin -> usr/bin/
drwxr-xr-x   2 root root 4096 Apr 15 11:09 boot/
drwxr-xr-x   5 root root  360 Jul 25 04:16 dev/
drwxr-xr-x   1 root root 4096 Jul 25 04:16 etc/
drwxr-xr-x   2 root root 4096 Apr 15 11:09 home/
lrwxrwxrwx   1 root root    7 Jul  3 01:56 lib -> usr/lib/
lrwxrwxrwx   1 root root    9 Jul  3 01:56 lib32 -> usr/lib32/
lrwxrwxrwx   1 root root    9 Jul  3 01:56 lib64 -> usr/lib64/
lrwxrwxrwx   1 root root   10 Jul  3 01:56 libx32 -> usr/libx32/
drwxr-xr-x   2 root root 4096 Jul  3 01:57 media/
drwxr-xr-x   2 root root 4096 Jul  3 01:57 mnt/
drwxr-xr-x   2 root root 4096 Jul  3 01:57 opt/
dr-xr-xr-x 126 root root    0 Jul 25 04:16 proc/
drwx------   2 root root 4096 Jul  3 02:00 root/
drwxr-xr-x   1 root root 4096 Jul  6 21:56 run/
lrwxrwxrwx   1 root root    8 Jul  3 01:56 sbin -> usr/sbin/
drwxr-xr-x   2 root root 4096 Jul  3 01:57 srv/
dr-xr-xr-x  12 root root    0 Jul 25 04:16 sys/
drwxrwxrwt   2 root root 4096 Jul  3 02:00 tmp/
drwxr-xr-x   1 root root 4096 Jul  3 01:57 usr/
drwxr-xr-x   1 root root 4096 Jul  3 02:00 var/
root@e106a17edd0d:/# ls-a
bash: ls-a: command not found
root@e106a17edd0d:/# ls -a
.   .dockerenv  boot  etc   lib    lib64   media  opt   root  sbin  sys  usr
..  bin         dev   home  lib32  libx32  mnt    proc  run   srv   tmp  var
root@e106a17edd0d:/# exit
exit
PS C:\Users\Aarju>
```

Docker Registry -



Docker Hub is a registry service on cloud which uploads your Docker built images and also downloads images built by others. Let's see pulling and uploading Jenkins

1. Let's pull Jenkins from Docker Hub from https://hub.docker.com/_/jenkins

```
$docker pull jenkins
```

2. To run Jenkins on 8090 port

```
$docker run -p 8090:8080 -p 50000:50000 jenkins
```

3. You can run on "localhost/8090" on browser to run jenkins on your system

We will see how to upload an image:-

1. Create repository on Docker hub by using "Create Repository" button by giving any name say "new_repo" here

2. Pull that repository on your system by
```
$docker pull username/new_repo
```

3. Login on system by giving credentials of docker hub. This command will ask your username and password
```
$docker login
```

5. Provide a tag to the image
```
$ docker tag ImageID(ab0971d23) username/new_repo:1.0
```

6. Now push to hub to get displayed for others too
```
$docker push username/new_repo:1.0
```

Your docker will be uploaded on the docker hub as public repository. This can be pulled by anyone from your docker hub account to use further.